VILNIAUS UNIVERSITETO
KAUNO FAKULTETAS

SOCIALINIŲ MOKSLŲ IR TAIKOMOSIOS INFORMATIKOS INSTITUTAS
Finansų technologijų studijų programa
Kodas 6211BX022

DOMANTAS KARAŠA

MAGISTRO BAIGIAMASIS DARBAS

MAŠININIU MOKYMUSI PAGRĮSTOS KLIENTŲ RIZIKOS VERTINIMO STRATEGIJOS AVIACIJOS PRAMONĖJE

Kaunas 2025

VILNIAUS UNIVERSITETO

KAUNO FAKULTETAS

SOCIALINIŲ MOKSLŲ IR TAIKOMOSIOS INFORMATIKOS INSTITUTAS

DOMANTAS KARAŠA

MAGISTRO BAIGIAMASIS DARBAS

MAŠININIU MOKYMUSI PAGRĮSTOS KLIENTŲ RIZIKOS VERTINIMO STRATEGIJOS AVIACIJOS PRAMONĖJE

Magistrantas

(parašas)

Darbo vadovas: Prof. Dr. Mantas Vaitonis

(parašas)

Recenzentas:

(parašas)

Kaunas 2025

TURINYS

PAVEIKSLĖLIŲ SĄRAŠAS

LENTELIŲ SĄRAŠAS

SANTRUPŲ SĄRAŠAS

1. **ORM** - *Online Reputation Management* (Interneto reputacijos valdymas)
2. **KYC** - *Know Your Customer* (Pažink savo klientą)
3. **AML** - *Anti-Money Laundering* (Pinigų plovimo prevencija)
4. **NLP** - *Natural Language Processing* (Natūralios kalbos apdorojimas)
5. **ML** - *Machine Learning* (Mašininis mokymasis)
6. **API** - *Application Programming Interface* (Programavimo sąsaja)
7. **CDD** - *Continuous Due Diligence* (Nuolatinis patikimumo patikrinimas)
8. **ICAO** - *International Civil Aviation Organization* (Tarptautinė civilinės aviacijos organizacija)
9. **FAA** - *Federal Aviation Administration* (Federalinė aviacijos administracija)
10. **EASA** - *European Union Aviation Safety Agency* (Europos Sąjungos aviacijos saugos agentūra)
11. **OFAC** - *Office of Foreign Assets Control* (Užsienio turto kontrolės tarnyba)
12. **AI** - *Artificial Intelligence* (Dirbtinis intelektas)
13. **TF-IDF** - *Term Frequency-Inverse Document Frequency* (Termino dažnis–atvirkštinis dokumento dažnis)
14. **CUDA** - *Compute Unified Device Architecture* (Vieningos įrenginių architektūros skaičiavimai)
15. **BERT** - *Bidirectional Encoder Representations from Transformers* (Dvipusės kodavimo reprezentacijos iš transformatorių)
16. **RoBERTa** - *A Robustly Optimized BERT Approach* (Optimizuotas BERT požiūris)
17. **VADER** - Valence Aware Dictionary and sEntiment Reasoner (Semantinės vertės analizės ir nuotaikos nustatymo algoritmas)
18. **GPT** - *Generative Pre-trained Transformer* (Generatyvinis iš anksto išmokytas transformatorius)
19. **EU** - *European Union* (Europos Sąjunga)
20. **UN** - *United Nations* (Jungtinės Tautos)

SANTRAUKA

Karaša, Domantas (2025). *Machine learning-based customer risk assessment strategies in the aviation industry*. MBA Graduation Paper. Kaunas: Vilnius University, Kaunas Faculty, Institute of Social Sciences and Applied Informatics. 104 p.

This master thesis explores the application of machine learning (ML) to customer risk assessment strategies in the aviation industry. The aviation sector, which is the backbone of global transport and trade, faces unique challenges such as regulatory compliance, reputation management and mitigating risk in high value transactions. Traditional approaches to managing these challenges are becoming increasingly inadequate due to the complexity of global networks, dynamic regulatory frameworks and vast amounts of data. Therefore, advanced technological solutions are needed to ensure operational and reputational integrity.

The study focuses on the integration of ML models into processes such as Know Your Customer (KYC) and Continuous Due Diligence (CDD), with an emphasis on reputational risk assessment. Building on methodologies used in the financial sectors, the study applies techniques such as Natural Language Processing (NLP) to analyse large datasets including social media, publicly available data and sanctions lists. Models such as GPT, BERT, RoBERTa and VADER are being used and developed to meet the specific needs of the aviation industry and provide greater efficiency in sentiment analysis, reputation assessment and risk classification.

The study describes a new ML-based framework for dynamic and efficient reputational risk assessment for aviation companies. The developed models demonstrate improved accuracy in risk identification and are able to adapt to rapidly changing data patterns and regulatory requirements. The implementation of these solutions aims to improve compliance, maintain stakeholder confidence and strengthen industry resilience in a competitive and interconnected global environment.

Finally, this research contributes to the academic and practical understanding of the use of ML to address critical issues in the aviation industry. The results not only highlight the potential of AI-driven tools, but also provide useful insights on how to improve KYC and risk management processes to ensure sustainable and safe operations.

ĮVADAS

Temos naujumas ir aktualumas

Aviacijos sektorius, kaip pasaulinės transporto ir ekonominės veiklos pagrindas, pasižymi unikaliu sudėtingumu dėl savo plataus tinklo, didelės operacijų apimties ir griežtos reguliavimo aplinkos. Šiame sektoriuje kasdien vykdomi įvairūs kompleksiniai sandoriai, apimantys skirtingus pasaulio regionus bei įtraukiantys daugybę suinteresuotų šalių. Toks kontekstas reikalauja ne tik tikslaus operacijų valdymo, bet ir didelio dėmesio klientų reputacijai, patikimumui bei reguliavimo reikalavimų laikymuisi.

Šioje srityje svarbų vaidmenį atlieka principai, tokie kaip „Pažink savo klientą“ (angl. *Know Your Customer – KYC)* bei nuolatiniai klientų reputacijos ir elgesio patikrinimai (angl. *Due Diligence*). Jie užtikrina skaidrumą, mažina suinteresuotųjų šalių keliamą riziką ir padeda išvengti galimų reguliavimo pažeidimų. Tačiau tradiciniai šių procesų metodai dažnai yra nepakankami dėl augančio duomenų kiekio, jų sudėtingumo ir greitai kintančio pasaulinio reguliavimo konteksto.

Čia į pagalbą ateina mašininis mokymasis, kuris leidžia efektyviau analizuoti įvairius didelių duomenų rinkinius, nustatyti subtilius modelius bei įžvelgti galimą reputacijos riziką. Šiuolaikiniai metodai leidžia ne tik vertinti klientų patikimumą, bet ir prisitaikyti prie besikeičiančio jų elgesio bei reguliacinių reikalavimų. Tokie sprendimai ne tik didina aviacijos sektoriaus veiklos efektyvumą, bet ir padeda išlaikyti stiprią reputaciją bei patikimumą, kurie yra neatsiejami nuo šio sektoriaus ilgalaikės sėkmės.

Šiame darbe analizuojamas mašininio mokymosi pritaikymas aviacijos sektoriuje, siekiant efektyviau spręsti KYC ir reputacijos rizikos iššūkius. Nagrinėjami metodai ne tik įgalina inovatyvesnį požiūrį į reputacijos vertinimą, bet ir prisideda prie sektoriaus skaidrumo, reguliavimo laikymosi ir ilgalaikės patikimumo išlaikymo.

Darbo problema

Tradiciniai klientų reputacijos tikrinimo metodai aviacijoje dažniausiai paremti žmogiškaisiais ištekliais, o tai lemia, kad jiems sunku veiksmingai valdyti didžiulį duomenų sudėtingumą ir kiekį. Tokie metodai dažnai apima rankinius patikrinimus, paprastas taisykles ir iš anksto nustatytus algoritmus, kurie gali aptikti tik aiškiai nurodytus anomalijas ar sukčiavimo požymius. Deja, šie metodai susiduria su ribotomis galimybėmis, kai kalbama apie greitai besikeičiančius sukčiavimo modelius, kurie nuolat prisitaiko prie naujų saugumo priemonių. Be

to, rankiniai procesai yra lėti, brangūs ir reikalauja didelių žmogiškųjų išteklių, o tai ne tik padidina veiklos sąnaudas, bet ir apsunkina greitą atsaką į potencialius sukčiavimo atvejus.

Šiuolaikinėje aviacijos industrijoje, kur kasdien apdorojami milijonai duomenų įrašų iš viso pasaulio, tradiciniai metodai dažnai tampa nepakankami, nes negali apdoroti tokio masto duomenų srauto ir greitai reaguoti į grėsmes. Tai reiškia, kad tokios sistemos ne tik praleidžia naujas ir pažangias sukčiavimo taktikas, bet ir sukelia nemažai klaidingų signalų, kurie trukdo efektyviai identifikuoti tikrąsias grėsmes. Pavyzdžiui, jei sistema remiasi vien taisyklėmis, kurios nustato tam tikras ribas ar elgsenos modelius, tai sureaguoti laiku bei gauti greitas išvadas su įmonės veikla bei reputacija susietiems sprendimams atlikti gali būti sudėtinga ar neįmanoma.

Dėl šių apribojimų atsiranda poreikis pažangesnėms technologijoms, kurios galėtų veiksmingai analizuoti didelius duomenų kiekius ir atskleisti neaiškius modelius, būdingus sukčiavimui. Tik pažangūs analitiniai įrankiai, tokie kaip mašininis mokymasis, geba įžvelgti ir išmokti subtilias sukčiavimo ypatybes, kurios dažnai būna per sudėtingos ar dinamiškos tradiciniams metodams.

Darbo tikslas

Tikslas – sukurti universalų mašininio mokymosi modelį, specialiai pritaikytą rizikos vertinimo procesui aviacijos pramonėje, kuris leistų efektyviai ir tiksliai identifikuoti potencialias reputacijos žalos grėsmes, vertinant klientų elgseną ir reputaciją realiuoju laiku. Toks modelis būtų skirtas ne tik analizuoti žinomus elgesio modelius, bet ir dinamiškai prisitaikyti prie naujų, kintančių veiksnių. Aviacijos pramonėje, kurioje tarptautinių operacijų mastai yra ypač dideli, toks modelis užtikrintų sklandų ir efektyvų procesų valdymą.

Šis modelis yra kuriamas specialiai internetinės reputacijos tyrimui ir skirtas taikyti „Pažink savo klientą“ procesams bei nuolatiniam klientų patikrinimui. Modelis turi gebėti integruoti ir analizuoti duomenis iš įvairių šaltinių, tokių kaip socialinių tinklų reputacija, viešai prieinami duomenys, sankcijų sąrašai ir istorinis klientų elgesys, siekiant įvertinti potencialias rizikas ir užtikrinti klientų tinkamumą.

Šio modelio tikslas nustatyti kliento reputacijos rizikos lygį, suteikiant tikslias ir laiku pateikiamas įžvalgas, kurios padeda priimti internetine reputacija pagrįstus sprendimus. Taikant šį modelį, aviacijos pramonės įmonės būtų įgalintos efektyviau vykdyti reguliavimo reikalavimus, palaikyti aukštą paslaugų kokybę ir stiprinti klientų pasitikėjimą jų veikla.

Be to, mašininio mokymosi modelis turėtų būti kuriamas taip, kad gebėtų mokytis iš naujų duomenų bei nuolat tobulinti savo prognozavimo tikslumą. Tai reiškia, kad jis būtų pritaikytas dinamiškoms aviacinės industrijos sąlygoms ir galėtų greitai prisitaikyti prie kintančių rizikos veiksnių bei naujų aplinkybių. Tokiu būdu avialinijos galėtų realiuoju laiku užtikrinti savo veiklos sklandumą ir aukštą paslaugų kokybę, stiprindamos klientų pasitikėjimą ir reputaciją rinkoje.

Pagrindiniai uždaviniai:

1. Išsamiai išnagrinėti KYC (angl. *Know Your Customer*) procesą ir identifikuoti mašininio mokymosi potencialą atliekant rizikos vertinimą bei reputacijos analizę.
2. Atrinkti tinkamiausius metodus tekstinių duomenų rinkimui ir analizavimui, užtikrinant duomenų kokybę ir reprezentatyvumą.
3. Pritaikyti pažangius „tinklo gramdymo“ (angl. *web scraping*) metodus norint surinkti tekstinius duomenis apie klientus iš viešai prieinamų šaltinių ir tinkamai apdoroti gautus duomenis.
4. Sudaryti aktualiausių aviacijos sektoriui duomenų rinkinį, kuris būtų tinkamas apmokyti mašininio mokymosi modelius, orientuotus į reputacijos rizikos vertinimą.
5. Pritaikyti iš anksto ištreniruotus natūralios kalbos apdorojimo (NLP) modelius surinktiems klientų duomenims ir įvertinti jų efektyvumą.
6. Ištreniruoti surinktą aviacinės pramonės duomenų rinkinį naudojant jau pritaikytus NLP metodus ir optimizuoti šiuos modelius konkrečiai aviacijos sektoriaus kontekstui.
7. Atlikti išsamią metodų palyginamąją analizę, siekiant nustatyti tinkamiausią modelį ar modelių derinį, kuris užtikrintų tikslų ir patikimą reputacijos rizikos vertinimą.

Laukiamas rezultatas

Pasiūlyti modelį, kuris būtų ne tik tikslesnis ir veiksmingesnis nustatant rizikos lygį klientų atpažinimo (KYC) ir tęstinio patikrinimo (angl. *ongoing due diligence*) procesuose, bet ir pakankamai universalus, kad galėtų prisitaikyti prie sparčiai kintančios informacinės ir technologinės aplinkos. Šiuolaikinėje aviacijos pramonėje, kur didelės apimties tarptautiniai sandoriai ir nuolat atsinaujinantys reguliavimo reikalavimai kelia naujų iššūkių, itin svarbu, kad rizikos vertinimo modelis būtų lankstus ir gebėtų greitai prisitaikyti prie naujų situacijų bei netikėtumų. Laukiamas modelis turi būti sukurtas taip, kad galėtų apdoroti didžiulius duomenų srautus iš įvairių šaltinių – tokių kaip tarptautinės sankcijų ir stebimųjų sąrašai, viešai prieinama informacija, socialinių tinklų duomenys – ir atlikti analizę realiuoju laiku.

Toks modelis taip pat turi užtikrinti aukštą prognozavimo tikslumą, kuo mažinant klaidingų teigiamų ir klaidingų neigiamų atvejų skaičių. Tai reiškia, kad modelis efektyviai identifikuos potencialiai rizikingus klientus ar partnerius, tačiau kartu sumažins perteklinius įspėjimus, kurie gali sukelti nereikalingas sąnaudas ir neigiamai paveikti klientų patirtį. Šiuo požiūriu tikslas yra užtikrinti pusiausvyrą tarp griežtų rizikos valdymo priemonių ir sklandaus bei patikimo klientų aptarnavimo, leidžiančio aviacijos sektoriui veikti efektyviai ir užtikrinti vartotojų pasitikėjimą.

Be to, laukiamas modelis turėtų pasižymėti gebėjimu prisitaikyti prie globalių rinkos pokyčių ir naujų taisyklių, kad galėtų būti pritaikomas skirtingose šalyse, atitinkant įvairius reguliavimo reikalavimus. Tokia sistema turėtų būti lengvai integruojama su esamomis aviacijos sektoriaus IT sistemomis, kad modelio diegimas ir naudojimas būtų paprastas ir greitas. Tikimasi, kad siūlomas modelis taps strateginiu sprendimu, padedančiu avialinijoms geriau suprasti savo klientus, gauti vertingų įžvalgų apie jų reputaciją bei elgsenos tendencijas ir iš anksto įvertinti potencialias rizikas.

Apibendrinant, laukiama, kad šis modelis veiks kaip dinamiškas, adaptyvus įrankis, kuris nuolat mokysis iš naujų duomenų, tobulės ir ilgainiui taps patikima priemone, užtikrinančia aukštą saugumo ir reputacijos valdymo lygį aviacijos pramonėje.

1. LITERATŪROS APŽVALGA

1.1. Esami sprendimai rinkoje internetinei reputacijai

1.1.1. Tradiciniai internetinės reputacijos valdymo sprendimai

Dabartiniame internetinės reputacijos valdymo (angl. *Online reputation management - ORM*) kraštovaizdyje vis dažniau dominuoja sprendimai, kuriuose naudojamas dirbtinis intelektas, mašininis mokymasis ir automatizavimas, siekiant supaprastinti sąveikos internete stebėjimo, valdymo ir reagavimo į ją procesą. Įmonės labiau nei bet kada anksčiau suvokia, kokį poveikį jų buvimas skaitmeninėje erdvėje daro visuomenės nuomonei ir klientų pasitikėjimui. Todėl ORM sprendimai išsivystė nuo paprasčiausio atsiliepimų stebėjimo iki išsamių platformų, galinčių atlikti sentimentų analizę, socialinį klausymąsi ir aktyvų įsitraukimą.

Pavyzdžiui, tokiose priemonėse kaip „*Birdeye*“, „ *Podium*“ ir „ *Yext*“ daugiausia dėmesio skiriama sąveikos su klientais supaprastinimui, vietos SEO matomumo didinimui ir daugelio platformų atsiliepimų valdymui. *„Brand24“* ir „*Mention“* siūlo realiuoju laiku veikiančias stebėsenos ir įspėjimo sistemas, kuriose stebimi prekių ženklų paminėjimai ir analizuojamos nuotaikos įvairiuose kanaluose, pavyzdžiui, socialinėje žiniasklaidoje, naujienų portaluose ir forumuose. Šis realaus laiko elementas yra labai svarbus prekių ženklams, kurie nori greitai reaguoti tiek į teigiamas, tiek į neigiamas sąveikas internete, padėdamas jiems formuoti pasakojimą apie savo prekių ženklą. Tokie sprendimai kaip „*Meltwater*“ ir „*Reputation.com*“ papildo šią funkciją konkurencinės lyginamosios analizės funkcijomis, leidžiančiomis įmonėms matyti, kaip jos atrodo lyginant su konkurentais šioje srityje. Šios priemonės skirtos išsamiau apžvelgti prekės ženklo reputaciją internete, o tai labai svarbu didesnėms organizacijoms, kurios valdo kelis kanalus.

Be to, tokios platformos kaip „*RepBot*“ ir „*Grade.us*“ suteikia ORM aukštesnį automatizavimo lygį, nes ne tik stebi ir analizuoja nuotaikas, bet ir valdo sąveiką, automatiškai reaguodamos į atsiliepimus arba prašydamos naujų atsiliepimų. Toks automatizavimo lygis taupo įmonių laiką ir užtikrina pastovų įsitraukimo lygį, ypač vertingą įmonėms, kurių sąveikų internete yra daug. Tuo tarpu „*Reputation.com*“ ir „*BrandYourself*“ atstovauja labiau holistiniam požiūriui, nes siūlo sprendimus, kurie leidžia įmonėms padidinti teigiamą turinį, kad jis užgožtų bet kokius neigiamus atsiliepimus ir taip skatintų palankesnį įvaizdį visuomenėje.

Šių įrankių platų naudojimą patvirtina ir tai, kad, pavyzdžiui, apie „*BrandYourself*„ rašė žinomos žiniasklaidos priemonės, įskaitant „*Shark Tank*“, „*Wall Street Journal“*, „*Forbes*“,

„*NPR*“, „*FOX*“, „*CNBC*“ ir „*ABC*“. Šis žinomumas pabrėžia patikimumą ir platų pripažinimą šioje srityje.

Kadangi įmonės vis labiau orientuojasi į skaitmenines technologijas, išsamių ir automatizuotų ORM sprendimų paklausa ir toliau auga. Šios priemonės atspindi šiuolaikinio reputacijos valdymo judrumo ir skaidrumo svarbą, todėl įmonės gali geriau kontroliuoti ir stebėti savo skaitmeninį pėdsaką ir užbėgti už akių galimoms reputacinėms problemoms. ORM sprendimai vis dažniau laikomi esminiais skaitmeninės rinkodaros strategijų elementais, kurie suteikia prekių ženklams realiuoju laiku įžvalgų ir pro aktyvių strategijų, padedančių didinti klientų lojalumą ir apsaugoti reputaciją itin konkurencingoje skaitmeninėje aplinkoje.

1.1.2. Mašininiu mokymųsi patobulintos reputacijos sistemos

Naujausi mašininio mokymosi pasiekimai gerokai patobulino internetinės reputacijos sistemas, todėl jos gali tiksliau kaupti vertinimus ir būti atsparesnės šališkiems ar nesąžiningiems atsiliepimams. Tradicinės reputacijos sistemos, plačiai naudojamos e. prekyboje ir socialinėse platformose, dažnai remiasi naiviais apibendrinimo metodais, tokiais kaip reitingų vidurkiai ar medianos. Nors šie metodai yra paprasti, jie netinka sudėtingose reitingų aplinkose, kur vartotojų šališkumas ir manipuliavimas reitingais gali iškreipti rezultatus. Kaip teigiama dokumente *Application of Machine Learning for Online Reputation Systems* (Alqwadri, Azzeh, & Almasalha, 2021) „Dažnai naudojami naivūs metodai, kuriuos apskaičiuojant neatsižvelgiama į vartotojų profilius, todėl jie negali aptikti nesąžiningų reitingų ir tendencijų, atsirandančių naujuose reitinguose“.

Reaguojant į šiuos trūkumus, atsirado ML grindžiami reputacijos modeliai, kuriuose naudojami vartotojų profilių duomenys, kad būtų galima užfiksuoti daugiau vartotojų elgsenos niuansų ir numatyti kiekvieno reitingo patikimumą. Žymus ML grindžiamas metodas konstruoja „vartotojų profilio duomenų rinkinį, išskirdamas veiksnių, turinčių didelę įtaką vartotojų patikimumui, rinkinį, kuris yra mašininio mokymosi algoritmų įvestis“. Šį duomenų rinkinį sudaro tokie pagrindiniai rodikliai kaip vartotojų polinkis (atlaidumas arba griežtumas vertinimuose), svyravimas (nuokrypis nuo bendruomenės vertinimų), patirtis (vertinimų dažnumas) ir patikimumas (nuokrypis nuo vidutinio produkto vertinimo). Įtraukus šiuos veiksnius, ML modeliai gali numatyti vartotojų patikimumo balus, kurie atspindi ir patikimumą, ir galimą atskirų vertinimų šališkumą.

Keturi pagrindiniai mašininio mokymosi algoritmai - linijinė regresija (LR), atraminių vektorių regresija (SVR), K- artimiausių kaimynų (KNN) ir regresijos medis (RT)- naudojami

vartotojų svoriams, atspindintiems kiekvieno vartotojo pasitikėjimo lygį, prognozuoti. Pavyzdžiui, regresijos medžio modelis, kuriame taikomas rekursyvaus skaidymo metodas, yra ypač veiksmingas „klasifikuojant duomenis į nuoseklesnes grupes, o vartotojų svoriai prognozuojami pagal artimiausius vartotojus“. Tada šie prognozuojami svoriai integruojami į svertinį vidurkį, kad būtų gautas galutinis produktų ar paslaugų reputacijos balas, todėl, palyginti su tradiciniais metodais, gerokai padidėja tikslumas.

Siekdami įvertinti šiuos ML modelius, tyrėjai juos pritaikė įvairiems duomenų rinkiniams iš „*MovieLens*“ platformos, kuri yra dažnai naudojamas lyginamasis šaltinis rekomenduojančiųjų sistemų tyrimams. Duomenų rinkinių dydis svyravo nuo 100 000 iki 10 milijonų vertinimų, todėl buvo galima nuodugniai išbandyti modelių mastelio keitimo galimybes ir našumą. Straipsnyje pabrėžiama, kad „gauti rezultatai buvo daug žadantys, o tai leidžia manyti, kad siūlomas metodas galėtų būti potencialus reputacijos sistemų sprendimas“.

Našumas buvo matuojamas naudojant vidutinę absoliučiąją klaidą (MAE) ir „Kendall Tau“ koreliaciją, o rezultatai parodė, kad ML patobulinti modeliai pranoko tradicines sistemas, įskaitant Bajeso ir neaiškios logikos metodus, pasiekdami „didesnį tikslumą esant retiems duomenų rinkiniams ir tankiems duomenų rinkiniams“. Kaip parodyta 1 lentelėje, regresijos medžio (RT) modelis visuose duomenų rinkiniuose (100 tūkst., 1 mln. ir 10 mln.) nuosekliai pasiekė aukščiausią įvertinimą, po jo sekė tiesinė regresija (LR) ir K artimiausi kaimynai (KNN), o tai patvirtina ML metodų veiksmingumą reputacijai apibendrinti.

1 lentelė

Modelių reitingas remiantis vidutine absoliučiąją klaida (MAE) 3 – iuose duomenų rinkiniuose

Reitingas	100 tūkst. eilučių Duomenų rinkinys	1 milijono eilučių duomenų rinkinys	10 milijonų eilučių duomenų rinkinys
1	RT	RT	RT
2	LR	LR	LR
3	KNN	KNN	KNN
4	SVR	SVR	SVR
5	Median	Median	Median
6	BetaDR	BetaDR	BetaDR
7	Dirichlet	Dirichlet	Average
8	Bayesian	Bayesian	Dirichlet
9	Average	Average	Bayesian
10	IMDb	Fuzzy	Fuzzy
11	Fuzzy	IMDb	IMDb
12	LQ	LQ	LQ

Šaltinis: (Alqwadri, Azzeh, & Almasalha, 2021)

Svarbus mašininiu mokymųsi grindžiamų reputacijos modelių privalumas yra jų lankstumas prisitaikant prie vartotojų elgsenos realiuoju laiku, o to trūksta tradiciniams metodams. Dokumente pažymima, kad „dauguma dabartinių svertinių metodų sutelkia dėmesį į vieną vartotojų vertinimų aspektą..., tačiau jų neprognozuoja“. Kita vertus, ML grindžiamos sistemos gali dinamiškai koreguoti svorius, remdamosi vartotojų patikimumo profiliais, todėl jos geriau reaguoja į atsirandančias tendencijas ir yra mažiau linkusios iškreipti rezultatus dėl nukrypimų ar staigių vartotojų nuotaikų pokyčių.

Be to, šios ML grindžiamos reputacijos sistemos suteikia reikšmingų privalumų rekomendavimo sistemoms, nes didina vartotojų pasitikėjimą ir patirtį. Kaip pabrėžiama dokumente, „siūlomas metodas gali būti integruotas į internetines rekomendacijų sistemas, kad būtų galima teikti geresnes pirkimo rekomendacijas ir palengvinti naudotojų patirtį internetinėse

prekybos rinkose“. Susiejant produktų reitingus su vartotojų, kuriais labai pasitikima, profiliais, sistema pagerina rekomendacijų tinkamumą ir sumažina šlamšto ar suklastotų reitingų poveikį, todėl vartotojai gali pasikliauti rekomendacijomis.

Apibendrinant galima teigti, kad mašininio mokymosi integravimas į reputacijos sistemas suteikia daugiau galimybių ir tikslumo, pranoksta tradicinius metodus nustatant tendencijas, valdant šališkumą ir apskaičiuojant patikimą reputaciją, pagrįstą išsamiais vartotojų profiliais. Šis ML grindžiamas metodas veiksmingai tenkina e. prekybos ir socialinių platformų poreikius, suteikdamas vartotojams galimybę priimti labiau informuotus ir patikimesnius sprendimus, kartu didindamas internetinių reitingų ekosistemų vientisumą. Tačiau, nepaisant šios pažangos, vis dar trūksta konkrečiai aviacijai skirtų reputacijos sprendimų. Dabartiniai reputacijos valdymo modeliai neatsižvelgia į unikalius aviacijos pramonės reikalavimus, pavyzdžiui, atitiktį teisės aktų reikalavimams, saugos problemas ir konkrečiai pramonei būdingus reputacijos veiksnius, todėl šiame sektoriuje atsiranda galimybė taikyti specialiai pritaikytus ML grindžiamus sprendimus.

1.2. Mašininis mokymasis KYC srityje

Mašininis mokymasis (ML) ir natūralios kalbos apdorojimas (NLP) pastaruoju metu įsitvirtino kaip galingos technologijos, galinčios spręsti įvairius iššūkius finansų ir reguliavimo sektoriuose, įskaitant sukčiavimo aptikimą ir klientų pažinimo (KYC) procesus. Didėjant duomenų kiekiui ir sudėtingėjant finansinių operacijų struktūrai, tradiciniai metodai nebegali efektyviai aptikti sukčiavimo modelių ar užtikrinti atitikties reikalavimų. Aviacijos sektoriuje, kuris pasižymi didelės vertės sandoriais ir daugybe tarptautinių ryšių, šios technologijos tampa itin svarbios.

Išmanūs KYC procesai yra būtini norint apsaugoti įmones nuo finansinio sukčiavimo, pinigų plovimo ir kitų neteisėtų veiklų. Tačiau ši veikla yra ne tik techninė būtinybė; tai taip pat procesas, kuris gali daryti reikšmingą įtaką klientų pasitenkinimui ir verslo efektyvumui. ML pagrįsti sprendimai suteikia galimybę automatizuoti ir optimizuoti šiuos procesus, taip sumažinant klaidų tikimybę, padidinant greitį bei tikslumą. Šiame skyriuje nagrinėjama, kaip šiuolaikiniai ML metodai gali būti integruoti į KYC procesus, siekiant užtikrinti efektyvų sukčiavimo aptikimą ir rizikos valdymą.

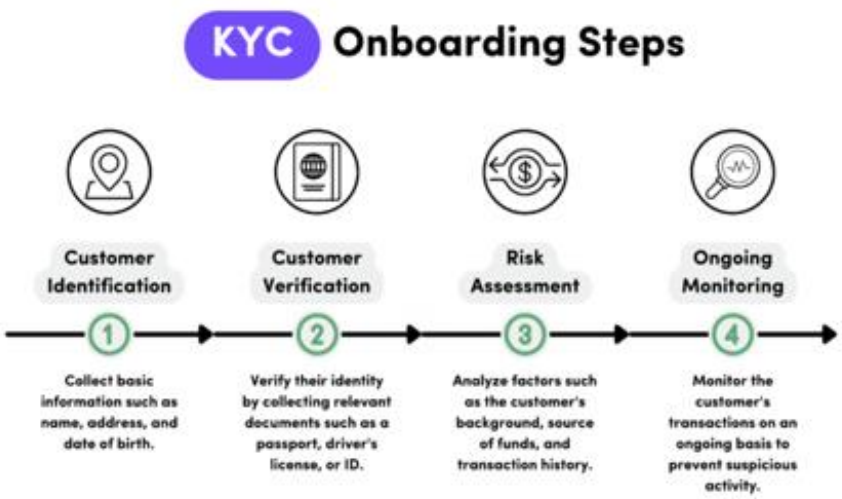
1.2.1. Atpažink savo klientą (KYC)

KYC (angl. *Know Your Customer*) reguliavimo pagrindai kilo iš būtinybės užkirsti kelią ilgą laiką nekontroliuojamiems finansiniams nusikaltimams. Pirmieji KYC gairių pagrindai buvo sukurti 1970 m., kai Jungtinės Valstijos priėmė Bank Secrecy Act (BSA), siekiant kovoti su pinigų plovimu. Vėliau gairės buvo papildytos po 2001 m. rugsėjo 11 d. teroristinių atakų ir 2008 m. pasaulinės finansų krizės, pabrėžiant jų svarbą pasaulinio saugumo ir finansinio stabilumo kontekste (Understanding the Steps of a "Know Your Customer" Process, 2024)

Pasak Chen (2020), „KYC duomenys bankams gali būti vertinga rizikos vertinimo priemonė, nes jie suteikia informacijos, pagal kurią galima nustatyti klientus, kurie yra labiau linkę negrąžinti paskolos". Ši įžvalga pabrėžia, kad tinkamai vykdomi KYC procesai ne tik užtikrina atitiktį teisiniams reikalavimams, bet ir padeda finansų įstaigoms veiksmingai valdyti riziką ir nustatyti klientus, kurių veikla gali kelti didesnę finansinę riziką.

KYC taip pat yra reikšminga priemonė, padedanti stiprinti valstybių atsparumą kovai su pinigų plovimu ir sankcijų įgyvendinimu. Lietuvos bankas (2023) teigia, kad „tiksliai ir laiku pateikta informacija mokėjimo paslaugų teikėjams leidžia didinti šalies atsparumą grėsmėms, susijusioms su pinigų plovimu ir sankcijų vengimu." Be to, KYC informacijos rinkimas iš klientų padeda užtikrinti reguliacinių reikalavimų laikymąsi ir prisideda prie šalies finansų sektoriaus skaidrumo

Kaip pažymi „*iDenfy*" (Stankevičiūtė, 2023), teikianti klientų identifikavimo paslaugas, KYC onboarding procesas – tai deramo klientų tikrinimo (angl. *Customer Due Diligence - CDD*) procedūra, kurią finansų įstaigos ir kiti reguliuojami subjektai privalo atlikti prieš užmegzdami santykius su nauju klientu. Šis procesas apima kliento tapatybės tikrinimą ir autentiškumo užtikrinimą, leidžiant įmonėms užkirsti kelią nesąžiningiems sandoriams ir apsisaugoti nuo galimų teisinių ir finansinių pasekmių. Standartiškai, KYC procesas apima keturis pagrindinius etapus: kliento identifikavimą, jo tapatybės tikrinimą, rizikos įvertinimą ir nuolatinę stebėseną (1 pav.).



Šaltinis: *(Stankevičiūtė, 2023)*

1 pav. iDenfy standartinis KYC modelis

Taip pat svarbu pažymėti, kad sudėtingas ar daug laiko užimantis KYC įtraukimo procesas gali nuvilti potencialius klientus ir atbaidyti juos nuo verslo su įmone. Įvedimas į sistemą yra ne tik pirmas įspūdis, kurį klientai susidaro, bet ir esminis procesas, apsaugantis įmonę ir jos klientus nuo sukčiavimo ir kitos neteisėtos veiklos. Kaip pabrėžia „*iDenfy*“ (Stankevičiūtė, 2023), įmonės turi išlaikyti pusiausvyrą tarp AML (angl. *Anti-Money Laundering*) reikalavimų įgyvendinimo ir teigiamos klientų patirties. Tai tampa itin svarbu didelės rizikos sektoriuose, kur KYC veiksmai yra neatsiejami nuo efektyvaus rizikos valdymo.

Kaip pažymi prof. Venkatesh U. Rajput (2013) savo leidinyje, KYC procedūros yra ne tik teisinė būtinybė, bet ir svarbus įrankis kuriant patikimą verslo aplinką. Taip pat pabrėžiama, kad KYC procesai neatsiejami nuo rizikos valdymo, skaidrumo užtikrinimo ir prevencinių veiksmų prieš neteisėtą veiklą, tokią kaip tapatybės vagystės ar pinigų plovimas. Be to, Pasaulinio reguliavimo institucijų, tokių kaip „*Basel Committee*“, gairės aiškiai nurodo, kad efektyvios KYC sistemos yra būtinos siekiant išvengti finansinių nuostolių ir apsaugoti rinkos stabilumą.

Aviacijos sektoriuje, kuris susiduria su didelės vertės sandoriais ir sudėtinga reguliacine aplinka, poreikis efektyviai vykdyti KYC procesus tampa dar aktualesnis. Šis sektorius yra išskirtinis dėl savo aukštų reguliavimo standartų, kurie dažnai skiriasi priklausomai nuo šalies ar regiono. Be to, rizikos veiksniai, tokie kaip potencialios sąsajos su didelės rizikos jurisdikcijomis ar sudėtingos nuosavybės struktūros, reikalauja griežtesnės kontrolės. Todėl automatizuoti ir mašininio mokymosi metodais paremti sprendimai tampa itin aktualūs, siekiant optimizuoti KYC procesus.

Efektyviai įgyvendinti KYC procesai aviacijos sektoriuje yra būtini ne tik tam, kad būtų užtikrinta atitiktis reguliavimo reikalavimams, bet ir tam, kad būtų kuriamas patikimas ir skaidrus ryšys su klientais bei partneriais. Šie procesai padeda užkirsti kelią nesąžiningiems sandoriams, gerina rizikos valdymą ir stiprina įmonės konkurencingumą globalioje rinkoje.

1.2.2. Esami KYC proceso pavyzdžiai aviacijoje

Aviacijos pramonėje taikomas išsamus KYC įdarbinimo (angl. *onboarding*) procesas, kuriuo siekiama patikrinti sandorio šalių patikimumą ir rizikos lygį, užtikrinti atitiktį reikalavimams ir sumažinti galimą verslo santykių riziką. Šis procesas apima kelis etapus, o konkrečios funkcijos ir atsakomybė priskirta pardavimų specialistui, atitikties specialistui ir vyriausiajam atitikties užtikrinimo pareigūnui.

Atsižvelgiant į sandorio šalies statusą, taikomi skirtingi įtraukimo į apskaitą būdai:

- Naujos sandorio šalies įtraukimas į rinką: Naudojamas subjektams, kurių bendrovė anksčiau nepatvirtino.
- Viešųjų subjektų įtraukimas į apskaitą: Taikomas mažos rizikos jurisdikcijose veikiančioms bendrovėms ir viešai listinguojamiems subjektams.
- Išskirtinės sandorio šalies įtraukimas į apskaitą: Taikoma tais atvejais, kai kita sandorio šalis atsisako pateikti KYC dokumentus arba jų nepateikia.

Įtraukimo į sistemą procesą sudaro keli etapai, kiekviename iš jų palaipsniui įvertinama sandorio šalies rizika:

1. Pradinis KYC dokumentų patikrinimas ir pateikimas, kurį atlieka pardavimų specialistas.
2. Dokumentų peržiūra, kurią atlieka atitikties specialistas, tikrinantis informaciją vidaus ir viešose duomenų bazėse.
3. Rizikos vertinimas - šiame etape naudojamas rizikos indikatoriaus įrankis, kuriuo nustatoma sandorio šalies rizikos lygio klasifikacija (pvz., didelė, vidutinė, maža). Remdamasis įvairiais kriterijais, pavyzdžiui, nuosavybės struktūra, geografiniais ryšiais ir galimomis sąsajomis su didelės rizikos jurisdikcijomis, atitikties užtikrinimo specialistas įvertina sandorio šalies rizikos statusą.
4. Vyriausiasis atitikties užtikrinimo pareigūnas peržiūri visas išvadas, nustato galutinį rizikos lygį ir priima sprendimą dėl sandorio šalies patvirtinimo arba atmetimo.
5. Priimamas galutinis sprendimas, įskaitant galimą įtraukimą į juodąjį sąrašą, jei sandorio šalis laikoma per daug rizikinga.

Būtent trečiasis KYC proceso etapas - rizikos vertinimo etapas - yra palankiausias automatizavimui naudojant mašininį mokymąsi (ML). Šiame etape vertinami įvairūs rizikos rodikliai siekiant klasifikuoti sandorio šalies rizikos lygį. Automatizavus šį etapą naudojant ML, aviacijos įmonės gali padidinti vertinimo efektyvumą ir tikslumą. ML modeliai gali dinamiškai analizuoti didelius duomenų rinkinius, nustatyti sudėtingus modelius ir greitai prisitaikyti prie naujų sukčiavimo taktikų. Toks automatizavimas ne tik pagreitina rizikos klasifikavimo procesą, bet ir sumažina atitikties užtikrinimo komandų rankinio darbo krūvį, todėl jos gali sutelkti dėmesį į sudėtingesnius atvejus. ML diegimas rizikos vertinime galiausiai sustiprina KYC procesą, todėl jis tampa patikimesnis, operatyvesnis ir efektyvesnis valdant riziką aviacijos pramonėje.

1.2.3. Nuolatinė KYC procesų stebėsena aviacijoje

Svarbu pažymėti, kad KYC sistema apima ne tik pradinį įtraukimo į sistemą procesą, bet ir nuolatinį deramą patikrinimą, kuriuo siekiama užtikrinti atitiktį (angl. *compliance*) ir ilgainiui sumažinti riziką. Kaip aprašyta aviacinės įmonės „*x*“ įdarbinimo ir deramo patikrinimo procedūroje, nuolatinė stebėsena atlieka pagrindinį vaidmenį siekiant išlaikyti atnaujintą ir tikslų kiekvienos sandorio šalies rizikos profilį. Nuolatinis deramas patikrinimas (angl. *Continuous Due Diligence, CDD*) - tai klientų ir jų veiklos stebėjimas, siekiant užtikrinti, kad kliento pateikta informacija išliktų aktuali ir patikima. Pasak Rajput (2013), „*CDD* reiškia klientų ir jų veiklos stebėseną, siekiant įsitikinti, ar klientas laikui bėgant reikšmingai nesikeičia. Iš esmės taip kovojama su galimybe, kad asmuo (ar dažniau organizacija), kuris praėjo KYC patikrinimą, vis dar yra tas, kuo sakosi esąs, ir daro tai, ką sakė darysiąs, kai buvo tikrinamas KYC.“

Pagal aviacinės įmonės „*x*” procedūra, pagrindiniai nuolatinės stebėsenos aspektai yra šie:

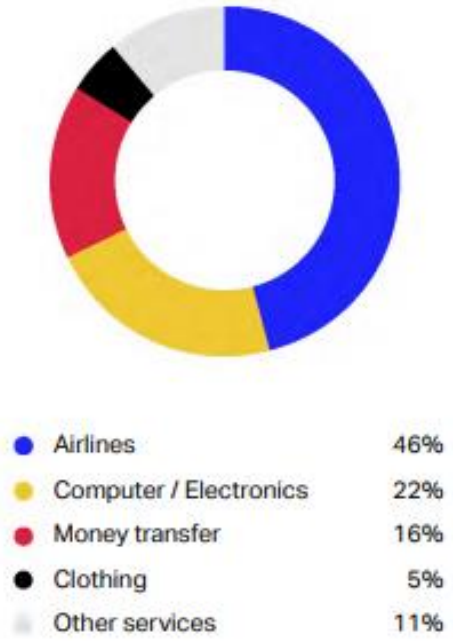
- KYC dokumentų patikrinimai: Atitikties užtikrinimo pareigūnai reguliariai tikrina sandorio šalies informaciją, pavyzdžiui, registruotus adresus, nuosavybės struktūras ir juridinio asmens duomenis, siekdami įsitikinti, kad nuo patekimo į rinką etapo neįvyko jokių pokyčių. Jei tarp pradinių ir atnaujintų duomenų atsiranda neatitikimų, pradedamas procesas, kurio metu prašoma pateikti naujus dokumentus arba inicijuojamas pakartotinis vertinimas.
- Nepalankios žiniasklaidos patikrinimas: Žiniasklaida ir viešos duomenų bazės reguliariai tikrinamos, ar nėra galimos reputacijos rizikos arba neigiamų išvadų, susijusių su kita sandorio šalimi, jos akcininkais ar direktoriais.
- Sandorių peržiūra: Sandorio šalies sandoriai peržiūrimi pagal jos verslo profilį, siekiant nustatyti neįprastus modelius ar neatitikimus. Šis procesas padeda užtikrinti, kad sandoriai atitiktų numatytą veiklą ir atitiktų reguliavimo standartus.

Remiantis šiomis išvadomis gali būti priimami tokie sprendimai: prašyti papildomos informacijos, koreguoti sandorio šalies rizikos lygį, išlaikyti esamą rizikos statusą arba nutraukti santykius dėl neatitikties ar didelės rizikos asociacijų.

Nuolatinė stebėsena yra labai svarbi siekiant užkirsti kelią nesąžiningai veiklai ir užtikrinti, kad KYC procesai išliktų veiksmingi ir pritaikyti prie kintančių aplinkybių. Šis požiūris ypač svarbus aviacijos sektoriuje, kur didelės vertės sandoriai ir tarptautiniai ryšiai kelia unikalių iššūkių.

1.3. Atitiktis ir reguliacinė aplinka aviacijoje

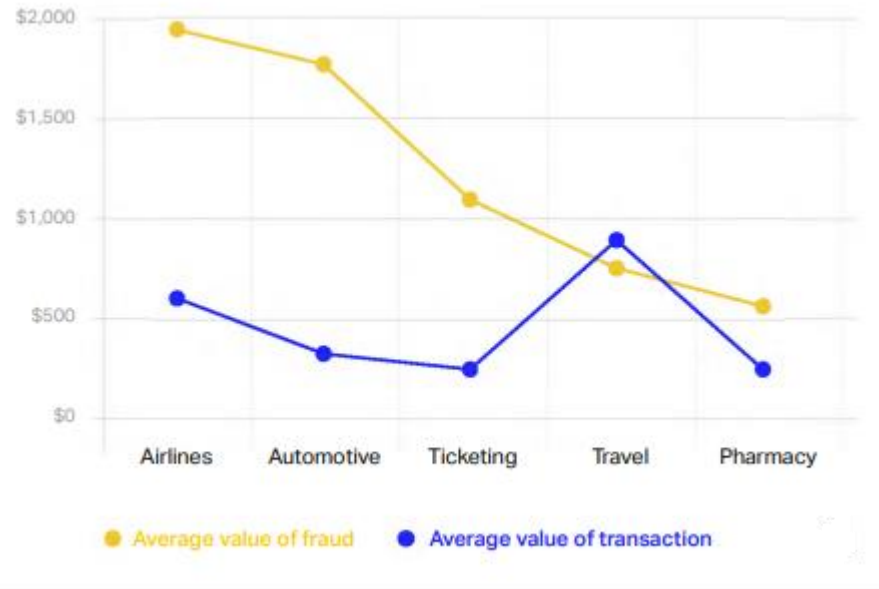
Spartus skaitmeninis oro linijų pramonės virsmas, kurį paspartino perėjimas prie internetinių užsakymų ir mobiliųjų bilietų pardavimo, sukūrė precedento neturinčias sukčiavimo galimybes. Remiantis IATA 2020 m. baltosios knygos duomenimis (IATA, 2020), oro transporto bendrovės dėl sukčiavimo kasmet patiria apie 1 mlrd. dolerių nuostolių, iš kurių „dažniausiai pasitaikantis sukčiavimas yra kredito kortelių sukčiavimas, ypač atliekant operacijas internetu“. Ši problema ne tik lemia finansinius nuostolius, bet ir mažina klientų pasitikėjimą bei didina veiklos sąnaudas, nes oro transporto bendrovės nuolat bando valdyti ir mažinti šią riziką.



Šaltinis: (IATA, 2020)

2 pav. Didžiausią sukčiavimo sandorių poveikį patyrę prekybininkai

Pagal pateiktą diagramą (2 pav.), aviacijos sektorius 2020 metais sudarė net 46% visų praneštų sukčiavimo atvejų, aiškiai parodydamas, kad šis sektorius yra vienas iš labiausiai pažeidžiamų. Tai galima paaiškinti dideliu internetinių operacijų mastu, kai užsakomi bilietai, valdomi lojalumo programų taškai ar atliekami kiti su oro transportu susiję sandoriai. Aukštos bilietų kainos ir didelės vertės sandoriai (3 pav.) sukuria palankią terpę sukčiavimui, ypač kai pasikliaujama skaitmeniniais mokėjimo kanalais. Aviacijos bendrovėms ši situacija kelia ne tik tiesioginių finansinių nuostolių, bet ir ilgalaikę grėsmę klientų pasitikėjimui. Todėl aviakompanijoms būtina diegti pažangius sukčiavimo prevencijos sprendimus, tokius kaip dirbtinio intelekto ir mašininio mokymosi pagrindu veikiančios sistemos, kad būtų galima efektyviai identifikuoti neteisėtas veiklas ir užkirsti joms kelią realiuoju laiku.



Šaltinis: (IATA, 2020)

3 pav. Vidutinė sukčiavimo sandorių vertė

Sukčiautojai tampa vis išradingesni, pasinaudodami mokėjimo procesų spragomis ir išnaudodami lojalumo programų pažeidžiamumą. „Kibernetiniai nusikaltėliai naudoja pažangius metodus, kad įvykdytų sukčiavimą mokėjimais, sąskaitų perėmimą ir lojalumo taškų vagystes“, - pažymi Feldman (2024) ‚*BlueVoyant*‘ straipsnyje. Perėjimas prie bilietų pardavimo internetu dar labiau padidino šiuos pažeidžiamumus, nes „nesant tradicinio tiesioginio bendravimo, padidėja sukčiavimo tikimybė“ (Amadeus, 2014). Ši besikeičianti aplinka reikalauja patikimų skaitmeninių apsaugos priemonių, kurių standartinė patikros praktika ir sandorių stebėsenos metodai nebegali visiškai užtikrinti.

1.3.1. Reguliacinės įstaigos aviacijoje

Tarp pagrindinių reguliacinių institucijų, veikiančių aviacijos srityje, yra Tarptautinė civilinės aviacijos organizacija (ICAO), JAV Federalinė aviacijos administracija (FAA) ir Europos Sąjungos aviacijos saugos agentūra (EASA). Kiekviena jų atlieka svarbų vaidmenį užtikrinant tarptautinės aviacijos saugą, skaidrumą ir atitiktį.

Tarptautinė civilinės aviacijos organizacija (ICAO) yra Jungtinių Tautų specializuota agentūra, kuri nustato tarptautinius standartus ir rekomendacijas, taikomas civilinei aviacijai. ICAO siekia užtikrinti saugų ir efektyvų tarptautinių skrydžių tinklą, ypač pabrėždama saugos ir saugumo taisyklių laikymąsi. Ji taip pat skatina pasaulinių aviacijos procedūrų harmonizavimą, kad būtų išvengta reguliacinių neatitikimų tarp šalių.

JAV Federalinė aviacijos administracija (FAA) yra pagrindinė institucija Jungtinėse Valstijose, atsakinga už civilinės aviacijos reguliavimą. FAA funkcijos apima oro transporto bendrovių ir jų veiklos licencijavimą, orlaivių sertifikavimą, taip pat aviacijos infrastruktūros priežiūrą. Be to, FAA turi didelę įtaką pasauliniu mastu, nes jos nustatyti standartai dažnai tampa pavyzdžiu kitoms šalims.

Europos Sąjungos aviacijos saugos agentūra (EASA) užtikrina, kad visose Europos Sąjungos valstybėse narėse būtų taikomos vienodos saugos ir atitikties taisyklės. EASA ne tik sertifikuoja orlaivius ir jų komponentus, bet ir koordinuoja aviacijos incidentų tyrimus bei stiprina reguliavimo nuoseklumą tarp valstybių. Ši agentūra taip pat aktyviai prisideda prie klimato kaitos mažinimo iniciatyvų, skatindama tvaresnį aviacijos sektoriaus vystymąsi.

Šios reguliacinės institucijos yra kertinės ne tik užtikrinant saugumą, bet ir sprendžiant finansinių nusikaltimų riziką aviacijos sektoriuje. Jų įsteigtos gairės ir standartai reikalauja nuoseklaus KYC procesų vykdymo, taip pat prisideda prie kovos su pinigų plovimu ir terorizmo finansavimu.

1.3.2. Sankcijų svarba

Sankcijos yra labai svarbi geopolitinių veiksmų reguliavimo ir tarptautinių įstatymų laikymosi užtikrinimo priemonė. Tačiau jų poveikis toli peržengia šalių, kurioms taikomos sankcijos, ribas, ypač pasauliniu mastu susietose pramonės šakose, pavyzdžiui, aviacijos. Aviacijos pramonė patyrė didelių sutrikimų dėl įvairių sankcijų, ypač tų, kurios buvo taikomos sudėtingos geopolitinės padėties šalims, pavyzdžiui, Rusijai, Iranui ir Šiaurės Korėjai. Pasak žurnalo „*Forbes*“ (2024), Rusija yra daugiausiai sankcijų pasaulyje taikanti šalis - jai nustatyta

daugiau kaip 20 000 apribojimų, o tai rodo, kad sankcijos turi didelę geopolitinę reikšmę tokioms pramonės šakoms kaip aviacija.

Tyrimo „Sankcijų išplitimo poveikis oro transporto bendrovėms“ autoriai Shenas Corbet, Juanas L. Nicolau ir Lesas Oxley savo darbe (2023) išskiria keletą pagrindinių poveikio sričių, kurias sankcijos daro aviacijos pramonei:

1. Oro erdvės apribojimai ir veiklos suvaržymai: Sankcijos dažnai apima oro erdvės apribojimus, kurie riboja sankcijų šalių oro transporto bendrovių galimybes laisvai vykdyti skrydžius. Pavyzdžiui, po 2022 m. Rusijos ir Ukrainos karo tokios šalys kaip JAV, ES narės ir Kanada uždraudė Rusijos orlaiviams skristi jų oro erdvėse. Tai ne tik sutrikdė skrydžių veiklą, bet ir privertė oro transporto bendroves rinktis ilgesnius ir mažiau efektyvius maršrutus, dėl to gerokai padidėjo veiklos sąnaudos.
2. Orlaivių nuomos ir techninės priežiūros iššūkiai: Sankcijos taip pat nukreiptos prieš orlaivių techninei priežiūrai ir eksploatavimui būtinas tiekimo grandines. Pavyzdžiui, pagrindiniai orlaivių gamintojai, tokie kaip „*Boeing*“ ir „*Airbus*“, laikydamiesi sankcijų sustabdė dalių tiekimą, techninę priežiūrą ir nuomos veiklą Rusijos oro transporto bendrovėms. Tai sukėlė logistinę ir finansinę krizę Rusijos oro linijų bendrovėms, kurios labai priklauso nuo vakarietiškų orlaivių.
3. Rinkos nepastovumas ir investuotojų pasitikėjimas: Aviacijos pramonė patyrė didelį rinkos svyravimą, nes sankcijos turėjo įtakos pagrindinių oro transporto bendrovių akcijų vertinimui. Remiantis Corbet et al. tyrimu (2023), Rusijos aviacijai įvestos sankcijos sukėlė didelį „šalutinį poveikį“, neigiamai paveikusį investuotojų pasitikėjimą oro transporto bendrovėmis visame pasaulyje. Didžiausią poveikį patyrė didžiosios oro transporto bendrovės, nes jos dirba tarptautiniais maršrutais ir patiria didesnes veiklos sąnaudas.
4. Atitikties ir reguliavimo iššūkiai: Pasaulyje veikiančios oro transporto bendrovės turi užtikrinti, kad būtų laikomasi tarptautinių sankcijų, įskaitant JAV Užsienio turto kontrolės biuro (OFAC), ES konsoliduotojo sankcijų sąrašo ir Jungtinių Tautų Saugumo Tarybos sankcijų sąrašo, reikalavimų. Už šių direktyvų nesilaikymą gali būti taikomos griežtos nuobaudos, įskaitant baudas ir veiklos draudimus.

Sankcijos nėra tik baudžiamosios priemonės; jos taip pat veikia kaip reguliavimo priemonės, užtikrinančios, kad pasaulinė aviacijos pramonė atitiktų tarptautines normas. Tokios organizacijos kaip OFAC taiko finansinius apribojimus, kad oro transporto bendrovės negalėtų bendradarbiauti su subjektais, kuriems taikomos sankcijos, o ES ir JT sudaro išsamius asmenų, grupių ir subjektų, kuriems taikomos sankcijos, sąrašus. Šiomis priemonėmis siekiama skatinti

atskaitomybę ir užkirsti kelią piktnaudžiavimui aviacija neteisėtai veiklai, pavyzdžiui, terorizmui ar pinigų plovimui. Kaip aiškina „*Forbes*“ (2024), sankcijos dažnai taikomos „siekiant daryti politinį ar ekonominį spaudimą, tačiau jos dažnai turi nenumatytų pasekmių ir daro poveikį pasauliniu mastu veikiančioms pramonės šakoms, pavyzdžiui, aviacijai“.

Nors sankcijos padarė didelį poveikį Rusijos oro transporto bendrovėms ir jų artimiausioms rinkoms, jų poveikis išplito ir į kitus regionus, todėl atsirado asimetrija. Europos, Šiaurės Amerikos ir Azijos oro transporto bendrovės patyrė įvairaus laipsnio veiklos ir finansinę įtampą dėl savo ryšių su Rusijos rinka ir platesnėmis pasaulinėmis tiekimo grandinėmis. Ateityje ši pramonės šaka turi subalansuoti būtinybę laikytis sankcijų ir išlaikyti veiklos efektyvumą, galbūt pasitelkdama naujoviškas technologijas, pavyzdžiui, blokų grandinę, kad užtikrintų skaidrumą ir automatizuotą atitikties stebėjimą.

Suprasdamos ir spręsdamos šiuos iššūkius, oro transporto bendrovės gali geriau orientuotis sudėtingame sankcijų formuojamame reguliavimo kraštovaizdyje ir užtikrinti atsparią veiklą geopolitinių trikdžių akivaizdoje.

1.3.3. Netinkamų KYC/AML procedūrų pasėkmės

„Pažink savo klientą“ (KYC) protokolų nesilaikymas istoriškai palengvino didelio masto sukčiavimą, pinigų plovimą ir kitą neteisėtą veiklą, dėl ko finansų įstaigos patyrė didelę reputacinę ir finansinę žalą. Šios klaidos rodo, kad patikimos KYC priemonės atlieka itin svarbų vaidmenį užtikrinant atitiktį reguliavimo standartams ir pasaulinės finansų sistemos vientisumą. Toliau pateikiami svarbūs atvejai, rodantys nesėkmingų KYC procesų pasekmes.

Vienas svarbiausių naujausių nesėkmingų KYC procesų pavyzdžių yra „*Westpac Banking Corporation*“ atvejis. 2020 m. rugsėjį „*Westpac*“ sutiko sumokėti 1,3 mlrd. australų dolerių baudų po to, kai buvo pripažinta kalta pažeidusi Australijos pinigų plovimo ir kovos su terorizmo finansavimu įstatymą. Pažeidimai apėmė tai, kad bankas tinkamai nepranešė apie daugiau kaip 19,5 mln. tarptautinių lėšų pervedimo nurodymų (IFTI) ir neužtikrino tinkamo deramo klientų patikrinimo. Dėl šių pažeidimų įtartinos operacijos, įskaitant su vaikų išnaudojimu susijusius mokėjimus, galėjo būti nepastebėtos. Kaip pranešė Finansinių nusikaltimų akademija, šis atvejis atskleidė „katastrofiškas pasekmes“, kai neįgyvendinamos veiksmingos KYC ir klientų stebėsenos sistemos, todėl tai yra didžiausia bauda Australijos įmonių istorijoje. (Endorsed Regulatory Fines And Enforcement Actions Relating To CDD/KYC Failures, 2024)

„*Danske Bank*“ atvejis yra dar vienas pavyzdys, kaip nepakankamos KYC procedūros gali lemti didelį sukčiavimą. 2007-2015 m. „*Danske Bank*“ Estijos filialas buvo didžiulės pinigų

plovimo operacijos centre, kai per jo sąskaitas buvo atlikta daugiau kaip 200 mlrd. eurų įtartinų operacijų. Tyrimai atskleidė, kad netinkama KYC praktika leido fiktyvioms bendrovėms ir didelės rizikos klientams, įskaitant klientus iš šalių, kurioms taikomos sankcijos, naudotis banko paslaugomis neteisėtais tikslais. Dėl to, kad nebuvo sustiprinto deramo patikrinimo ir veiksmingos didelės rizikos klientų stebėsenos, bankas tapo nusikalstamos veiklos kanalu. Kaip pažymima JAV viešųjų reikalų biuro ataskaitoje, šis skandalas paskatino plačią reguliavimo institucijų priežiūrą ir atskleidė prastos KYC sistemos pražūtingas pasekmes. (Danske Bank Pleads Guilty to Fraud on U.S. Banks in Multi-Billion Dollar Scheme to Access the U.S. Financial System, 2022)

2019 m. bankui „*Standard Chartered*“ buvo skirta 1,1 mlrd. dolerių bauda už kovos su pinigų plovimu (AML) ir sankcijų taisyklių pažeidimus. Pažeidimai daugiausia buvo susiję su KYC ir klientų stebėsenos sistemų trūkumais, ypač dirbant su didelės rizikos klientais tokiose šalyse kaip Iranas ir Mianmaras. Pasak „*ACA Global*“, vykdymo užtikrinimo veiksmus lėmė trūkumai tikrinant klientus pagal sankcijų sąrašus ir netinkama sandorių, susijusių su šalimis, kurioms taikomos sankcijos, stebėsena. Šis atvejis tapo įspėjimu finansų įstaigoms visame pasaulyje apie būtinybę užtikrinti atitiktį KYC ir kovos su pinigų plovimu taisyklėms. (Makortoff, 2019)

2012 m. JAV valdžios institucijos „*HSBC“* skyrė 1,9 mlrd. dolerių baudą už nesugebėjimą užkirsti kelią pinigų plovimo veiklai. Silpna KYC kontrolė leido Meksikos ir Kolumbijos narkotikų karteliams per banką išplauti šimtus milijonų dolerių. Pasak pranešimų, dėl prastos „*HSBC*“ deramo patikrinimo praktikos jos paslaugomis galėjo naudotis ir su terorizmu susiję klientai. Šis atvejis pabrėžė patikimų KYC sistemų svarbą nustatant ir mažinant riziką, susijusią su didelės rizikos klientais ir jurisdikcijomis (HSBC Holdings Plc. and HSBC Bank USA N.A. Admit to Anti-Money Laundering and Sanctions Violations, Forfeit $1.256 Billion in Deferred Prosecution Agreement, 2012)

Maltos bankas „*Pilatus Bank*“ buvo uždarytas po to, kai buvo įtrauktas į didelį atgarsį sukėlusį pinigų plovimo skandalą. Reguliavimo institucijų tyrimai atskleidė, kad bankas nevykdė pagrindinių KYC procedūrų, todėl politiškai veiklūs asmenys (PEP) galėjo pervesti neteisėtas lėšas be jokio patikrinimo. Šis atvejis atskleidė, kad labai svarbu taikyti griežtesnes deramo patikrinimo priemones, skirtas PEP ir kitiems didelės rizikos asmenims. (Osborne, 2018)

2023 m. didžiausios baudos už AML ir KYC pažeidimus dar kartą parodė, kokias dideles pasekmes patiria organizacijos dėl reguliavimo neatitikimų. Pavyzdžiui, kriptovaliutų birža „*Binance*“ gavo rekordinius 4,3 mlrd. USD už pinigų plovimą, neregistruotų pinigų pervedimų vykdymą ir sankcijų pažeidimus. Australijos kazino tinklas „*Crown Resorts*“ buvo nubaustas 450

mln. USD už kovos su pinigų plovimu kontrolės trūkumus ir nesugebėjimą tinkamai įvertinti terorizmo finansavimo rizikos Melburno ir Perto kazino. (Miliauskas, 2024)

Lent. 2, sudarytoje remiantis „*AMLyze*“ sudarytu straipsniu apie didžiausias AML baudas 2023 m. matoma, kad finansų sektoriuje „*Deutsche Bank*“ susidūrė su 186 mln. USD bauda dėl nepakankamų kovos su pinigų plovimu procedūrų, neadekvačių rizikos vertinimo sistemų ir problemų, susijusių su duomenų valdymu. „*Bank of Queensland*“ gavo 50 mln. USD baudą už teisingumo standartų nesilaikymą ir AML įstatymų pažeidimus. Tuo tarpu lošimų sektoriuje „*William Hill*“ buvo nubausta 19,2 mln. GBP už socialinės atsakomybės ir kovos su pinigų plovimu procedūrų trūkumus. (Miliauskas, 2024)

2 lentelė

Didžiausios AML baudos 2023 m.

Baudos dydis	**Institucijos pavadinimas**	**Sektorius**	**Priežastis**
4.3 milijardų USD	Binance	Kriptovaliutų birža	Dalyvavimas pinigų plovime, nelicencijuotas pinigų pervedimas, sankcijų pažeidimai
450 milijonų USD	Crown Resorts	Kazino	Kovos su pinigų plovimu pažeidimai ir pinigų plovimo bei teroristų finansavimo rizikos neįvertinimas Melburno ir Perto kazino
186 milijonai USD	Deutsche Bank	Banking	Nepakankama kovos su pinigų plovimu kontrolė ir programos, rizikos ir duomenų valdymo problemos
50 milijonų USD	Bank of Queensland	Banking	Teisingumo standartų pažeidimas ir kovos su pinigų plovimu įstatymų nepaisymas
19,2 milijonų GBP	William Hill	Betting/Online casino	Socialinės atsakomybės ir kovos su pinigų plovimu procedūrų neįgyvendinimas

Šaltinis: (Miliauskas, 2024)

Šie atvejai pabrėžia, kad ne tik finansų, bet ir kitose industrijose, tokiose kaip kriptovaliutų prekyba, kazino ar lošimų paslaugos, stiprios KYC ir AML kontrolės procedūros yra būtinos. Jos ne tik užtikrina reguliavimo reikalavimų laikymąsi, bet ir apsaugo organizacijų reputaciją bei klientų pasitikėjimą.

“Silpna CDD (deramo klientų tikrinimo) praktika ir neveiksmingos KYC sistemos gali paversti finansų įstaigas finansinių nusikaltimų skatintojomis, o tai gali lemti dideles baudas ir žalą reputacijai.“ Tikimasi, kad finansų įstaigos vis dažniau naudos pažangias technologijas, pavyzdžiui, dirbtinį intelektą ir mašininį mokymąsi, kad padidintų savo KYC procesų tikslumą ir efektyvumą. Šalindamos šias spragas organizacijos gali ne tik išvengti baudų, bet ir prisidėti prie saugesnės pasaulinės finansų ekosistemos (Endorsed Regulatory Fines And Enforcement Actions Relating To CDD/KYC Failures, 2024)

1.4. Šiuolaikinių sprendimų technologiniai pagrindai

Kaip pabrėžia Evmorfia Biliri, Minas Pertselakis, Marios Phinikettos ir kiti savo darbe (2019), skirtame patikimos duomenų tarpininkavimo sistemos aviacijoje kūrimui, „Aviacijos pramonė gali gauti naudos iš struktūrizuotų, mašinomis apdorojamų duomenų susitarimų, kuriuose atsižvelgiama į privatumo, intelektinės nuosavybės ir jautrumo klasifikacijas. Tas pats požiūris gali pagerinti KYC procesus, užtikrinant, kad klientų duomenys aviacijoje būtų atsakingai valdomi ir saugomi. Priimdamas sistemas, kuriomis remiamas patikimas dalijimasis duomenimis, aviacijos sektorius gali skatinti patikimesnius KYC protokolus, derindamas duomenų įžvalgų poreikį su griežtais privatumo reikalavimais ir teisės aktų laikymusi.“. Visgi, tam, kad implementuoti praktinius sprendimus, svarbu suprasti ir pačių technologijų teorinį pagrindą. (Biliri, et al., 2019)

1.4.1. Mašininis mokymasis

Mašininis mokymasis (angl. *Machine Learning, ML*) yra dirbtinio intelekto poskyris, leidžiantis sistemoms mokytis ir tobulėti iš duomenų jų aiškiai neprogramuojant (Mahesh, 2019). Kaip knygoje „*Machine Learning Algorithms - A Review*“ (Mašininio mokymosi algoritmai - apžvalga) pabrėžė Batta Mahesh, ML plačiai taikomas įvairiose srityse - nuo prognozavimo analizės iki natūralios kalbos apdorojimo ir vaizdų atpažinimo.

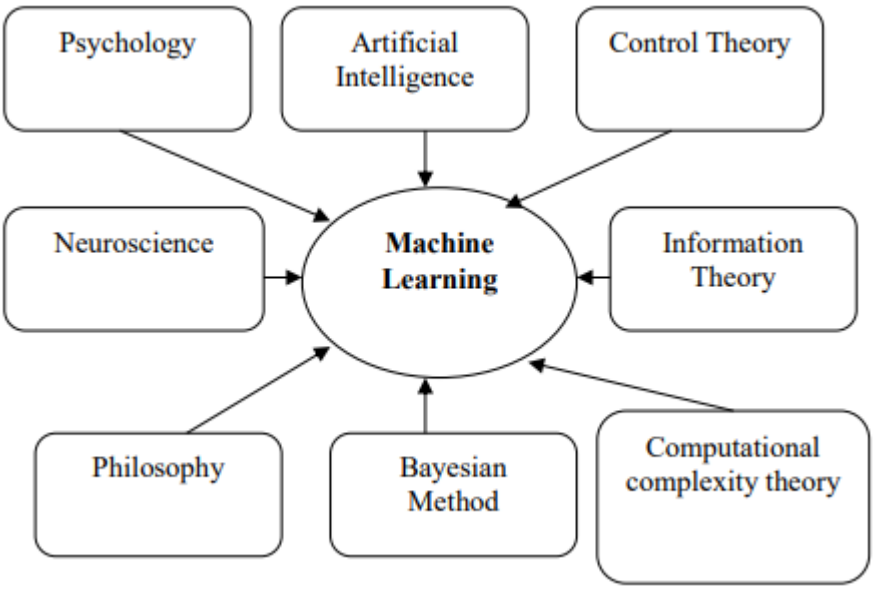
ML veikia pagal principą, kai iš duomenų mokomasi dėsningumų, kad būtų galima priimti sprendimus arba prognozuoti. Arthuras Samuelis, dažnai laikomas šios srities pradininku, apibūdino ML kaip suteikiančią „kompiuteriams galimybę mokytis be aiškaus programavimo“

(Petrik, 2017). Ši technologija tapo labai svarbi tvarkant didžiulius duomenų rinkinius, kai tradiciniai programavimo metodai nėra efektyvūs.

Pasak Batta Mahesh (2019), mašininį mokymąsį galima kategorizuoti į 5 kategorijas:

- Prižiūrimas mokymasis (angl. Supervised learning) remiasi pažymėtais duomenų rinkiniais, kai norimas rezultatas jau yra žinomas. Tokie algoritmai kaip sprendimų medžiai, atraminių vektorių mašinos (SVM) ir neuroniniai tinklai analizuoja mokymo duomenis, kad nuspėtų rezultatus arba klasifikuotų informaciją. Pavyzdžiui, sprendimų medžiai naudoja hierarchinę mazgų ir šakų struktūrą, kad priimtų sprendimus pagal sąlygas.
- Skirtingai nuo prižiūrimo mokymosi, neprižiūrimas mokymasis (angl. *Unsupervised learning*) veikia su neženklintais duomenimis. Juo siekiama atskleisti paslėptus duomenų rinkinių modelius ar struktūras. Tokie metodai, kaip K-Means klasterizavimas ir pagrindinių komponenčių analizė (PCA), dažniausiai naudojami duomenų dimensijai sumažinti arba panašiems duomenų taškams sugrupuoti.
- Pusiau prižiūrimas mokymasis (angl. *Semi-Supervised learning*) derina paženklintus ir nepaženklintus duomenis, todėl tinka tais atvejais, kai gauti paženklintus duomenis yra brangu arba užima daug laiko. Tokie algoritmai kaip transdukcinės atraminių vektorių mašinos (angl. *Transductive Support Vector Machines, TSVM*) taiko hibridinę metodiką neženklintiems duomenims ženklinti, kartu maksimaliai padidinant skirtumą tarp klasių.
- Sustiprintas mokymas (angl.*Reinforcement Learning - RL*) orientuotas į sistemų mokymą priimti nuoseklius sprendimus maksimizuojant kaupiamąjį atlygį. RL buvo labai svarbus tokiose srityse kaip robotika, žaidimų kūrimas ir autonominės sistemos.
- Ansamblinis mokymasis (angl. *Ensemble learning*) sujungia kelis algoritmus, kad būtų pasiektas geresnis prognozavimo našumas. Tokie metodai kaip „Bagging“ ir „Boosting“ didina tikslumą mažindami perteklinį pritaikymą ir dispersiją.

Mašininis mokymasis (angl. *Machine learning, ML*) yra daugiadisciplininė sritis, pagrįsta įvairiais teoriniais ir praktiniais pagrindais. Kaip pabrėžiama Jafar Alzubi ir kt. darbe, ML remiasi įvairiomis disciplinomis, tokiomis kaip dirbtinis intelektas, neuromokslai, psichologija, valdymo teorija, informacijos teorija, Bajeso metodai, kompiuterinio sudėtingumo teorija ir filosofija. Kiekviena iš šių sričių suteikia unikalių perspektyvų ir metodologijų, praturtinančių ML algoritmų kūrimą ir taikymą.



Šaltinis: *(Alzubi, Nayyar, & Kumar, 2018)*

4 pav. Daugiadisciplininis mašininis mokymasis

Pridedamoje diagramoje (4 pav.) pavaizduotas šis daugialypis ML pobūdis, atskleidžiantis jo teorines sąsajas. Pavyzdžiui:

- Dirbtinis intelektas (DI): ML sudaro pagrindinį dirbtinio intelekto poaibį, suteikiantį sistemoms galimybę mokytis ir priimti sprendimus remiantis duomenimis.
- Neurologija ir psichologija: Šios disciplinos įkvepia ML modelius, pavyzdžiui, neuroninius tinklus, kurie imituoja žmogaus smegenų struktūrą ir veikimą.
- Valdymo teorija: Tai padeda optimizuoti ir stabilizuoti mašininio mokymosi sistemas, ypač priimant sprendimus realiuoju laiku.
- Informacijos teorija: Ši teorija yra labai svarbi siekiant suprasti duomenų atvaizdavimą ir perdavimą, ja grindžiami daugelis ML procesų, pavyzdžiui, požymių atranka ir suspaudimas.
- Bajeso metodai: Bajeso metodai: statistiniai metodai, tokie kaip Bajeso išvada, padeda prognozuoti modeliavimą ir tikimybinius samprotavimus ML.
- Skaičiavimo sudėtingumo teorija: Ši teorija leidžia suprasti ML algoritmų skaičiavimo efektyvumą ir apribojimus.
- Filosofija: ML taip pat kelia filosofinius klausimus, susijusius su sprendimų priėmimu, etika ir intelekto prigimtimi.

Integruojant šių sričių sąvokas, ML tapo veiksminga sistema, leidžiančia kurti protingas sistemas, galinčias spręsti sudėtingas problemas. Suprasdami šias pamatines teorijas, mokslininkai

gali patobulinti ML algoritmus, pagerinti jų patikimumą, mastelio keitimą ir etinį suderinamumą įvairiose srityse.

Taip pat, Jafar Alzubi ir kt. darbe pateikiami įvairūs mašininio mokymosi (ML) taikymo būdai (lent 3), atskleidžiantys, kaip ši technologija gali būti pritaikoma skirtingose srityse – nuo žaidimų iki autonominių transporto priemonių ir laiškų filtravimo.

3 lentelė

Mašininio mokymosi pritaikymas

Taikymas	**Aprašymas**
Šaškių žaidimas	Kompiuterio programa mokosi žaisti šaškių žaidimą, gerina savo rezultatus, kuriuos lemia jos gebėjimas laimėti įvairios klasės užduotis, susijusias su šiuo žaidimu, remdamasi patirtimi, įgyta žaidžiant žaidimus su pačia savimi.
Kalbos atpažinimas	Šiuo metu sudėtingiausiose kalbos atpažinimo sistemose tam tikromis formomis naudojami mašininio mokymosi algoritmai. Pavyzdys: SPHINX sistema [20] iš kalbos signalų mokosi konkretaus kalbėtojo garsų ir žodžių. Įvairios neuroninių tinklų mokymosi metodikos, skirtos paslėptiesiems Markovo modeliams interpretuoti, yra labai veiksmingos automatiškai pritaikant kalbėtojus, žodyną, triukšmą ir pan.
Autonominiai automobiliai	Mašininio mokymosi modeliai šiomis dienomis taikomi autonominėms transporto priemonėms, pvz., automobiliams, dronams ir kt. Pavyzdys: Pavyzdžiui, „Google“ automobiliai be vairuotojo, „Tesla“ automobiliai. Mašininio mokymosi metodai taip pat labai veiksmingi valdant jutikliais pagrįstas programas.
Laiškų filtravimas (nepageidaujami laiškai)	Mašininis mokymasis gali būti taikomas nepageidaujamiems el. laiškams filtruoti. Mašininiu mokymusi pagrįstas modelis tiesiog įsimins visus laiškus, kuriuos naudotojas klasifikuos kaip nepageidaujamus laiškus. Kai į pašto dėžutę gaunamas naujas el. laiškas, mašininiu mokymusi paremtas modelis ieškos, lygins ir remsis ankstesniais nepageidaujamais el. laiškais. Jei naujas laiškas atitiks kurį nors iš jų, jis bus pažymėtas kaip nepageidaujamas; priešingu atveju jis bus perkeltas į naudotojo pašto dėžutę.
Robotika ir dirbtinis intelektas	Mašininis mokymasis laikomas patobulintu problemų sprendimo metodu. Naudojant bazines žinias ir mokymo duomenis su mašininio mokymosi modeliais, galima patobulinti mokymąsi, o tai leis robotikai ir dirbtiniam intelektui pereiti į naujos kartos lygį.
Žiniatinklis ir socialinė žiniasklaida	- Naiviojo Bajeso klasifikatoriai sėkmingai taikomi teksto gavybos srityje, filtruojant nepageidaujamus laiškus arba klasifikuojant tinklalapį, el. laišką ar kitą dokumentą. „Facebook“ naudoja „Naïve Bayes“ teigiamoms ir neigiamoms emocijoms išreikšti. Dokumentų kategorizavimas: „Google“ naudoja Naïve Bayes algoritmą dokumentams kategorizuoti. K-vidurkių klasterizavimą naudoja tokios paieškos sistemos, kaip „Google“, „Yahoo“, kad sugrupuotų tinklalapius pagal panašumą. Apriori naudoja tokios interneto svetainės kaip „Amazon“ ar „Flipkart“, kad rekomenduotų, kurios prekės dažnai perkamos kartu. Kitas dažnas „Apriori“ taikymo būdas - „Google“ automatinis papildymas. Kai asmuo įveda žodį, „Google“ paieškos sistema ieško kitų susijusių žodžių, kurie eina kartu su anksčiau įvestu žodžiu. Nuotaikų analizė socialinių tinklų svetainėse yra tipiška teksto klasifikavimo problema, sprendžiama taikant įvairius ML algoritmus
Medicinos sritis	TRISS: Traumos ir sužalojimų sunkumo balas, plačiai naudojamas sužeistų pacientų mirtingumui prognozuoti, iš pradžių buvo sukurtas Boydo ir kt. naudojant logistinę regresiją. Daugelis kitų medicininių skalių, naudojamų paciento sunkumui įvertinti, buvo sukurtos naudojant logistinę regresiją.

1 lentelės tęsinys

Taikymas	Aprašymas
Bajeso metodai	Bajeso teorema yra vienas iš populiariausių metodų sąlyginėms tikimybėms apskaičiuoti. hipotezių rinkinį. Ją galima naudoti sprendžiant sudėtingas duomenų mokslo ir analizės problemas integruojant su įvairiais mašininio mokymosi modeliais ir algoritmais. Keletas realaus pasaulio pavyzdžių problemų, kurias galima išspręsti naudojant Bajeso metodus, yra šie: • Pasiūlymai dėl „Netflix • Automatinis taisymas

Šaltinis: (Alzubi, Nayyar, & Kumar, 2018)

Atsižvelgiant į platų ML pritaikymo spektrą, jis susiduria su daugybe iššūkių. Kai kurie iš jų yra šie (Alzubi, Nayyar, & Kumar, 2018):

- Mašininio mokymosi algoritmams, kad jie būtų tikslūs ir veiksmingi, reikia didelių duomenų kiekių, kurie mokslininkams vis dar neprieinami. Tokie technologijų gigantai kaip „Facebook“ ir „Google“ turi prieigą prie tokių milžiniškų duomenų, todėl jie pirmauja dirbtinio intelekto srityje. Dar sunkiau tokius duomenis gauti tokiose srityse kaip bankininkystė ir sveikatos priežiūra, kur skaitmeninių duomenų yra nedaug, todėl sunku atlikti tikslias prognozes.
- Šlamšto aptikimas: Iki šiol sukurtos intelektinės sistemos vis dar nesugeba teisingai aptikti nepageidaujamų laiškų. Dėl to nepageidaujami laiškai patenka į pašto dėžutę, o nepageidaujami laiškai patenka į nepageidaujamų laiškų katalogą.
- Mašininio mokymosi algoritmai dar nesėkmingai atpažįsta objektus ir vaizdus. Ši sritis tebėra atviras mašininio mokymosi tyrimų laukas. Nors paminėjome keletą mašininio mokymosi iššūkių, yra dar daug sričių, kuriose giliojo mokymosi algoritmai vis dar susiduria su iššūkiais, t. y. kalbos supratimas, kreditinių kortelių sukčiavimo aptikimas, veidų aptikimas, skaitmenų atpažinimas pagal pašto kodą, produktų rekomendavimas ir kt.

Nepaisant įspūdingų pasiekimų, šios technologijos vystymasis vis dar susiduria su daugybe iššūkių, įskaitant didelių duomenų poreikį ir tikslių algoritmų tobulinimą. Tačiau, remiantis tyrimų duomenimis, ML potencialas yra milžiniškas, o jo ateitis – neatsiejama nuo mūsų kasdienio gyvenimo ir verslo sektoriaus pažangos.

Didelė mašininio mokymosi (ML) taikymo sritis susiduria su daugybe iššūkių. (Janiesch, Zschech, & Heinrich, 2021) pabrėžia, kad „ML pažanga leido neseniai sukurti protingas sistemas,

kurių pažinimo gebėjimai panašūs į žmogaus“, tačiau jie įspėja, kad ML sistemų įgyvendinimas realiomis verslo sąlygomis kelia didelių sunkumų, pavyzdžiui, duomenų iškraipymas ir modelių interpretavimas. Šie apribojimai išryškėja, kai mašininio mokymosi sistemos diegiamos prognozavimo užduotims atlikti sudėtingoje aplinkoje, kurioje istoriniai duomenys gali nebeatspindėti dabartinės realybės.

Zhou (2022) pabrėžia unikalias kliūtis, su kuriomis susiduriama taikant atvirosios aplinkos mašininį mokymąsi (*open ML*). Skirtingai nuo tradicinių artimos aplinkos ML modelių, atvirasis ML turi prisitaikyti prie besikeičiančių duomenų pasiskirstymų, laipsniško mokymosi tikslų ir atsirandančių naujų duomenų klasių. Zhou aiškina: „Artimos aplinkos prielaidos siūlo supaprastintą abstrakciją..., tačiau jos neatspindi dinamiškų realaus pasaulio užduočių sudėtingumo.“ Jis taip pat pažymi, kad svarbu sudaryti sąlygas modeliams tobulėti palaipsniui, neperkvalifikuojant jų iš naujo, ypač scenarijuose, kuriuose duomenys kaupiasi laikui bėgant kaip srautas.

Šie pastebėjimai pabrėžia, kad nors ML potencialas sukelti revoliuciją pramonės šakose išlieka didžiulis, šių pamatinių iššūkių sprendimas bus labai svarbus siekiant sukurti patikimas, pritaikomas ir etiškai pagrįstas išmaniąsias sistemas.

1.4.2. Mašininio mokymosi pritaikymas aviacijai

Kaip aiškina Sinha ir Kaul (2018), „pažink savo klientą“ (angl. *Know Your Customer, KYC*) „naudojamas tapatybės patikrinimui bankuose, ligoninėse, „*Aadhar*“, pasų patikrinimui ir kt.“, kur jis veikia kaip centralizuotas tapatybės patvirtinimo ir sukčiavimo prevencijos metodas. Jie apibūdina KYC kaip itin svarbų procesą, užtikrinantį duomenų vientisumą ir klientų pasitikėjimą, tradiciškai pasikliaujant sistemomis, kuriose neskelbtini duomenys saugomi centralizuotai, o tai gali kelti susirūpinimą dėl saugumo ir galimų vieno gedimo taškų. Siekdami išspręsti šias problemas, Sinha ir Kaul siūlo blokų grandine pagrįstą decentralizuotą KYC sistemą, kuri „užtikrina duomenų replikavimą, atsarginę duomenų kopiją ir vieno gedimo taško nebuvimą“, siūlydami nepažeidžiamą ir saugų klientų informacijos saugojimo būdą be „trečiųjų šalių dalyvavimo“.

Kaip taip pat pažymi Mansoor (2023) ir kt., tradiciniai KYC procesai dažnai apibūdinami kaip „nepatikimi ir brangūs“, ypač dėl jų priklausomybės nuo centralizuotos duomenų valdymo sistemos. Blockchain technologija siūlo sprendimą šiems iššūkiams: decentralizuotos technologijos dėka galima sukurti „nepakeičiamą ir saugią klientų duomenų bazę“, kuria

institucijos galėtų dalintis be tarpininkų dalyvavimo ir taip sumažinti klaidų bei piktnaudžiavimo galimybes.

Nors KYC procesai yra nusistovėję tokiuose sektoriuose kaip bankininkystė ir sveikatos priežiūra, aviacijos pramonei, kuriai taikomi unikalūs reguliavimo reikalavimai ir didelės vertės sandoriai, reikalaujantys griežto klientų ir partnerių patikrinimo, pritaikytų tyrimų ir įgyvendinimo apimtys yra ribotos. Skirtingai nuo kitų pramonės šakų, aviacijoje reikia įvertinti daugiau rizikos veiksnių, įskaitant atitiktį tarptautinėms taisyklėms, saugumo standartams ir galimas sąsajas su didelės rizikos jurisdikcijomis. Atsižvelgiant į aviacijos sandorių sudėtingumą ir apimtį, tradicinės KYC sistemos, net ir patobulintos blokų grandinės decentralizacijos tikslais, gali nespėti prisitaikyti prie sparčiai besikeičiančių sukčiavimo taktikų ir sudėtingų klientų profilių.

Būtent čia mašininis mokymasis (ML) siūlo perspektyvų sprendimą. ML grindžiamos sistemos gali analizuoti didelius duomenų rinkinius, nustatyti sudėtingus modelius ir realiuoju laiku prisitaikyti prie naujų sukčiavimo rodiklių, užtikrindamos dinamišką ir patikimą požiūrį į klientų patikrą. Nagrinėdami ML grindžiamą KYC modelį, pritaikytą specialiai aviacijai, siekiame sukurti sistemą, kuri ne tik padidintų potencialios rizikos nustatymo tikslumą, bet ir prisitaikytų prie besikeičiančios sukčiavimo taktikos ir įvairių reguliavimo reikalavimų. ML gali palengvinti nuolatinę stebėseną ir automatinį rizikos vertinimą, todėl tai yra galinga priemonė, galinti patobulinti aviacijos pramonės KYC procesus ir padėti bendrovėms priimti pagrįstus sprendimus apie potencialius klientus ir partnerius.

1.4.3. Žiniatinklio duomenų nuskaitymas

Tinklapių nuskaitymas (angl. *web scraping*) arba žiniatinklio naršymas (angl. *web crawling*) - tai automatinis duomenų išgavimas iš svetainių naudojant programinę įrangą. Tai procesas, kuris ypač svarbus tokiose srityse kaip verslo išmanyme (angl. *Business inteligence*) šiuolaikiniame amžiuje. Žiniatinklio duomenų nuskaitymas - tai technologija, leidžianti iš teksto, pavyzdžiui, HTML, išgauti struktūrizuotus duomenis. Šis metodas labai naudingas tais atvejais, kai duomenys nepateikiami mašininio skaitymo formatu, pavyzdžiui, JSON arba XML. (Khder, 2021)

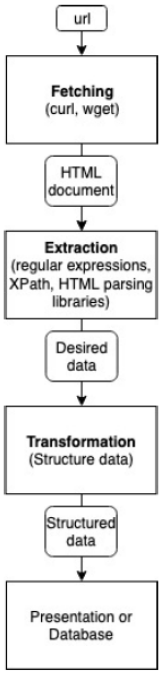
Pagal Mine Dogucu ir Mine Çetinkaya-Rundel (2020) yra dviejų tipų žiniatinklio nuskaitymo būdai:

- Pirmasis - ekrano nuskaitymas, kai duomenis išgaunate iš svetainės išeities kodo, naudodami HTML analizatoriumi arba reguliariosios išraiškos atitikmeniu.

- Antrasis yra naudojant programų programavimo sąsajas, paprastai vadinamas API. Šiuo atveju svetainė siūlo struktūrizuotas HTTP užklausas, pagal kurias grąžinami JSON arba XML failai.

Pagal Bo Zhao (2022), žiniatinklio duomenų nuskaitymas yra galinga priemonė, naudojama didelėms heterogeniškoms duomenų bazėms rinkti ir analizuoti. Šios technologijos dėka galima greitai ir efektyviai gauti vertingą informaciją iš didžiulio skaičiaus internetinių šaltinių, tokių kaip HTML, XML, JSON ar multimedijos failai.

Žiniatinklio duomenų nuskaitymo procesas susideda iš dviejų pagrindinių etapų: interneto resursų gavimo ir norimos informacijos ištraukimo. Kaip paaiškina Zhao (2022), šis procesas prasideda siunčiant HTTP užklausą į tikslinį tinklalapį, kuris grąžina resursą, pavyzdžiui, HTML kodą, XML ar JSON failą. Vėliau, naudojant tokias priemones kaip „*BeautifulSoup*“ ar „*Scrapy*“, iš šių resursų ištraukiama ir apdorojama reikiama informacija. Šis procesas leidžia duomenis pertvarkyti į tvarkingą struktūrizuotą rinkinį, kurį galima naudoti analizei ar integruoti į kitas sistemas.



Šaltinis: (Persson, 2019)

5 pav. Žiniatinklio nuskaitymo procesas

Tinklapių nuskaitymui paaiškinti gali būti išskiriama ir daugiau etapų. Pavyzdžiu, pasak Persson (2019), jis apima keturis pagrindinius etapus: Ištraukimas, kai HTML turinys gaunamas naudojant tokias priemones kaip “*curl*“ ar “*wget*“; išskyrimas, kai išskiriami konkretūs duomenys naudojant tokius metodus kaip reguliariosios išraiškos ar HTML analizatoriai; transformavimas, kai išskirtieji duomenys konvertuojami į struktūrizuotus formatus, pavyzdžiui, lenteles ar JSON;

ir pateikimas, kai struktūrizuoti duomenys saugomi arba rodomi tolesniam naudojimui. Šis procesas labai svarbus tokioms programoms kaip klientų profiliavimas ir rinkos analizė.

Žiniatinklio nuskaitymas turi daugybę praktinių pritaikymo sričių. Dogucu ir Çetinkaya-Rundel (2021) pažymi, kad ši technologija leidžia automatizuotai rinkti duomenis apie rinkos kainas, produktų atsiliepimus, oro sąlygas ar net stebėti socialinių tinklų informaciją. Be to, žiniatinklio nuskaitymas plačiai naudojamas e-komercijos sektoriuje, pavyzdžiui, kainų palyginimo tinklalapiuose ar konkurentų kainų stebėjimui. Valstybinės statistikos agentūros taip pat pasitelkia šią technologiją, norėdamos efektyviau rinkti ir analizuoti duomenis, pavyzdžiui, vartotojų kainų indeksams apskaičiuoti ar mokesčių pajamoms stebėti

Nepaisant žiniatinklio nuskaitymo galimybių, ši technologija susiduria su tam tikrais iššūkiais, įskaitant teisinius ir etinius klausimus. Pagal Zhao (2022), kai kurios svetainės aktyviai riboja automatizuotus užklausų srautus, įdiegiant saugumo priemones, tokias kaip CAPTCHA ar IP blokavimas. Be to, tam tikrų tinklalapių duomenų naudojimas gali pažeisti autorių teises ar tinklalapio naudojimo sąlygas. Tačiau, kaip teigia autoriai Dogucu ir Çetinkaya-Rundel (2021), tinkamas reguliavimo išmanymas ir atsakingas technologijos naudojimas gali padėti įveikti šias kliūtis ir maksimaliai išnaudoti žiniatinklio nuskaitymo potencialą

Apibendrinant, žiniatinklio duomenų nuskaitymas yra neatsiejama šiuolaikinės duomenų analizės ir statistikos praktikos dalis, suteikianti galimybę efektyviai gauti ir naudoti didelius kiekius aktualių duomenų. Ši technologija ne tik supaprastina duomenų rinkimo procesą, bet ir atveria naujas galimybes verslui ir moksliniams tyrimams.

1.4.4. Tekstinių duomenų apdorojimas naudojant NLP

Natūralios kalbos apdorojimas (NLP) - tai dirbtinio intelekto sritis, skirta suteikti mašinoms galimybę suprasti, interpretuoti ir prasmingai kurti žmogaus kalbą. Ši sritis pastaraisiais metais padarė didelę pažangą, derindama kompiuterinę lingvistiką su giliuoju mokymusi, kad būtų galima apdoroti ir analizuoti didžiulius tekstinių duomenų kiekius. Kaip pabrėžia K. R. Chowdhary (2020), NLP metodais siekiama užpildyti atotrūkį tarp žmonių bendravimo ir mašinų supratimo, apimant tokias užduotis kaip informacijos paieška, teksto apibendrinimas, kalbos vertimas ir nuotaikų analizė.

NLP iš esmės apima nestruktūrizuotų tekstinių duomenų konvertavimą į struktūrizuotus formatus, kuriuos gali apdoroti algoritmai. Šie etapai apima žymėjimą, kai tekstas suskirstomas į mažesnius komponentus, pavyzdžiui, žodžius ar frazes, ir žodžių įterpimą, kai šie žymenys

paverčiami skaitmeniniais vektoriais. Šie vektoriai išsaugo semantinius ryšius, todėl mašinos gali atpažinti teksto kontekstą ir niuansus.

Pažangūs NLP modeliai, pavyzdžiui, BERT (*Bidirectional Encoder Representations from Transformers*), naudoja dėmesio mechanizmus, kurie leidžia sutelkti dėmesį į svarbias sakinio dalis, o duomenis apdoroja dviem kryptimis. Šie modeliai iš esmės pakeitė tokias užduotis, kaip atsakymai į klausimus ir konteksto supratimas, nustatydami naujus teksto analizės tikslumo standartus.

NLP taikymas apima įvairias sritis, įskaitant sveikatos priežiūrą, finansus ir švietimą. Pavyzdžiui, bioinformatikoje NLP sėkmingai naudojamas baltymų sekoms iššifruoti, traktuojant jas kaip lingvistinius duomenis. Šis metodas parodo, kaip lingvistiniai modeliai gali peržengti tradicines ribas ir būti naudingi mokslo disciplinose. Be to, kaip pažymi Chowdhary, NLP vaidina pagrindinį vaidmenį kuriant sistemas, kurios gali natūraliai sąveikauti su žmonėmis, nesvarbu, ar tai būtų pokalbių robotai, virtualūs asistentai, ar teksto prognozavimo mechanizmai.

Nepaisant pažangos, NLP susiduria su iššūkiais, pavyzdžiui, susiduria su žmogaus kalbos dviprasmybėmis, tokiomis kaip daugiareikšmiškumas (žodžiai su keliomis reikšmėmis) ir sintaksinis kintamumas. Chowdhary pabrėžia, kad norint pasiekti gilesnį kalbos supratimą, būtina į NLP modelius integruoti semantines ir epizodines žinias. Be to, D. Ofer (2021) ir kt. pabrėžia, kad šiems modeliams apmokyti dažnai reikia didelių skaičiavimo išteklių ir didelių anotuotų duomenų rinkinių, todėl jų kūrimas yra sudėtingas ir reikalauja daug išteklių.

NLP iššūkius pabrėžia ir Fuchs (2023) „Nors NLP modeliai yra perspektyvūs analizuojant sudėtingus duomenų rinkinius, pernelyg didelis pasitikėjimas jais kartais gali pakenkti svarbiems sprendimų priėmimo procesams, o mokymo duomenų šališkumas gali padidinti analizės netikslumus.“

Apibendrinant galima teigti, kad NLP yra transformuojanti technologija, turinti toli siekiančią reikšmę įvairioms pramonės šakoms. Tobulėjant jos metodikoms, gebėjimas apdoroti ir suprasti žmogaus kalbą ir toliau tobulins automatizavimo ir sprendimų priėmimo procesus įvairiose srityse.

1.4.5. Lygiagretieji skaičiavimai ML pagrįstuose sprendimuose

Lygiagretieji skaičiavimai - tai metodas, kai vienu metu suskaidomos ir apdorojamos užduotys, kad skaičiavimai vyktų greičiau ir efektyviau, ypač sprendžiant didelės apimties mašininio mokymosi (ML) problemas. Šis metodas ypač svarbus ML taikomosiose programose,

kai tvarkant didelius duomenų rinkinius ir atliekant sudėtingus skaičiavimus reikia optimaliai panaudoti aparatinės ir programinės įrangos išteklius. Lygiagretinimo revoliuciją sukėlė bendrosios paskirties grafikos procesorių (GPGPU), tensorių procesorių (TPU) ir daugiabranduolinių procesorių pažanga.

Šiuolaikinėms ML užduotims lygiagretus vykdymas labai naudingas, nes taikomi tokie metodai, kaip:

- Duomenų lygiagretinimas, kuris padalina duomenis į poaibius, kad juos būtų galima apdoroti vienu metu, išlaikant sinchronizuotą modelį. Šis metodas yra pagrindinis tokiose sistemose kaip "TensorFlow" ir "PyTorch", kaip pabrėžė Xu ir kiti (2021), kurie pademonstravo duomenų lygiagretumo taikymą trilijono parametrų modeliams mokyti.
- Modelio lygiagretumas, kuriuo įvairios ML modelio dalys, pavyzdžiui, sluoksniai arba svoriai, paskirstomos keliems procesoriams. Metz ir kiti (2021) pabrėžė jo veiksmingumą mokant sudėtingus giliuosius neuroninius tinklus (DNN), kurie viršija vieno procesoriaus atminties talpą.
- Vamzdynų lygiagretumas, kuriuo skaičiavimo grafikai suskirstomi į etapus, kurių kiekvieną tvarko skirtingi įrenginiai. Šis metodas, kurį nagrinėjo Xu ir kiti (2021), sušvelnina nuosekliojo apdorojimo sukeliamus trikdžius.

GPGPU atlieka pagrindinį vaidmenį atliekant lygiagrečiuosius ML skaičiavimus. Naudodamiesi tokiomis sistemomis kaip CUDA, kūrėjai gali efektyviai įgyvendinti labai lygiagretinamus neuroninių tinklų skaičiavimus. Kaip nurodo Metz et al. (2021), konvoliuciniams neuroniniams tinklams (CNN), kuriems reikia trilijonų skaičiavimų per sekundę, labai naudingas GPGPU pagrįstas lygiagretinimas. Įvertinę ML užduočių našumą ir energijos suvartojimą GPGPU, inžinieriai gali pasirinkti efektyviausią aparatinę įrangą dar kūrimo proceso pradžioje.

Dar vienas iš lygiagrečiųjų skaičiavimų taikymo pavyzdžių - kredito rizikos vertinimas naudojant „*Random Forest*" algoritmą. Kaip pabrėžė Hentosh et al. (2023), Random Forest algoritmo lygiagretinimas leido gerokai sutrumpinti duomenų rinkinio apdorojimo ir modelio mokymo laiką. Jų darbas parodė, kad naudojant 12 gijų dvylikos branduolių procesoriuje laikas sumažėjo daugiau nei šešis kartus, palyginti su nuosekliuoju apdorojimu. Šis efektyvumo padidėjimas pabrėžia lygiagrečiųjų algoritmų potencialą apdorojant didelės apimties duomenis ir spartinant skaičiavimus ML pagrįstuose sprendimuose.

Taip pat, Xu ir kt. pristatytoje sistemoje „*General and Scalable Parallelization for ML Computation Graphs*" (GSPMD) (2021) pabrėžiama, kaip galima supaprastinti didelės apimties modelių lygiagretinimą. GSPMD palaiko duomenų, modelio ir jo vykdymo (angl. *pipeline*)

lygiagretinimą vieningoje sistemoje, užtikrindama mastelio keitimą tūkstančiuose įrenginių, pavyzdžiui, „Cloud TPUv3“ branduolių. Ši sistema padėjo mokyti trilijonų parametrų modelius tokiose srityse kaip natūralios kalbos apdorojimas ir vaizdų atpažinimas.

Lygiagretinimo privalumai ML pagal Metz.:

1. Efektyvumas: Dėl vienalaikio užduočių vykdymo gerokai sumažėja mokymo ir išvadų darymo laikas.
2. mastelio didinimas: Galimybė apdoroti didžiulius duomenų rinkinius ir didelius modelius naudojant paskirstytąsias sistemas.
3. Išteklių optimizavimas: Efektyvus techninės įrangos išteklių, tokių kaip atmintis ir skaičiavimo vienetai, naudojimas.

Nors lygiagretinimas teikia didžiulę naudą, jis neapsieina be iššūkių:

- Ryšių perviršis: Metz et al. (2021), aptardami paskirstytąsias ML sistemas, atkreipė dėmesį į šią problemą.
- Apkrovos balansavimas: Užtikrinti vienodą užduočių paskirstymą išlieka iššūkis, ypač heterogeninėse architektūrose.
- Masteliškumas: Xu et al. (2021) pažymėjo, kad linijinio mastelio pasiekimas didėjant techninei įrangai išlieka sudėtingas, bet būtinas ateities tyrimų tikslas.

Apibendrinant galima teigti, kad lygiagretinimas yra svarbiausias šiuolaikinių ML sistemų mastelio keitimo ir efektyvumo veiksnys. Tobulindami tokias sistemas kaip GSPMD ir diegdami aparatinės įrangos naujoves, tokie tyrėjai kaip Metz et al. (2021) ir Xu et al. (2021) toliau tiesia kelią efektyviems didelės apimties ML skaičiavimams. Ateityje daugiausia dėmesio reikėtų skirti komunikacijos pridėtinėms išlaidoms mažinti ir užtikrinti sklandų lygiagrečiųjų skaičiavimų integravimą įvairiose ML taikomosiose programose.

1.4.6. Pažangūs sukčiavimo aptikimo mechanizmai aviacijoje

Sukčiavimo aptikimas aviacijos sektoriuje kelia unikalių iššūkių dėl skirtingų šios pramonės šakos duomenų struktūrų ir sparčiai besikeičiančios sukčiavimo veiklos. Kaip pabrėžiama Mehmedo Taha Araso ir Mehmeto Amaço Güvensano dokumente „Sukčiavimo aptikimo aviacijos sektoriuje iššūkiai ir pagrindiniai aspektai“ (2021), įprastinis rėmimasis taisyklėmis grindžiamomis sistemomis turi didelių trūkumų. Šios sistemos, nors ir veiksmingos statiniams modeliams, sunkiai kovoja su „nulinės dienos“ sukčiavimo atakomis - naujomis ir nenuspėjamomis schemomis, kurių tradiciniai metodai nesugeba aktyviai aptikti. Be to,

taisyklėmis grindžiamos sistemos labai priklauso nuo žmogaus patirties kuriant ir tobulinant sukčiavimo aptikimo taisykles, todėl jos yra linkusios klysti ir yra reaktyvios, o ne prevencinės.

Siekiant pašalinti šiuos trūkumus, aviacijos pramonėje vis dažniau naudojamas mašininis mokymasis (ML) ir pažangūs skaičiavimo metodai. ML metodai leidžia dinamiškai nustatyti sukčiavimo atvejus analizuojant didžiulių duomenų rinkinių, įskaitant keleivių vardų ir pavardžių įrašus, mokėjimų duomenis ir pagalbinę skrydžio informaciją, modelius. Dokumente pristatomos naujoviškos metodikos, pavyzdžiui, mažesnė atranka siekiant subalansuoti nesubalansuotus duomenų rinkinius, chronologinis mokymo ir testavimo duomenų išdėstymas, kad būtų galima realiai modeliuoti, ir sąnaudoms jautrios metrikos, kad būtų galima vertinti sistemas tiek tikslumo, tiek pelningumo požiūriu. Šie metodai ne tik pagerina aptikimo rodiklius, bet ir sumažina finansinius nuostolius, o tai labai svarbu oro transporto bendrovėms, susiduriančioms su nesąžiningais grįžtamaisiais mokesčiais.

Be to, gilaus mokymosi metodų, tokių kaip „*autoenkoderiai*“ ir dirbtiniai neuroniniai tinklai (ANN), taikymas pasirodė esąs veiksmingas nustatant sudėtingus sukčiavimo scenarijus. Gilusis mokymasis leidžia sistemoms atskleisti daugialypius sukčiavimo modelius, kurių dažnai nepastebi paviršutiniški ML modeliai. Tačiau tokie iššūkiai kaip atsako laikas, jautrumas sąnaudoms ir poreikis optimizuoti požymius tebėra itin svarbios tobulintinos sritys. Tyrime pabrėžiama požymių atrankos svarba, parodant, kad nereikšmingų duomenų sumažinimas gali gerokai padidinti sistemos našumą.

Ši sukčiavimo aptikimo evoliucija išryškina didėjantį adaptyvių, duomenimis pagrįstų sprendimų poreikį aviacijoje. Pasitelkus pažangius ML ir gilaus mokymosi metodus, pramonė gali peržengti reaktyvių priemonių ribas, atverdama kelią patikimesnėms ir keičiamo dydžio sukčiavimo aptikimo sistemoms. (Aras & Güvensan, 2021)

1.5. Duomenų privatumo ir teisiniai aspektai KYC automatizacijoje

Pažangių technologijų, tokių kaip mašininis mokymasis (ML), natūralios kalbos apdorojimas (NLP) ir žiniatinklio duomenų nuskaitymas, naudojimas „Pažink savo klientą“ procesuose labai padidino efektyvumą. Tačiau šios naujovės taip pat kelia svarbių iššūkių, susijusių su duomenų privatumu ir teisės aktų laikymusi. Nors šios priemonės padidina galimybes analizuoti didžiulius duomenų rinkinius, jos turi būti kruopščiai suprojektuotos ir įdiegtos, kad nebūtų pažeistos teisės į privatumą ar teisės aktai. Toliau nagrinėjame literatūrą šiais aspektais, išsamiai apžvelgdami problemas ir cituodami susijusius tyrimus.

1.5.1. Teisinės problemos, susijusios su mašininiu mokymusi KYC srityje

ML algoritmai, naudojami KYC sistemose, apdoroja neskelbtinus klientų duomenis, todėl kyla susirūpinimas dėl to, kaip šie duomenys saugomi, naudojami ir dalijamasi jais. Turksen et al. (2024) atliktame tyrime pabrėžiama, kad finansų įstaigose vis plačiau taikant AI/ML priemones, imta atidžiai tikrinti, ar galima paaiškinti, atskaitomybę ir atitiktį besikeičiantiems reguliavimo reikalavimams. Nors ML užtikrina neprilygstamą efektyvumą nustatant anomalijas ir automatizuojant rizikos vertinimą, jo sudėtingumas ir neskaidrumas dažnai trukdo reguliavimo institucijoms pasitikėti šiomis sistemomis.

Be to, pagal tokias teisines sistemas, kaip ES dirbtinio intelekto įstatymas ir Bendrasis duomenų apsaugos reglamentas (BDAR), reikalaujama, kad įstaigos užtikrintų, jog jų dirbtinio intelekto modeliai atitiktų etikos standartus, įskaitant skaidrumą ir sąžiningumą. Šiuose įstatymuose nurodoma naudoti „paaiškinamąjį AI“ - šis reikalavimas atsispindi Turkseno ir kt. išvadose, kuriose pabrėžiama, kaip svarbu, kad algoritmai būtų suprantami ir reguliavimo institucijoms, ir galutiniams naudotojams. (Turksen, Benson, & Adamyk, 2024)

1.5.2. Privatumo iššūkiai NLP ir žiniatinklio nuskaitymo srityje

NLP modelių integravimas į KYC procesus kelia unikalių iššūkių. Kaip aiškina Yin ir Habernal (2022), dideli transformatoriais pagrįsti NLP modeliai yra linkę į privatumo atakas, kai jautrią informaciją galima išgauti net iš nuasmenintų duomenų rinkinių. Dėl to buvo sukurti privatumo išsaugojimo metodai, pavyzdžiui, diferencinis privatumas, kuriais siekiama apsaugoti duomenis ir kartu išlaikyti NLP modelių veiksmingumą jautriose srityse.

Tinklapių nuskaitymas - galingas viešai prieinamos informacijos rinkimo metodas - dažnai veikia teisiškai pilkosiose zonose. Krotov (2020) pabrėžia, kad nors interneto duomenų nuskaitymas gali būti labai veiksmingas plečiant KYC procesus, jis turi atitikti intelektinės nuosavybės įstatymus, svetainių paslaugų teikimo sąlygas ir privatumo taisykles. Neatitikimas gali lemti teisinius ginčus arba žalą finansų įstaigų reputacijai.

1.5.3. Rekomendacijos dėl atitikties užtikrinimo

Norėdamos sumažinti teisines ir privatumo problemas, susijusias su KYC automatizavimu, įstaigos turi taikyti įvairiapusį požiūrį:

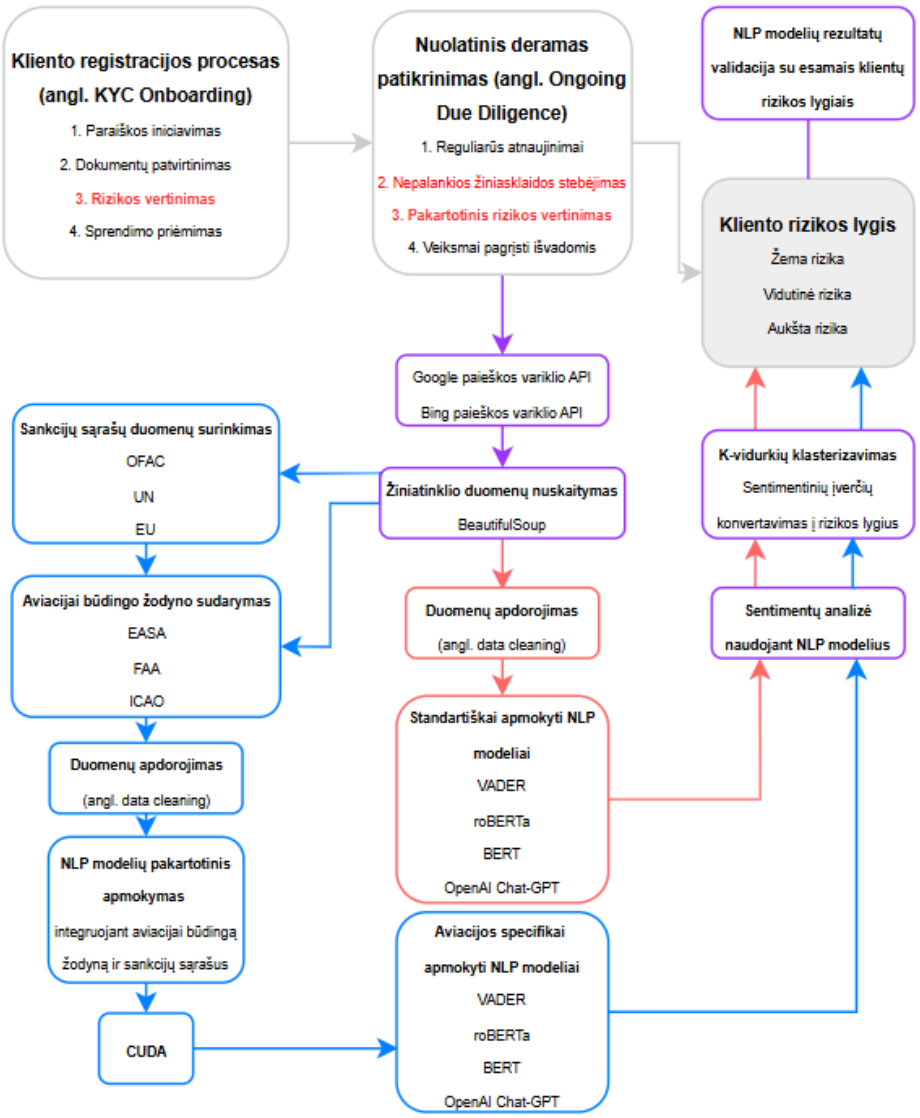
1. Diferencinio privatumo metodai: Kaip siūlo Yin ir Habernal, įstaigos savo ML ir NLP modeliuose turėtų įdiegti diferencinį privatumą, kad apsaugotų jautrią informaciją duomenų apdorojimo metu.
2. Paaiškinamumas dirbtinio intelekto modeliuose: Turksen ir kt. tyrime pabrėžiama būtinybė naudoti paaiškinamus dirbtinio intelekto metodus, kad būtų įvykdyti reguliavimo reikalavimai ir įgytas suinteresuotųjų šalių pasitikėjimas.
3. Etiška žiniatinklio nuskaitymo praktika: Vadovaudamosi Krotovo rekomendacijomis, organizacijos turėtų užtikrinti atitiktį teisiniams standartams, gaudamos aiškų sutikimą arba licencijas, kai iš interneto svetainių nuskaito duomenis.

Šios strategijos ne tik padeda organizacijoms laikytis teisinių pagrindų, bet ir didina visuomenės pasitikėjimą KYC procesais. Ateityje atliekant mokslinius tyrimus daugiausia dėmesio turėtų būti skiriama visuotinai priimtinoms etiško pažangiųjų technologijų naudojimo tokiose jautriose srityse kaip KYC gairėms parengti.

2. SIŪLOMAS SPRENDIMAS IR NAUDOTI METODAI

2.1. Automatizuotas internetinės reputacijos vertinimo modelis

Šiame skyriuje pristatomas automatizuotas modelis, skirtas vertinti klientų internetinę reputaciją, naudojant pažangius technologinius sprendimus, tokius kaip žiniatinklio duomenų nuskaitymas (angl. *Web Scraping*), natūralios kalbos apdorojimo (angl. *NLP*) metodai, lygiagretieji skaičiavimai ir mašininis mokymasis. Šis modelis yra pritaikytas KYC procesui, siekiant ne tik efektyviai identifikuoti ir įvertinti klientus, bet ir užtikrinti atitiktį reguliavimo reikalavimams. Toliau pateikiama išsami kiekvieno etapo analizė ir technologijų taikymo aprašymas.



Šaltinis: sudaryta autoriaus

6 pav. Siūlomas sprendimas

Remiantis standartinėmis KYC procedūromis aviacijos sektoriuje, identifikuotas 4 žingsnių procesas buvo pasirinktas kaip pagrindas rizikos vertinimo šakai, skirtai tolimesniems automatizacijos tyrimams. Tai padaryta dėl šios šakos plataus pritaikomumo, leidžiančio efektyviai integruoti mašininio mokymosi (ML) sprendimus. Svarbu pabrėžti, kad šie rizikos vertinimo veiksmai yra reguliariai kartojami, o tai dar kartą patvirtina modelio kūrimo svarbą. Sukurtas modelis leis užtikrinti automatizuotą sprendimą, kuris reguliariai atnaujins duomenis ir prisitaikys prie kintančių aplinkybių

Internetinės reputacijos duomenų gavimui pasitelkiami du populiariausi paieškos varikliai – Google ir Bing. Duomenų rinkimą bei automatizaciją palengvina šių paieškos sistemų teikiamas API funkcionalumas. Perduodant naujo ar esamo kliento pavadinimą kaip užklausą, API grąžina visą su tuo pavadinimu susijusią informaciją, kurią galima rasti šiuose paieškos varikliuose. Tai leidžia efektyviai rinkti duomenis ir juos panaudoti tolimesnei analizei ar sprendimų priėmimui.

Siekiant giliau ir automatizuotai analizuoti gautus paieškų rezultatus, naudojamas žiniatinklio nuskaitymo (angl. *web scraping*) įrankis, integruotas per „Python“ programavimo kalbos bibliotekas. Šio įrankio pagalba įeinama į kiekvieno Google ir Bing pateikto rezultato nuorodą, iš kurios esamas HTML kodas iššifruojamas ir išgaunama tik svarbi tekstinė informacija – konkretus straipsnio ar puslapio turinys. Toliau šis turinys apdorojamas atliekant duomenų valymo (angl. *data cleaning*) operacijas pagal poreikį. Taip paruošti duomenys suteikia išsamią ir tikslią informaciją apie kliento reputaciją internete.

Paruoštus internetinius duomenis apie kiekvieną naują ar esamą klientą galima perduoti natūralios kalbos apdorojimui (angl. *Natural Language Processing* – NLP) bei eksperimentiniais tikslais naudojamamas „*OpenAI ChatGPT*“ API funkcionalumui. „*ChatGPT*“ priima tuos pačius apdorotus tekstus kaip ir NLP modeliai, tačiau užklausa yra įkoduota taip, kad prašytų sentimentų analizės. Pavyzdžiui, į užklausos laukelį yra įrašomas tekstas: *“Please analyze the sentiment of this text and provide a score for positive, negative, and neutral. Provide scores strictly in this format: [neg:,neu:,pos:] (for example: [neg:-0.156,neu:0,pos:0.950]). Values should be strictly from -1 to 1.”.* Šis metodas leidžia gauti dirbtinio intelekto sugeneruotus sentimentinius įverčius kiekvienam tekstui. Likusiems NLP metodams, tokiems kaip „*BERT*“, „*RoBERTa*“ ir „*VADER*“, atskiros užklausos formuoti nereikia, nes jie analizuoja tekstą pagal savo vidines logikas, t.y. „*BERT*“ ir „*RoBERTa“* analizuoja žodžius dviprasmiškai, atsižvelgiant į jų kontekstą tiek prieš, tiek po nagrinėjamo žodžio. Jie yra pritaikyti suvokti gilų tekstų reikšmių ryšį ir įvertinti sentimentus remiantis viso teksto žodyno ir gramatikos kontekstu. Tuo tarpu „*VADER*“ metodas yra žodyno pagrindu sukurtas sentimentų analizės įrankis, skirtas paprastesnėms, ypač trumpoms,

tekstinėms žinutėms, tokioms kaip socialinių tinklų įrašai. „*VADER*“ analizuoja individualius žodžius ir jų poliarumą (teigiamą, neigiamą ar neutralią reikšmę) be detalaus konteksto analizės, todėl jis greitesnis, bet mažiau tikslus sudėtingesniuose tekstuose. Tokiu būdu sentimentų analizė gali būti atliekama tiek naudojant pažangesnius kontekstinius modelius („*BERT*“ / „*RoBERTa*“), tiek paprastesnius metodus („*VADER*“), atsižvelgiant į analizės tikslą ir teksto pobūdį.

Gautus sentimentinius įverčius sugrupuojame ir apskaičiuojame bendrą vidurkį, remdamiesi visais internete prieinamais šaltiniais. Pagal šį vidurkį kiekvienas klientas gali būti priskirtas skirtingam rizikos lygiui (žemas, vidutinis, aukštas). Be to, reguliariai kartojant šį procesą nustatytu dažnumu, galima stebėti, ar konkretaus kliento reputacija laikui bėgant keičiasi. Tokia sistema leidžia efektyviai vertinti klientų riziką bei identifikuoti potencialius pokyčius reputacijoje, kurie gali turėti įtakos verslo sprendimams.

Siekiant veiksmingai pritaikyti šį procesą konkretiems aviacijos pramonės poreikiams, esami NLP ir dirbtinio intelekto metodai yra perrenkami, išplečiant jų žodynus ir įtraukiant į juos aviacijai būdingą terminiją, sankcijų sąrašus ir bendrą aviacijos naujienų kontekstą. Šiuo tikslu duomenys gaunami iš pagrindinių aviacijos reguliavimo institucijų, tokių kaip EASA, ICAO ir FAA naujienų portalų, taip pat iš JT, ES ir OFAC tvarkomų sankcijų sąrašų, kuriais plačiai remiamasi ir kurių griežtai laikosi aviacijos organizacijos.

Kadangi „*OpenAI ChatGPT*“ funkcijos negalima tiesiogiai modifikuoti, siekiant gauti labiau aviacijos kontekstui pritaikytus nuotaikų balus, užklausa pakoreguota taip, kad pabrėžtų šį dėmesį. Pavyzdžiui, įvesties užklausa gali būti tokia: *"Please analyze the sentiment of this text and provide a score for positive, negative, and neutral. Rate this text sentiment in the context of the aviation sector and provide scores strictly in this format: [neg:,neu:,pos:] (for example: [neg:-0.156,neu:0,pos:0.950]). Values should be strictly from -1 to 1."*.

NLP metodų perkvalifikavimas yra daug išteklių reikalaujantis procesas, kuriam reikia didelės skaičiavimo galios. Čia neįkainojamais tampa lygiagretieji skaičiavimai, ypač naudojant CUDA (*Compute Unified Device Architecture*). CUDA pagreitina mokymo procesą, perkeldama skaičiavimus į GPU, todėl modelio mokymo laikas veiksmingai sutrumpėja daugiau nei perpus. CUDA funkcijų, prieinamų tokiose platformose kaip *„Google Colab Pro“*, integravimas padidina efektyvumą ir prieinamumą. Kadangi „*Google Colab*“ yra debesijos platforma, ji leidžia atlikti intensyvius skaičiavimus nenaudojant galingo vietinio kompiuterio, todėl yra puikus pasirinkimas atliekant daug išteklių reikalaujančias mokymo užduotis.

Galiausiai, atlikus sentimentinių įverčių perskaičiavimą ir sugrupavus juos iki kliento lygmens, sukuriamas naujas duomenų rinkinys, pagrįstas aviacijos sektoriaus specifika

pritaikytais ir apmokytais NLP modeliais. Šį rezultatą lyginame su pirminiais sentimentų analizės duomenimis, siekdami įvertinti nuoseklumą ir tikslumą. Papildomai, palyginus šiuos rezultatus su esamais, aviacijos įmonėje patvirtintais klientų rizikos lygiais, galima pasirinkti labiausiai priimtiną ir tiksliausią variantą, kuris užtikrina didesnį duomenų patikimumą ir analizės pagrįstumą.

2.2. Duomenų surinkimo metodai

Šio modelio kūrimui bei analizės procesams vykdyti būtina surinkti įvairių šaltinių duomenis, kurie būtų ne tik patikimi, bet ir aktualūs aviacijos sektoriui. Naudojami metodai užtikrina, kad surinkta informacija apimtų tiek bendrąją internetinę informaciją, tiek specifinius duomenis, susijusius su tarptautinėmis taisyklėmis ir reguliavimu. Toliau pateikiami pagrindiniai duomenų surinkimo šaltiniai ir jų svarba.

2.2.1. Google/BING API

„*Google*“ pasirinktinės paieškos JSON API ir „*Microsoft*“ , „*Bing*“ žiniatinklio paieškos API kūrėjams suteikia pažangias programinės prieigos prie paieškos rezultatų priemones, leidžiančias veiksmingai gauti ir analizuoti duomenis.

„*Google*“ pasirinktinės paieškos JSON API leidžia kūrėjams pateikti užklausas ir gauti paieškos rezultatus iš programuojamos paieškos sistemos (angl. *Programmable Search Engine - PSE*). Ši API palaiko pritaikymą, todėl naudotojai gali sutelkti dėmesį į konkrečias svetaines ar temas, apibrėždami pasirinktinius paieškos variklius. Atsakymai pateikiami JSON formatu, kurį galima lengvai integruoti į automatizuotas darbo eigas. Pavyzdžiui, paieškos užklausos gali būti pritaikytos konkrečiai aviacijos terminologijai arba reguliavimo svetainėms. Kaip pabrėžiama „Google“ API dokumentuose, „ši API leidžia programiškai gauti pasirinktinės paieškos ir vaizdų paieškos rezultatus, integruojant juos į savo programas, kad atitiktų konkrečius paieškos reikalavimus“ (Google APIs Explorer, 2024)

Microsoft „*Bing*“ žiniatinklio paieškos API teikia išsamų funkcijų rinkinį, įskaitant prieigą prie tinklalapių, vaizdų, vaizdo įrašų ir naujienų paieškos rezultatų. Dėl savo pažangių funkcijų, tokių kaip pritaikomi filtrai, „puslapiavimas“ ir užklausos terminų paryškinimas, ji yra universali priemonė žiniatinklio informacijai rinkti. Pasak „*Microsoft*“, API „leidžia siųsti paieškos užklausą į ‚Bing‘ ir gauti atitinkamus rezultatus įvairiais formatais, todėl idealiai tinka įterpti paieškos funkcijas į programas“ (What is the Bing Web Search API?, 2022)

Naudojantis šiomis API galima rinkti didelius kiekius duomenų, kartu išnaudojant unikalius kiekvienos platformos privalumus. Tačiau reikia atsižvelgti į tam tikrus apribojimus:

1. Paieškos rezultatų šališkumas: Kaip pabrėžiama Poudel ir Weninger (2024) tyrime, paieškos algoritmai dažnai teikia pirmenybę autoritetingiems šaltiniams, o prieštaringą ar politiškai jautrų turinį filtruoja. Jie pažymi, kad „SERP iš esmės teikia pirmenybę didelio autoriteto domenams, o tai gali apriboti užfiksuotų požiūrių įvairovę“.
2. Mastelio keitimas (angl. *scalability*) ir sąnaudų efektyvumas: Narayanan ir kt. taip pat pabrėžia, kad SERP API yra ekonomiškai efektyvios didelio masto duomenų rinkimui, palyginti su tiesiogine API prieiga prie socialinės žiniasklaidos platformų, ir teigiama: „Kadangi API apribojimai didėja pagrindinėse platformose, SERP panaudojimas yra prieinama duomenų gavimo alternatyva“ Poudel ir Weninger (2024)

„*Google*“ JSON API ir „*Microsoft*“ „*Bing*“ žiniatinklio paieška API išsiskiria kaip labai lanksčios priemonės, kurios supaprastina duomenų rinkimą ir analizę. Suteikdamos programinę prieigą prie paieškos rezultatų struktūrizuotu formatu , šios API supaprastina darbo eigą ir leidžia kūrėjams pritaikyti paieškas specifiniams poreikiams, pavyzdžiui, konkrečioms pramonės šakoms būdingoms temoms ar terminologijoms. Dėl jų pritaikomumo ir lengvo integravimo į automatizuotas sistemas jos yra nepakeičiamos norint efektyviai surinkti didelius kiekius svarbių duomenų. mažinant rankų darbo sąnaudas. Nepaisant tam tikrų apribojimų , pavyzdžiui, galimų paieškos sistemos šališkumų. rezultatai, šios API labai palengvina duomenų paieškos uždavinius ir palaiko keičiamo mastelio tikslines taikomąsias programas.

2.2.2. Sankcijų sąrašai

Sankcijos yra galinga užsienio politikos ir tarptautinio valdymo priemonė, kuria siekiama atgrasyti nuo žalingo elgesio, priversti laikytis tarptautinių normų arba bausti už pasaulinių standartų pažeidimus. Sankcijos taikomos individualiu, organizaciniu ir nacionaliniu lygmenimis, nukreiptos prieš tokią veiklą kaip terorizmo finansavimas, žmogaus teisių pažeidimai ir suvereniteto pažeidimai. Kaip aiškina Cortrightas ir Lopezas (2018), sankcijomis „siekiama atimti iš taikinių vertę ir priversti laikytis pasaulinių normų“. Nuo Šaltojo karo laikų jų taikymas labai išaugo, tapdamas svarbiausiu veiksniu palaikant tarptautinį saugumą ir valdymo struktūras.

Sankcijų tikslas - daryti įtaką politiniam ir ekonominiam elgesiui nesiimant karinio konflikto. Pasak Seyfi ir Hall (2020), sankcijos taikomos siekiant „kovoti su terorizmu, įtvirtinti taikos susitarimus, apriboti ginklų platinimą ir užtikrinti tarptautinės teisės laikymąsi“. Jos

taikomos taikant finansinius, prekybos, kelionių ar diplomatinius apribojimus, kuriais siekiama daryti spaudimą taikiniams, kartu kuo mažiau kenkiant civiliams gyventojams.

Pagrindiniai sankcijų sąrašai:

- OFAC (Užsienio turto kontrolės biuras) : OFAC (Užsienio prekybos tarnyba) (angl. *Office of Foreign Assets (OFAC)*), valdoma JAV Iždo departamento, administruoja ekonomines ir prekybos sankcijas, skirtas šalims, subjektams ir asmenims, susijusiems su terorizmu, narkotikų prekyba ir kita neteisėta veikla. Jos „Specialiai įvardytų piliečių ir blokuojamų asmenų sąraše“ (SDN) yra daugiau kaip 6 000 įrašų, kurie rodo JAV sankcijų vykdymo apimtis ir specifiškumą (U.S. Department of the Treasury, 2024). Griežtas OFAC vykdymas pasitelkia pasaulinę priklausomybę nuo JAV dolerio, užtikrindamas, kad reikalavimų laikytųsi net trečiosios šalys (Seyfi & Hall, 2020).
- JT sankcijos: Šias sankcijas administruoja Jungtinių Tautų Saugumo Taryba (JT ST), jos yra visuotinai privalomos valstybėms narėms. Jomis daugiausia dėmesio skiriama tokiems klausimams kaip terorizmas, branduolinių ginklų platinimas ir regioniniai konfliktai. Biersteker ir Portela (2015) pažymi, kad JT sankcijos yra unikalios savo teisėtumu, kuris kyla iš visuotinio sutarimo pagal JT Chartiją.
- ES sankcijos: Europos Sąjunga taiko sankcijas savarankiškai arba derindama jas su JT priemonėmis, spręsdama žmogaus teisių pažeidimų, geopolitinių konfliktų ir terorizmo problemas. ES sankcijos dažnai naudojamos kaip papildomos priemonės JT veiksmams, sustiprinant jų taikymo sritį ir vykdymą. Biersteker ir Portela (2015) pabrėžia ES vaidmenį plečiant sankcijas, kad jos apimtų regioninius interesus, kaip matyti iš priemonių prieš Siriją ir Rusiją.

OFAC, JT ir ES sankcijų sąrašai pasirinkti dėl jų visuotinio taikymo ir patikimumo. JT pasauliniai įgaliojimai užtikrina visuotinį teisėtumą, o OFAC taikomi griežti vykdymo užtikrinimo mechanizmai užtikrina maksimalų reikalavimų laikymąsi. ES sankcijos turi regioninį aspektą, nes jomis sprendžiamos konkrečios geopolitinės problemos. Kartu šie sąrašai sudaro išsamią tarptautinių normų stebėsenos ir vykdymo užtikrinimo sistemą, todėl jie nepakeičiami tokiose pramonės šakose kaip aviacija, kur sankcijų laikymasis yra labai svarbus pasaulinei veiklai.

2.2.3. Aviacijos institucijų duomenys

Aviacijos institucijoms tenka pagrindinis vaidmuo užtikrinant pasaulinių kelionių oro transportu saugą, veiksmingumą ir standartizavimą. Pagrindiniai jų tikslai - reguliavimas ir

priežiūra, sertifikavimas, standartizavimas ir saugos skatinimas. Toliau pateikiamas pagrindinių aviacijos institucijų ir jų funkcijų paaiškinimas:

1. Europos Sąjungos aviacijos saugos agentūra (EASA) yra atsakinga už civilinės aviacijos saugos ir aplinkos apsaugos užtikrinimą visoje Europoje. Jos funkcijos apima aviacijos saugos taisyklių derinimą, orlaivių ir aviacijos organizacijų sertifikavimą ir bendrosios ES aviacijos rinkos plėtros skatinimą. Kaip pažymėjo Europos Komisija, EASA „užtikrina vienodai aukštą saugos lygį visoje Europoje ir remia aplinkos apsaugą bei ekonominį efektyvumą aviacijoje“ (Europos Sąjunga, 2024). Be to, EASA aktyviai bendradarbiauja su suinteresuotosiomis šalimis, siekdama skatinti saugos kultūrą ir skleisti geriausią patirtį (EASA, 2024).
2. Federalinė aviacijos administracija (FAA) prižiūri visus civilinės aviacijos aspektus Jungtinėse Valstijose, daugiausia dėmesio skirdama saugai ir teisės aktų laikymuisi. Ji teikia išsamią rekomendacinę medžiagą, pavyzdžiui, patariamuosius aplinkraščius, kurie padeda aviacijos suinteresuotiesiems subjektams laikytis taisyklių. FAA pabrėžia savo kaip „nacionalinės aviacijos institucijos, kuriai pavesta užtikrinti civilinės aviacijos saugą, saugumą ir veiksmingumą“ (FAA, 2024), vaidmenį. Šia medžiaga siekiama remti geriausią praktiką ir naujoves aviacijos bendruomenėje.
3. Tarptautinė civilinės aviacijos organizacija (ICAO) yra specializuota Jungtinių Tautų agentūra, vienijanti 193 šalis nares, kad skatintų saugią ir tvarkingą pasaulinę aviaciją. Ji nustato tarptautinius standartus ir taisykles, reglamentuojančias aviacijos saugą, navigaciją ir aplinkos apsaugą. Pasak ICAO, jos tikslas - „padėti valstybėms dalytis pasaulio dangumi siekiant abipusės naudos“, suteikiant pagrindą bendradarbiavimui civilinės aviacijos srityje (ICAO, 2024).

Šios institucijos bendradarbiauja siekdamos palaikyti ir didinti pasaulinės aviacijos saugą ir veiksmingumą, reguliuodamos, sertifikuodamos ir skatindamos pasaulinę saugos kultūrą. Jų suderintomis pastangomis užtikrinama, kad tarptautinis oro susisiekimas atitiktų vienodus standartus, taip skatinant pasitikėjimą ir patikimumą visoje aviacijos pramonėje.

Darbe naudojami šių institucijų naujienų srautai siekiant surinkti kontekstinę informaciją apie aviacijos sektorių. Ši informacija yra itin svarbi, kad NLP metodai galėtų būti pritaikyti ir įgyti specifinių žinių, reikalingų aviacijos sektoriaus terminologijai, aktualijoms ir reguliavimo aspektams analizuoti. Tokiu būdu modeliai tampa jautresni sektoriaus specifikai, užtikrinant tikslumą ir kontekstualumą apdorojant su aviacijos sektoriumi susijusius duomenis.

2.3. Pararеliniai skaičiavimai (CUDA)

Lygiagretieji skaičiavimai - tai kelių skaičiavimų vykdymas vienu metu, dažnai pasiekiamas didesnę problemą padalijus į mažesnes, nepriklausomas užduotis, kurias galima apdoroti vienu metu. Šis metodas gerokai padidina skaičiavimo efektyvumą ir yra plačiai taikomas mokslinio modeliavimo, didelių duomenų analizės ir realaus laiko taikomosiose programose. Pasak Ciccozzi ir kitų (2022), „lygiagretieji skaičiavimai tapo dominuojančiu metodu, leidžiančiu panaudoti masinę lygiagrečiąją aparatinę įrangą, padidinti skaičiavimo galią ir sumažinti energijos sąnaudas“.

CUDA (angl. *Compute Unified Device Architecture*) - tai lygiagrečiųjų skaičiavimų platforma ir taikomųjų programų programavimo sąsaja (API), kurią sukūrė NVIDIA, siekdama panaudoti GPU (grafikos procesorių) bendrosios paskirties skaičiavimams. Ji leidžia kūrėjams rašyti labai lygiagretines programas, naudojant C, C++ ir Fortran kalbas, ir taip gerokai pagreitinti duomenų apdorojimo užduotis. Kaip pažymi Skirnevsky ir kiti (2017), „CUDA technologija leidžia sparčiau atlikti duomenų operacijas paskirstant užduotis keliems GPU branduoliams, todėl idealiai tinka dideliems duomenų rinkiniams ir intensyvių skaičiavimų algoritmams tvarkyti“ (Skirnevskiy, Pustovit, & Abdrashitova, 2017)

CUDA ypač veiksminga taikant programas, kurioms reikia didelio lygiagretumo, pavyzdžiui, apdorojant vaizdus, mokantis mašinomis ir atliekant duomenų analizę. Ji palengvina:

1. Duomenų perdavimą: Duomenų perkėlimą iš pagrindinės atminties į GPU atmintį apdorojimui.
2. Srautai: Naudoti tūkstančius GPU branduolių lygiagrečiam užduočių vykdymui.
3. Optimizavimas: Skaičiavimo laiko mažinimas, vienu metu atliekant operacijas su keliais duomenų taškais.

Pavyzdžiui, CUDA grindžiami lygiagretieji skaičiavimai buvo naudojami skaitmeninių vaizdų apdorojimui optimizuojant triukšmo šalinimo algoritmus. Skirnevsky ir kiti (2017) aprašo, kaip „naudojant CUDA technologiją triukšmo šalinimo procese galima vienu metu apdoroti pikselius, gerokai pagreitinant algoritmą ir sumažinant bendrą skaičiavimo laiką“ (Skirnevsky ir kt., 2017). Panašiai Hentosh ir kiti (2023) iliustruoja, kaip „naudojant CUDA realizuoti lygiagretieji algoritmai pasiekė daugiau nei šešiskart didesnį pagreitį mokant Random Forest modelius dideliuose duomenų rinkiniuose“ (Hentosh ir kt., 2023).

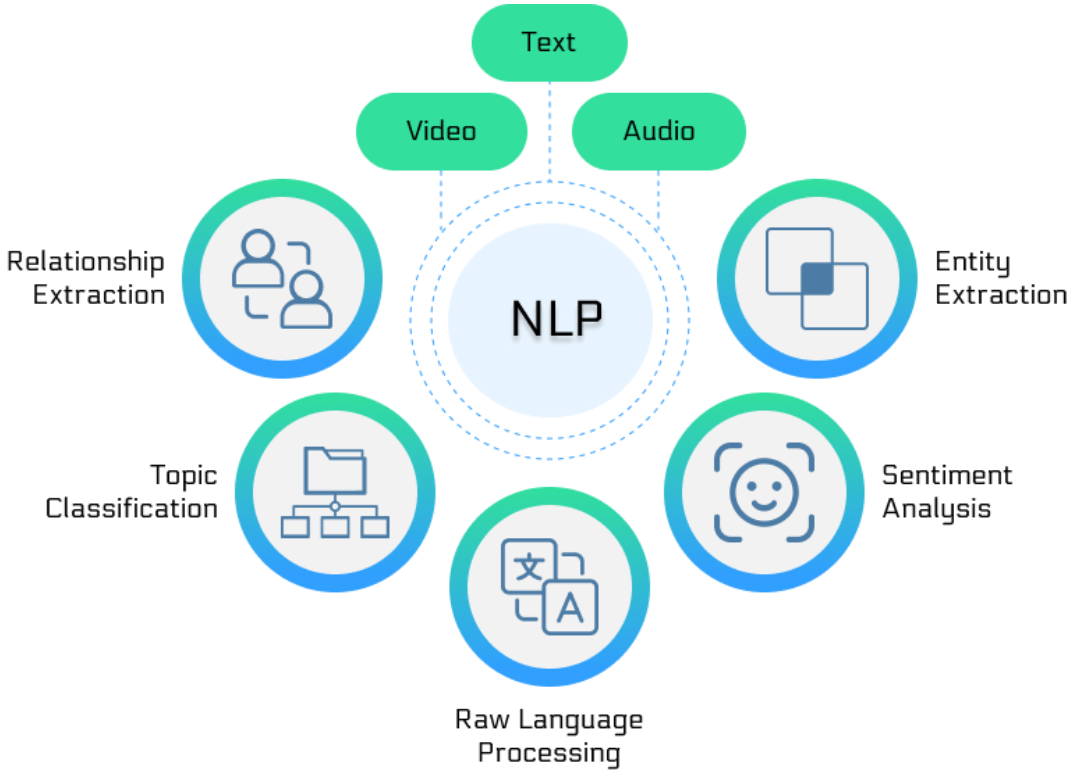
CUDA gebėjimas labai efektyviai atlikti lygiagrečius skaičiavimus paverčia ją svarbia priemone srityse, kuriose reikalingas didelės apimties duomenų apdorojimas arba realaus laiko analizė, transformuojant pramonės šakas, priklausomas nuo skaičiavimo našumo.

2.4. Sentimentinė analizė

Sentimentinė/nuotaikų analizė yra pagrindinė natūralios kalbos apdorojimo (NLP) užduotis, kuria siekiama suprasti teksto emocinį toną - teigiamą, neigiamą ar neutralų. Įvairūs metodai, pradedant leksikonu pagrįstais metodais ir baigiant mašininio mokymosi modeliais, siūlo skirtingus būdus nuotaikų analizei atlikti. Šiame skyriuje aptariamos trys dažniausiai naudojamos priemonės: VADER, BERT ir RoBERTa.

2.4.1. Natūralios kalbos apdorojimas (NLP)

Natūralios kalbos apdorojimas (angl. *Natural Language Processing - NLP*) yra dirbtinio intelekto sritis, kurios tikslas – suteikti kompiuteriams gebėjimą analizuoti, suprasti ir generuoti natūraliai vartojamą žmonių kalbą. NLP derina kompiuterių mokslo, lingvistikos ir mašininio mokymosi principus, kad padėtų spręsti įvairias kalbos apdorojimo užduotis, tokias kaip sentimentų analizė, teksto klasifikacija, automatinis vertimas ir kalbos atpažinimas (7 pav.).



Šaltinis: (What is NLP and how it is implemented in our lives, 2024)

7 pav. Pagrindiniai natūralios kalbos apdorojimo (NLP) komponentai

NLP procesas paprastai apima kelis svarbius etapus:

1. Pirminis apdorojimas (angl. *Preprocessing*) - Šis etapas apima tekstinių duomenų paruošimą tolimesnei analizei:

a. Tokenizacija: Tekstas suskaidomas į mažesnius vienetus, vadinamus tokenais (pvz., žodžius arba sakinius). Šis žingsnis padeda kompiuteriui apdoroti tekstą struktūruotai.

b. Stopžodžių šalinimas: Dažnai pasikartojantys, mažai informacijos turintys žodžiai (pvz., „ir“, „bet“, „tai“) pašalinami siekiant sumažinti nereikalingą triukšmą analizėje.

c. Steminimas ir lematizacija: Žodžių grąžinimas į jų bazines arba šaknines formas, kad sumažėtų žodyno įvairovė ir būtų lengviau analizuoti prasmę (Nadkarni, Ohno-Machado, & Chapman, 2011)

2. Sintaksinė analizė - Šiame etape siekiama suprasti žodžių tarpusavio ryšius. Tai apima sakinių struktūros kūrimą, dalų kalbos atpažinimą (angl. *POS tagging*) ir sakinio gramatinių ryšių nustatymą. Ši analizė padeda sukurti hierarchiją, leidžiančią tiksliau suprasti sakinį.

3. Semantinė analizė - Semantinės analizės tikslas yra suprasti žodžių prasmę kontekste. Naudojant metodus, pagrįstus giliaisiais neuroniniais tinklais, NLP modeliai analizuoja kontekstą tiek į kairę, tiek į dešinę nuo nagrinėjamo žodžio. Tai ypač svarbu, nes tas pats žodis gali turėti skirtingą reikšmę skirtinguose kontekstuose (Min et al., 2023). Pavyzdžiui, žodis „*bankas*“ (angl. “*bank*”) gali reikšti finansinę instituciją arba upės krantą, priklausomai nuo sakinio.

4. Mašininis mokymasis ir neuroniniai tinklai - NLP modeliai, tokie kaip BERT ir RoBERTa, naudoja pažangius giliojo mokymosi algoritmus, kurie apdoroja tekstą lygiagrečiai ir adaptuojasi skirtingiems duomenų rinkiniams. Šie modeliai yra apmokomi iš didžiulių duomenų rinkinių, naudojant savarankiško mokymosi metodus, leidžiančius modeliams išmokti žodžių reikšmes, jų kontekstą ir tarpusavio ryšius (Koroteev, 2021)

Pasak Devlin ir kt., (2019) NLP įgyvendinimas dažnai apima iš anksto apmokytų modelių pritaikymą konkrečiam kontekstui. Pavyzdžiui:

- Duomenų paruošimas: Žodžių vektorizacija naudojant metodus, tokius kaip „*Word2Vec*“ arba „TF-IDF“, siekiant paversti tekstą į skaitinę formą, kurią gali suprasti mašininiai modeliai.
- Modelio pritaikymas: Naudojant iš anksto apmokytus modelius (pvz., BERT), jie pritaikomi naujam tekstui per papildomą apmokymą specifiniuose duomenų rinkiniuose. Šis procesas vadinamas „*fine-tuning*“.
- Vertinimas: Po apmokymo modeliai vertinami remiantis tikslumo, prisiminimo ir F1 matu, siekiant užtikrinti, kad rezultatai yra patikimi ir atitinka užduoties tikslus.

Pasak Min et al. (2023), NLP technologijos „suteikia galimybę apdoroti didžiulius teksto duomenų kiekius efektyviai ir tiksliai, pritaikant automatizavimo metodus įvairiems taikymo atvejams“. Tokie metodai ne tik sumažina darbo sąnaudas, bet ir užtikrina didesnį tikslumą, kai dirbama su struktūrizuotu ir nestruktūrizuotu tekstu.

NLP revoliucija, paremta giliai integruotais neuroniniais tinklais, suteikia galimybes spręsti sudėtingas kalbos apdorojimo užduotis, kurios anksčiau reikalavo intensyvaus rankinio darbo ar net buvo neįmanomos atlikti.

2.4.1.1. VADER

VADER (angl. *Valence Aware Dictionary and sEntiment Reasoner*) yra taisyklėmis pagrįstas leksikono metodas, skirtas nuotaikų analizei, ypač pritaikytas socialinei žiniasklaidai ir trumpiems tekstams. Jis priskiria tekstui sentimentų balus (teigiamus, neigiamus ir neutralius), remdamasis iš anksto nustatytu sentimentų leksikonu. Kiekvienam leksikono žodžiui priskiriamas atitinkamas balas, o bendra nuotaika apskaičiuojama sumuojant šiuos balus. Pasak Hutto (2014), VADER yra veiksmingas nustatant trumpo, glausto teksto, pavyzdžiui, twitter‘io žinutės, nuotaikas, nes remiasi euristika ir kontekstiniais modifikatoriais, tokiais kaip neiginiai (angl. *negations*) ar stiprintuvai (angl. *amplifiers*).

Nors VADER yra paprastas ir skaičiavimo požiūriu efektyvus, jis apsiriboja tik savo leksikone esančiais žodžiais ir yra mažiau pajėgus suprasti niuansuotus ar kontekstinius nuotaikų pokyčius tekste.

2.4.1.2. BERT

2019 m. Devlin ir kt. sukurtas BERT yra paradigminis NLP pokytis, nes pristatoma giliuoju mokymusi pagrįsta transformatoriaus architektūra, galinti atlikti dvikryptę teksto analizę. Tai leidžia BERT suprasti žodžio kontekstą remiantis tiek ankstesniais, tiek vėlesniais žodžiais sakinyje. Atliekant nuotaikų analizę, BERT puikiai supranta sudėtingus kontekstus, idiomatinius išsireiškimus ir specifinės srities žodyną.

Pavyzdžiui, atlikdamas teksto klasifikavimo užduotis, BERT tiksliai pritaiko savo iš anksto apmokytą modelį konkrečiame duomenų rinkinyje, kad jis būtų pritaikytas prie nuotaikų analizės srities. Pasak Koroteev (2021), BERT gerokai pranoksta tradicinius metodus, kai sprendžiami uždaviniai, reikalaujantys gilaus kalbos niuansų supratimo.

2.4.1.3. RoBERTa

2019 m. Liu ir kt. sukurta RoBERTa - tai BERT patobulinimas, kuriame daugiausia dėmesio skiriama mokymo efektyvumui ir optimizavimui. Pašalinus tam tikrus apribojimus, pavyzdžiui, kito sakinio prognozavimą, ir padidinus modelio mokymo duomenis, „RoBERTa“ pasiekia dar didesnį sentimentų analizės ir kitų NLP užduočių tikslumą. (Tarunesh, Aditya, & Choudhury, 2021)

Vienas iš pagrindinių skirtumų - mokymo režimas; „RoBERTa“ naudoja išsamesnį išankstinį mokymą ir geresnį hiperparametrų derinimą. Pasak Müller ir kitų (2023), „RoBERTa“ yra ypač veiksminga, kai reikia tiksliai sureguliuoti konkrečios srities sentimentų uždavinius, pavyzdžiui, analizuojant klientų atsiliepimus arba viešąsias nuomones socialinėse platformose.

2.4.1.4. OpenAI Chat-GPT API

„OpenAI“ sukurta ‚ChatGPT API‘ yra patikima priemonė dirbtiniu intelektu pagrįstiems pokalbių agentams integruoti į programas. „ChatGPT“, sukurta remiantis GPT-3.5 ir GPT-4 architektūromis, naudoja gilaus mokymosi ir natūralios kalbos apdorojimo (NLP) metodus, kad suprastų ir generuotų į žmogų panašų tekstą. Šis gebėjimas leidžia jį įgyvendinti įvairiose NLP užduotyse, įskaitant nuotaikų analizę, turinio generavimą ir bendravimo su klientais sistemas (Fitria, 2023; Perlman, 2022).

Pagrindinė „ChatGPT“ funkcija - transformatoriais pagrįstas (angl. *transformer - based*) modelis, apmokytas naudojant didelius duomenų rinkinius, kuriuos sudaro knygos, straipsniai ir kiti teksto šaltiniai. API veikia priimdama tekstinę įvestį (užklausą arba raginimą) ir generuodama nuoseklų atsakymą pagal mokymo metu išmoktus modelius. Šis procesas apima keletą pagrindinių etapų:

1. Ženklinimas (angl. *Tokenization*): Įvesties tekstas suskaidomas į mažesnius vienetus, vadinamus žetonais, kad modelis galėtų veiksmingai apdoroti informaciją.
2. Konteksto supratimas: Transformatoriaus modelis įvertina įvesties tekste esančių ženklų ryšius. Jis naudoja dėmesio mechanizmus, kad nustatytų, kurios įvesties teksto dalys yra svarbiausios užklausai.
3. Atsakymo generavimas: Remdamasis konteksto supratimu, modelis numato kitą ženklų seką atsakymui generuoti. Šiam prognozavimo procesui daro įtaką mokymo duomenys ir tikslinimas naudojant mokymąsi pastiprinant su žmogaus grįžtamuoju ryšiu (angl. *reinforcement learning with human feedback, RLHF*) (Fitria, 2023).

OpenAI užtikrino, kad API būtų patogi ir universali. Kūrėjai gali lengvai siųsti tekstines užklausas į API galinį tašką ir gauti atsakymą realiuoju laiku. API lankstumas leidžia pritaikyti,

pavyzdžiui, nustatyti atsakymų toną ar sudėtingumą, o tai ypač naudinga konkrečioms sritims skirtose taikomosiose programose, pavyzdžiui, švietimo ar klientų aptarnavimo srityse (Haque et al., 2022).

Įgyvendinimas NLP užduotyse pagal Perlman (2022):

1. Sentimentų analizė: API gali analizuoti emocinį teksto toną interpretuodama kontekstines užuominas. Pavyzdžiui, kūrėjas gali užprogramuoti API klasifikuoti tekstą kaip teigiamą, neigiamą ar neutralų pagal iš anksto nustatytus rodiklius.
2. Teksto apibendrinimas: ChatGPT gali sutraukti ilgus dokumentus į glaustas santraukas, išryškindama pagrindinius dalykus ir išlaikydama nuoseklumą.
3. Pokalbių dirbtinis intelektas: jis įgalina pokalbių robotus, galinčius imituoti į žmogų panašų dialogą, kurie gali būti diegiami klientų aptarnavimo, švietimo ir kitose interaktyviosiose programose.

ChatGPT API supaprastina sudėtingų NLP galimybių integravimą į taikomąsias programas. Ji yra labai keičiamo dydžio, prieinama per debesijos infrastruktūrą ir palaiko įvairias kalbas, o tai didina jos pritaikomumą visame pasaulyje. Tačiau kūrėjai turi atsižvelgti į etines problemas, pavyzdžiui, galimą generuojamų atsakymų šališkumą, ir įgyvendinti apsaugos priemones, kad užtikrintų tikslius ir tinkamus rezultatus (Fitria, 2023; Haque et al., 2022).

Naudodamiesi ChatGPT API, kūrėjai gali kurti išmaniąsias sistemas, kurios pagerina naudotojų patirtį, automatizuoja pasikartojančias užduotis ir teikia prasmingas įžvalgas NLP grindžiamose programose. Kaip pažymi Fitria (2023), „ChatGPT gebėjimas generuoti į žmogų panašius atsakymus mažina atotrūkį tarp mašinų efektyvumo ir žmonių sąveikos, atverdamas kelią novatoriškiems dirbtinio intelekto sprendimams“.

2.5. Rezultatų validacija ir palyginimas

K-vidurkių klasterizavimas yra plačiai pripažintas nekontroliuojamas mašininio mokymosi algoritmas, naudojamas duomenų rinkiniams suskirstyti į k klasterių pagal jiems būdingus modelius. Algoritmas optimizuoja klasterio vidinį panašumą, kartu mažindamas klasterio tarpusavio panašumą, naudodamas pasirinktą atstumo metriką, pavyzdžiui, Euklido atstumą. Kaip pabrėžia Ahmed et al. (2020), K-vidurkių algoritmas dėl savo paprastumo ir skaičiavimo efektyvumo dažnai taikomas scenarijuose, kuriuose reikia efektyvaus ir keičiamo mastelio klasterizavimo.

Šiame tyrime K-vidurkių klasterizavimas taikomas nuotaikų balams, gautiems iš natūralios kalbos apdorojimo (NLP) analizės, klasifikuoti į tris rizikos lygius: „Didelis“, ‚Vidutinis‘ ir ‚Mažas‘. Šis klasifikavimo procesas padeda kiekvienam analizuojamam subjektui priskirti kategorinę rizikos etiketę, o tai leidžia atlikti struktūrizuotą rizikos vertinimą pagal KYC (angl. *Know Your Customer*) ir nuolatinio deramo patikrinimo procesus.

Klasterizavimo procesas pradedamas nustatant optimalų klasterių skaičių (k), kuris šiuo atveju iš anksto nustatytas kaip trys, atitinkantys tikslinį rizikos lygį. Po inicializacijos algoritmas iteratyviai atnaujina klasterių centroidus, priskirdamas kiekvieną nuotaikos balą artimiausiam klasteriui ir perskaičiuodamas centroidus, kol pasiekiama konvergencija. Šis procesas užtikrina, kad duomenų taškai (nuotaikų balai) būtų sugrupuoti pagal jų santykinį panašumą.

Kaip pabrėžia Ikotun et al. (2023), vienas iš K-vidurkių iššūkių yra jo priklausomybė nuo atsitiktinio inicijavimo, dėl kurio gali būti gaunami neoptimalūs klasterizavimo rezultatai arba konvergavimas į vietinius minimumus. Šiame tyrime tokie apribojimai sprendžiami taikant kelis inicijavimus ir pasirenkant mažiausią vidinę klasterio dispersiją turintį sprendimą.

Be to, K-vidurkių efektyvumas šiame kontekste priklauso nuo išankstinio apdorojimo etapų, įskaitant nuotaikų balų normalizavimą ir nukrypimų pašalinimą, nes šie etapai padidina klasterizavimo patikimumą ir tikslumą. Šis procesas ypač svarbus atsižvelgiant į tai, kad analizuojami duomenys yra didelės dimensijos, kaip siūlo Ahmedas ir kiti (2020), kurie rekomenduoja taikyti tokias strategijas kaip centroidų inicializacija ir išankstinis duomenų paruošimas, kad pagerėtų klasterizavimo rezultatai.

Baigus klasterizavimą, rezultatai patvirtinami lyginant priskirtus rizikos lygius su ekspertų priskirtomis etiketėmis arba etalonais, jei tokių yra. Klasterizavimo kokybei įvertinti naudojamos tokios metrikos kaip silueto balas ir Davieso-Bouldino indeksas, užtikrinant, kad gautos klasifikacijos atitiktų tikėtinus duomenų rinkinio modelius ir ryšius.

Integruojant šias įžvalgas į KYC ir išsamaus patikrinimo darbo eigą, tyrime parodoma, kaip K-vidurkių klasterizacija gali veiksmingai padėti atlikti dinamišką ir adaptyvų rizikos vertinimą, dar labiau įtvirtinant jos, kaip pagrindinio duomenų pagrindu priimamų sprendimų komponento, vaidmenį.

3. EKSPERIMENTINIS SKYRIUS

3.1. Darbo aplinkos paruošimas

Aplinka paruošta naudojant „Google Colab“, nemokamą debesų kompiuterijos platformą, leidžiančią naudotojams rašyti ir vykdyti „Python“ kodą „Jupyter“ sąsiuvinio (angl. *notebook*) sąsajoje. Google Colab pasirinkta dėl lankstumo, prieinamumo ir integruoto GPU ir TPU palaikymo, kuris gerokai pagreitina skaičiavimo užduotis, ypač mašininio mokymosi ir gilaus mokymosi projektuose (Google Research, 2024).

Siekiant užtikrinti, kad aplinka būtų paruošta efektyviam duomenų apdorojimui ir modeliavimui, būtinos „Python“ bibliotekos įdiegiamos naudojant „*pip*“ funkcionalumą. Šios bibliotekos užtikrina įvairių projekto aspektų funkcionalumą:

- Duomenų tvarkymas: Siekiant palengvinti veiksmingą duomenų tvarkymą ir analizę, įdiegiamos tokios bibliotekos kaip „pandas“ ir „numpy“.
- NLP įrankiai: Tokie paketai kaip „BeautifulSoup“, „Transformers“, „nltk“ ir „spaCy“ įdiegiami teksto apdorojimo ir natūralios kalbos apdorojimo užduotims atlikti.
- Gilaus mokymosi sistemos: „TensorFlow“ ir „PyTorch“ įdiegtos tam, kad būtų galima mokyti ir vertinti gilaus mokymosi modelius.
- Vizualizavimo ir statistikos įrankiai: Įdiegtos tokios bibliotekos kaip „matplotlib“ ir „seaborn“, kad būtų galima kurti įžvalgias duomenų vizualizacijas ir atlikti statistinę analizę.
- Tinklapių nuskaitymas ir užklausos: Įdiegti tokie įrankiai kaip „BeautifulSoup“, „requests“ ir „selenium“, kad būtų galima efektyviai nuskaityti žiniatinklį ir gauti duomenis iš žiniatinklio šaltinių.

Įdiegus šias bibliotekas užtikrinama, kad aplinkoje būtų įdiegtos visos NLP darbo eigai, įskaitant išankstinį duomenų apdorojimą, modelių kūrimą, vertinimą ir vizualizavimą, reikalingos priemonės.

Bibliotekų importavimas: Įdiegus reikiamas bibliotekas, jos importuojamos į projektą. Pagrindinės importuojamos bibliotekos: „transformers“ darbui su NLP modeliais, „BeautifulSoup“ žiniatinklio nuskaitymui ir „scipy“ pažangioms matematinėms operacijoms. Šis importas užtikrina, kad projektas turėtų prieigą prie visų esminių funkcijų kiekviename darbo eigos etape.

Integracija su „Google Drive“: Aplinka integruota su „Google Drive“ naudojant drive.mount(). Tai suteikia sklandžią prieigą prie debesyje saugomų duomenų rinkinių ir užtikrina, kad rezultatus būtų galima įrašyti tiesiai į diską, kad jie būtų nuolat pasiekiami.

GPU/TPU naudojimas: Naudojant „*Google Colab*“ GPU ir TPU palaikomąsias funkcijas, atliekamos skaičiavimams imlios užduotys, pavyzdžiui, didelių NLP modelių mokymas. Ši funkcija leidžia greičiau ir efektyviau vykdyti užduotis nereikalaujant aukštos klasės vietinės aparatinės įrangos.

Būtent dėl GPU/TPU funkcionalumo projekto vykdymui pirmenybė buvo teikta „Google Colab“ platformai, kurios privalumai dokumentacijoje yra apibrėžiami taip (Google Research, 2024).:

- Ekonomiškai efektyvi: Ji yra nemokama ir apima prieigą prie galingų GPU ir TPU.
- Debesų pagrindu: Naudojant debesų saugyklą galima lengvai pasiekti failus arba jais dalytis.
- Iš anksto įdiegtos bibliotekos: Daugelis įprastų bibliotekų yra iš anksto įdiegtos, todėl supaprastėja sąranka.
- Bendradarbiavimo funkcijos: Keli naudotojai gali dirbti su tuo pačiu užrašų knygeliu realiuoju laiku, taip padidindami produktyvumą

Tokia sąranka užtikrina paruoštą naudoti, galingą NLP ir mašininio mokymosi darbo eigos aplinką, panaudojant debesijos išteklių skaičiavimo galią ir sumažinant sąrankos sudėtingumą.

3.2. Bazinių duomenų surinkimas ir apdorojimas

Bazinių duomenų rinkimas ir apdorojimas yra labai svarbus bet kurio duomenų modeliavimo projekto etapas. Duomenų kokybės ir nuoseklumo užtikrinimas yra esminis dalykas kuriant patikimus modelius ir generuojant prasmingas įžvalgas. Šiame skyriuje aprašomas duomenų rengimo procesas, kuris apima pirminių duomenų valymą, standartizavimą ir struktūrizavimą, kad būtų užtikrintas jų tikslumas ir vientisumas tolesnei analizei. Duomenų paruošimas sušvelnina klaidas, sumažina šališkumą ir pagerina bendrą analitinių ir mašininio mokymosi modelių veikimą.

Pirminių duomenų failas, pavadintas „*MTD_Companies.csv*“, yra iš esamos aviacijos bendrovės KYC duomenų bazės paimtas atsitiktinių duomenų rinkinys, kuriame egzistuoja 28 žemos rizikos įvertinimą turinčios įmonės, 53 vidutinį rizikos vertinimą turinčios bei 9 aukštu rizikos lygiu įvertintos įmonės. Pirmiausia duomenų failas įkeliamas į „Pandas“ duomenų struktūrą (angl. *dataframe*) iš „Google Drive“ debesijos platformoje esančios saugyklos. Šioje

integracijoje naudojama „*Google Colab*“ platformoje įdiegta duomenų saugojimo funkcija. Griežtai taikant atitinkamus duomenų tipus ir atliekant tikslinius valymo veiksmus, šiuo procesu užtikrinama, kad baziniame duomenų rinkinyje nebūtų anomalijų ir nenuoseklumų, taip padedant tvirtą pagrindą KYC sprendimo kūrimui.

Duomenų rinkinyje prieinami keturi stulpeliai:

- id - unikalus klientų kompanijos identifikacinis numeris
- legal_name - klientų kompanijos pilnas pavadinimas
- risk_level - kiekvienai klientų kompanijai priskirtas rizikos lygis pagal vidinius įmonės klientų rizikos vertinimo procesus
- updated_at - pateiktos informacijos atnaujinimo data

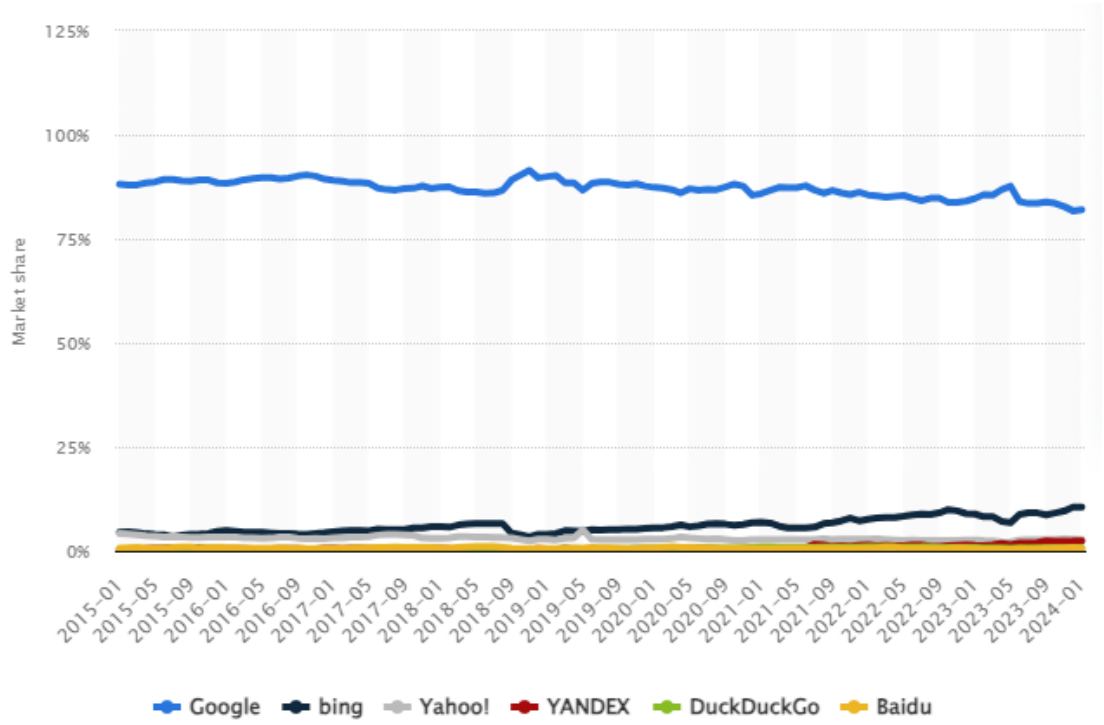
Siekiant užtikrinti nuoseklumą ir išvengti klaidų atliekant analizę, visiems laukams priskiriami duomenų tipai. Pavyzdžiui, laukas „*id*“, kuriame pateikiami unikalūs kiekvieno įrašo identifikatoriai, buvo pakeistas į skaitmeninį duomenų tipą, kad būtų išvengta galimų neatitikimų dėl neskaitmeninių arba netinkamai suformatuotų verčių. Šis konvertavimas apėmė negaliojančių įrašų vertimą į „*NaN*“ ir vėlesnį jų pašalinimą, kad būtų išsaugotas duomenų vientisumas. Panašiai laukas „*updated_at*“, kuriame saugomi laikiniai duomenys, buvo konvertuotas į standartinį datos ir laiko formatą. Šis konvertavimas leidžia atlikti tikslias laiko operacijas ir palyginimus, pavyzdžiui, stebėti atnaujinimų aktualumą.

Siekiant užtikrinti vienodumą ir išvengti analitinių neatitikimų, laukas „*risk_level*“, kuris yra esminis kategorinis kintamasis, buvo apribotas iš anksto nustatytomis reikšmėmis („*Low*“, „*Medium*“, „*High*“). Visi negaliojantys ar nenuoseklūs šio lauko įrašai buvo atmesti. Be to, siekiant suvienodinti lauką „*legal_name*“, jis buvo pertvarkytas taip, kad būtų pašalintos pradiniai ir galiniai „baltieji tarpai“ (angl. *whitespaces*) ir užtikrintas nuoseklus rašymas didžiosiomis raidėmis. Ši transformacija ypač svarbi subjektų pavadinimams, nes dėl formatavimo skirtumų, pavyzdžiui, nenuoseklaus didžiųjų ir mažųjų raidžių naudojimo, analizės metu gali atsirasti pasikartojančių arba neteisingai suklasifikuotų įrašų. Standartizavus šį lauką užtikrinama, kad pavadinimai būtų palyginami ir juose nebūtų formatavimo klaidų, o tai padidintų duomenų rinkinio aiškumą ir patogumą naudoti.

Griežtai taikant tinkamus duomenų tipus ir atliekant tikslinius valymo veiksmus, šiuo procesu užtikrinama, kad baziniame duomenų rinkinyje nebūtų anomalijų ir nenuoseklumų, ir taip sukuriamas tvirtas pagrindas KYC sprendimo kūrimui.

3.3. Žiniatinklio duomenų nuskaitymas (web-scraping)

Šiame skyriuje aprašomas interneto duomenų nuskaitymo procesas, daugiausia dėmesio skiriant informacijos paieškai ir apdorojimui iš interneto šaltinių. Šiame projekte pirminiais duomenų šaltiniais pasirinktos „Google“ ir „Bing“ paieškos sistemos dėl jų užimamos didelės pasaulinės paieškos sistemų rinkos dalies, kaip parodyta „Statista“ grafike (pav. 8). Pagal 2024 m. Sausio mėnesio duomenis, dominuojanti „Google“ su 81,95 proc. rinkos dalies siūlo išsamius paieškos rezultatus, o antroje vietoje žengia „Bing“ su 10,51 proc. Kartu šios dvi paieškos sistemos užtikrina išsamią duomenų rinkimo strategiją.



Šaltinis: Statista (2024)

8 pav. Pagrindinių pasaulinių stacionariųjų paieškos sistemų rinkos dalis

3.3.1. Žiniatinklio duomenų nuskaitymas naudojant “Google” ir “Bing”

Siekiant automatizuotai rinkti informaciją apie įmones, šiame darbe naudojami du paieškos varikliai – „Google“ ir „Bing“. Abiejų variklių API suteikia galimybę atlikti užklausas pagal įmonių juridinius pavadinimus ir gauti struktūrizuotus paieškos rezultatus JSON formatu. Šis procesas leidžia sistemingai rinkti ir analizuoti duomenis apie įmones.

Abi paieškos API („*Google Custom Search JSON API*“ ir „*Bing Search API*“) reikalauja įgaliojimo raktų, kurie užtikrina autorizuotą prieigą prie paslaugų. Šie raktai gaunami atitinkamai

per „Google Cloud Console“ ir „Microsoft Azure“ platformas. Raktai naudojami siunčiant API užklausas, kurios grąžina paieškos rezultatus, susijusius su pateiktu juridiniu pavadinimu.

Procesas susideda iš kelių etapų:

1. Paieškos užklausos pateikimas:
   - Naudojama sukurta funkcija „google_search“ arba „bing_search“ tam, kad būtų pateikta užklausa atitinkamam API. Užklausose nurodoma, kad reikia grąžinti iki 10 rezultatų.
   - Grąžinti JSON duomenys apima metaduomenis, tokius kaip tinklalapių pavadinimai, nuorodos ir fragmentai (angl. *snippets*).
2. Pilno teksto gavimas iš tinklalapių:

Naudojant funkciją „get_full_text“, iš tinklalapio HTML turinio išgaunamas pagrindinis tekstas. Ši funkcija išrenka tekstą iš <p> žymų, kurios paprastai apima prasmingą tinklalapio informaciją.

- Jei tinklalapio turinys negali būti gautas arba įvyksta klaida, grąžinama reikšmė „None“.

3. Duomenų kaupimas:

Kiekvienas paieškos rezultatas apdorojamas ir paverčiamas struktūrizuotu įrašu, kuriame pateikiama:

- Įmonės unikalus identifikatorius (ID).
- Juridinis pavadinimas.
- Tinklalapio pavadinimas, nuoroda, fragmentas ir visas tekstas.

Nors procesai „Google“ ir „Bing“ paieškos API yra beveik identiški, yra keletas skirtumų:

- **Paieškos API struktūra**:
  - „Google“ API grąžina rezultatus per items lauką, kuriame pateikiami straipsnių pavadinimai, nuorodos ir fragmentai.
  - „Bing“ API grąžina rezultatus per webPages.value lauką, o fragmentai gali turėti HTML žymų, kurias reikia pašalinti.
- **Autentifikacija**:
  - „Google“ API reikalauja API rakto ir paieškos variklio ID (angl. *Custom Search Engine ID*).
  - „Bing“ API naudoja vieną API raktą su pritaikytu antraščių lauku Ocp-Apim-Subscription-Key.
- **Fragmentų formatas**:
  - „Google“ API fragmentai paprastai yra tekstiniai.
  - „*Bing*“ API fragmentuose gali būti HTML dekoracijų, todėl naudojama „*BeautifulSoup*“, kad būtų pašalinti HTML žymos.

Surinkti rezultatai apdorojami ir pateikiami lentelėje su šiais stulpeliais:

- id: Įmonės unikalus identifikatorius.
- legal_name: Įmonės juridinis pavadinimas.
- Title: Tinklalapio ar straipsnio pavadinimas.
- Link: Nuoroda į tinklalapį.
- Snippet: Paieškos variklio sugeneruotas trumpas tinklalapio aprašymas (santrauka).
- FullText: Visas tinklalapio turinys, išgautas iš <p> žymų.

Apibendrinant, „Google“ ir „Bing“ paieškos variklių API leidžia automatizuotai rinkti informaciją apie įmones, naudojant jų juridinius pavadinimus kaip paieškos užklausas. Abu procesai yra panašūs: užklausos pateikiamos per API, grąžinti JSON rezultatai analizuojami, o tinklalapių turinys apdorojamas ir struktūrizuojamas tolesnei analizei. Skirtumai tarp šių API apima autentifikacijos metodus, fragmentų formatą ir rezultatų duomenų struktūrą, tačiau abu įrankiai suteikia galimybę efektyviai kaupti informaciją iš įvairių internetinių šaltinių. Šis integruotas metodas užtikrina platų duomenų aprėptį, jų kokybę ir tinkamumą semantinei analizei, reputacijos tyrimams ar kitoms žiniatinklio duomenų apdorojimo užduotims.

3.4. Iš anksto apmokytų NLP modelių vykdymas

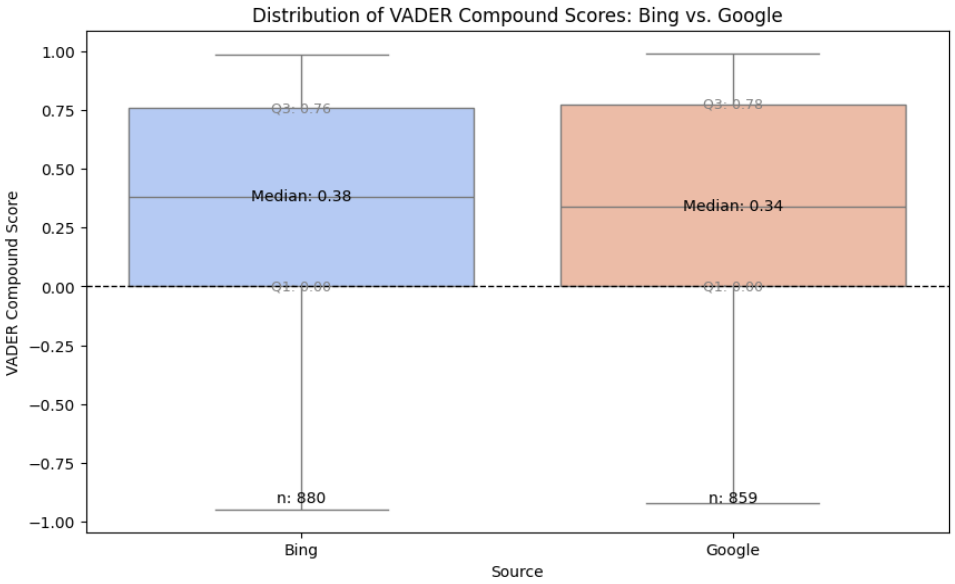
Iš anksto apmokyti natūraliosios kalbos apdorojimo (NLP) modeliai atlieka svarbų vaidmenį išgaunant naudingas įžvalgas iš tekstinių duomenų. Iš „Google“ ir „Bing“ paieškos sistemų surinktuose tekstiniuose duomenyse, kuriuos sudaro fragmentai, antraštės ir viso teksto turinys, pateikiama daugybė informacijos, tačiau jiems dažnai trūksta struktūros ir, norint atskleisti gilesnę prasmę, reikia pažangios analizės. Tokie modeliai kaip „*VADER*“, „*RoBERTa*“, „*BERT*“ ir „*OpenAI*“ „*ChatGPT*“ leidžia sudėtingai apdoroti tekstą, atlikti nuotaikų analizę, semantinį supratimą ir turinio kategorizavimą.

Šie modeliai ypač tinkami paieškos sistemos duomenims apdoroti, nes jais galima nustatyti nuotaikų tendencijas, išskirti įvardytas esybes ir analizuoti kontekstinę reikšmę dideliu mastu. Naudojant iš anksto apmokytus modelius, nestruktūrizuotą tekstinį turinį galima paversti prasmingomis įžvalgomis, išryškinančiomis nuotaikų modelius, reputacijos signalus ar teminę svarbą Šis metodas leidžia geriau ir išsamiau suprasti duomenis, padeda atlikti patikimą analizę ir priimti pagrįstus sprendimus.

3.4.1. VADERS

Šiame skyriuje VADER (*Valence Aware Dictionary and sEntiment Reasoner*) modelis buvo pritaikytas nuotaikų analizei atlikti iš „Google“ ir „Bing“ paieškos sistemų gautiems tekstiniams duomenims. Analizės tikslas buvo kiekybiškai įvertinti iš dviejų šaltinių gautų duomenų nuotaikas ir palyginti jų tonalines charakteristikas. Kiekvienas duomenų rinkinių teksto įrašas buvo apdorotas siekiant nustatyti keturių dimensijų sentimentų balus: teigiamo, neutralaus, neigiamo ir sudėtinio balo, kuris atspindi apibendrintą sentimentą. Atliekant analizę pirmenybė buvo teikiama „*FullText*“ laukui, kai jis buvo prieinamas, ir, siekiant nuoseklumo, jis buvo sutrumpintas iki 500 simbolių, o prireikus buvo naudojamas „*Snippet*“ laukas. Taip buvo užtikrinta, kad būtų veiksmingai analizuojamas ir išsamus turinys, ir glaustos santraukos.

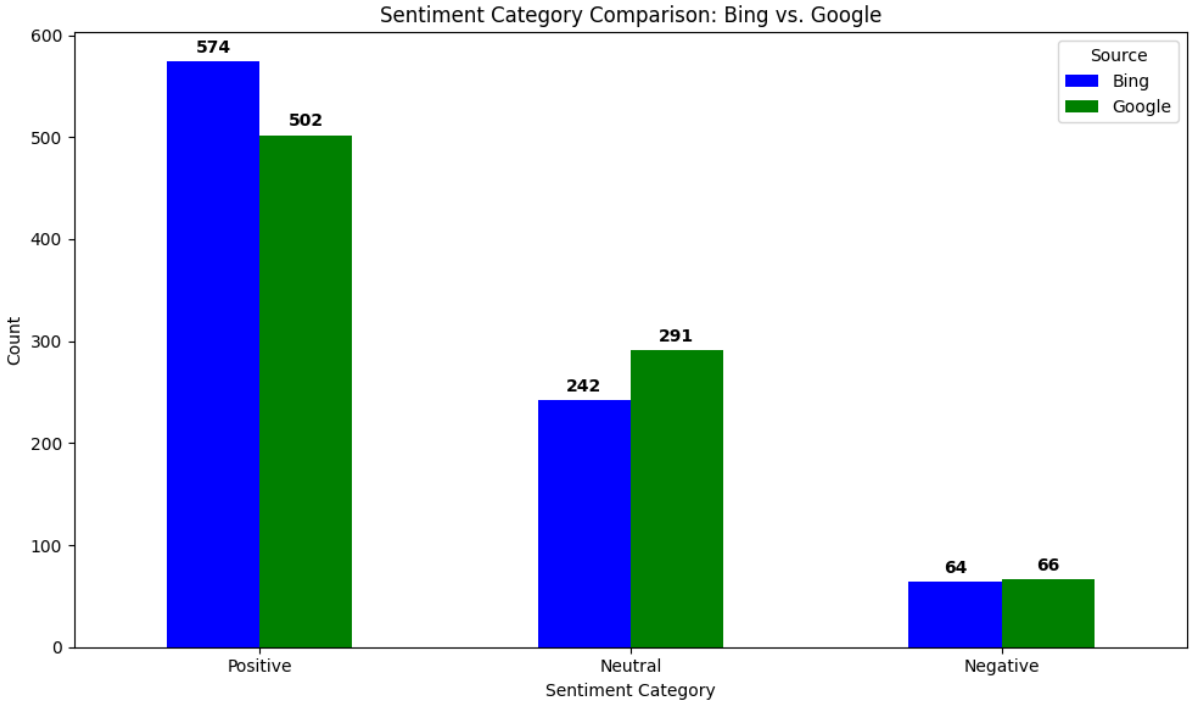
VADER gauti nuotaikų balai buvo vizualizuoti dviem pagrindiniais grafikais, kad būtų galima pateikti lyginamąsias įžvalgas. Pirmojoje vizualizacijoje „boxplot“ (9 pav.)- išryškinamas „Bing“ ir „Google“ duomenų sudėtinių nuotaikų balų pasiskirstymas. Bing“ duomenys rodo šiek tiek didesnę sudėtinio vertinimo medianą (0,38), palyginti su ‚Google‘ (0,34), o tai rodo, kad ‚Bing‘ paieškos rezultatuose apskritai vyrauja pozityvesnis tonas. Abu šaltiniai rodo panašų tarpkvartilinį intervalą: „Bing“ balai svyruoja nuo 0,00 iki 0,76 (I-III ketv.), o „Google“ - nuo 0,00 iki 0,78, o tai rodo, kad abu šaltiniai renka panašaus intensyvumo nuotaikų turinį. Šių intervalų pastovumas pabrėžia abiejų paieškos sistemų patikimumą pateikiant subalansuotą jausmų turtingo turinio derinį, nors atrodo, kad „Bing“ šiek tiek labiau linksta į pozityvumą.



Šaltinis: Sudaryta autoriaus

9 pav. VADER junginių balų pasiskirstymas: „bing“ ir „google

Antroje vizualizacijoje - stulpelinėje diagramoje - įrašai suskirstyti į teigiamų, neutralių ir neigiamų nuotaikų grupes, atskleidžiant pastebimus nuotaikų pasiskirstymo skirtumus tarp abiejų šaltinių. Palyginti su „Google“ (502), „Bing“ buvo daugiau teigiamų įrašų (574), o tai rodo, kad „Bing“ gali teikti pirmenybę arba gauti turinį, kuris yra pozityviau suformuluotas. Priešingai, „Google“ grąžino daugiau neutralaus turinio (291 įrašas) nei „Bing“ (242), o tai gali atspindėti reitingavimo algoritmų arba turinio įvairovės skirtumus. Neigiamos nuotaikos buvo retos abiejuose duomenų rinkiniuose - panašus skaičius Bing (64) ir Google (66), o tai patvirtina, kad abiejose platformose daugiausia pateikiamas teigiamo arba neutralaus tono turinys.



Šaltinis: Sudaryta autoriaus

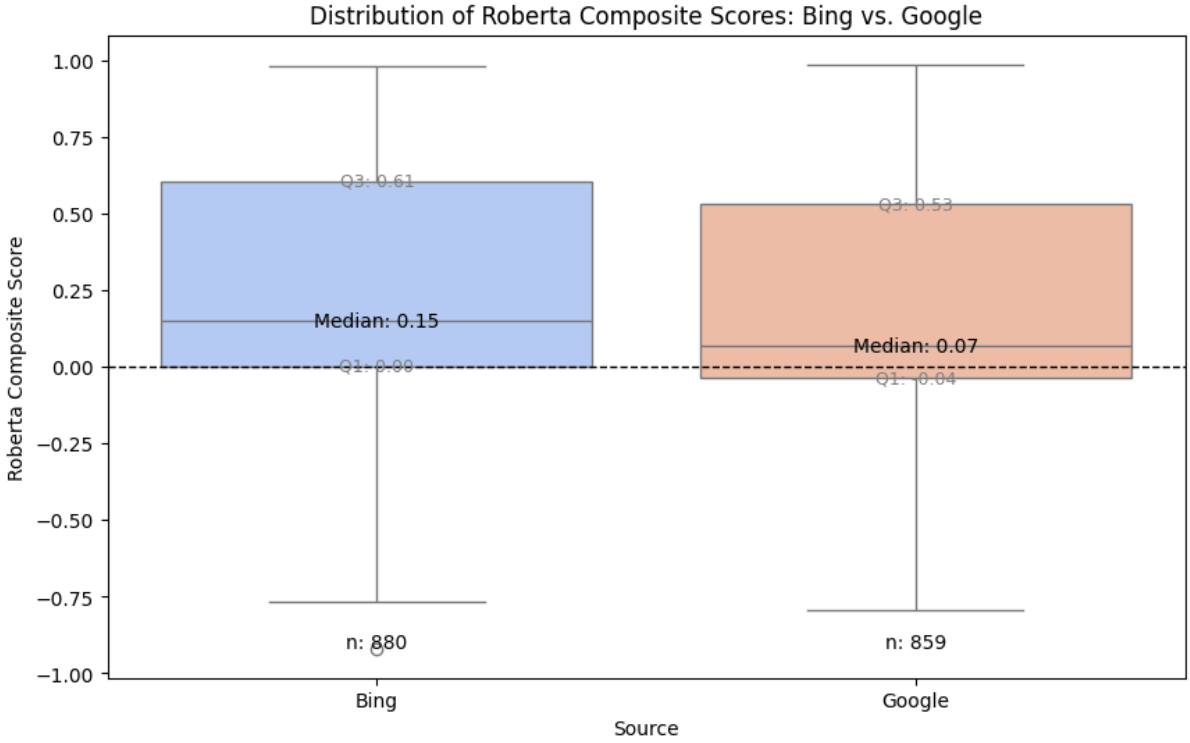
10 pav. Sentimentų kategorijų palyginimas „Google“ ir „Bing“ (VADERS)

Šios analizės rodo, kad VADER yra vertingas, nes leidžia greitai ir lengvai interpretuoti didelių tekstinių duomenų rinkinių nuotaikų balus. Jos taip pat išryškina subtilius paieškos sistemų, tokių kaip „Google“ ir „Bing“, turinio kuravimo skirtumus, kurie gali turėti įtakos naudotojų suvokimui ir sąveikai su gautais rezultatais. Pavyzdžiui, dėl to, kad „Bing“ šiek tiek akcentuoja teigiamas nuotaikas, naudotojų patirtis gali būti optimistiškesnė, o „Google“ didesnė neutralaus turinio dalis gali suteikti labiau subalansuotą informacinį toną. Šios išvados yra

svarbios siekiant suprasti paieškos sistemų elgsenos ir nuotaikomis grindžiamų sprendimų priėmimo pasekmes verslo ir reputacijos valdymo kontekste.

3.4.2. RoBERTa

Šiame skyriuje RoBERTa modelis buvo panaudotas nuotaikų analizei atlikti iš „Google“ ir „Bing“ paieškos sistemų surinktiems tekstiniams duomenims. RoBERTa, transformatoriumi pagrįstas kalbos modelis, yra iš anksto apmokytas fiksuoti teksto nuotaikas ir, naudodamasis išvesties tikimybėmis, pateikia tris nuotaikų balus - neigiamą, neutralų ir teigiamą. Atliekant analizę daugiausia dėmesio buvo skiriama „ FullText“ arba „ Snippet“ laukų naudojimui, priklausomai nuo galimybių, o siekiant efektyviai apdoroti tekstą, jis buvo sutrumpinamas iki maksimalaus modelio įvesties dydžio.

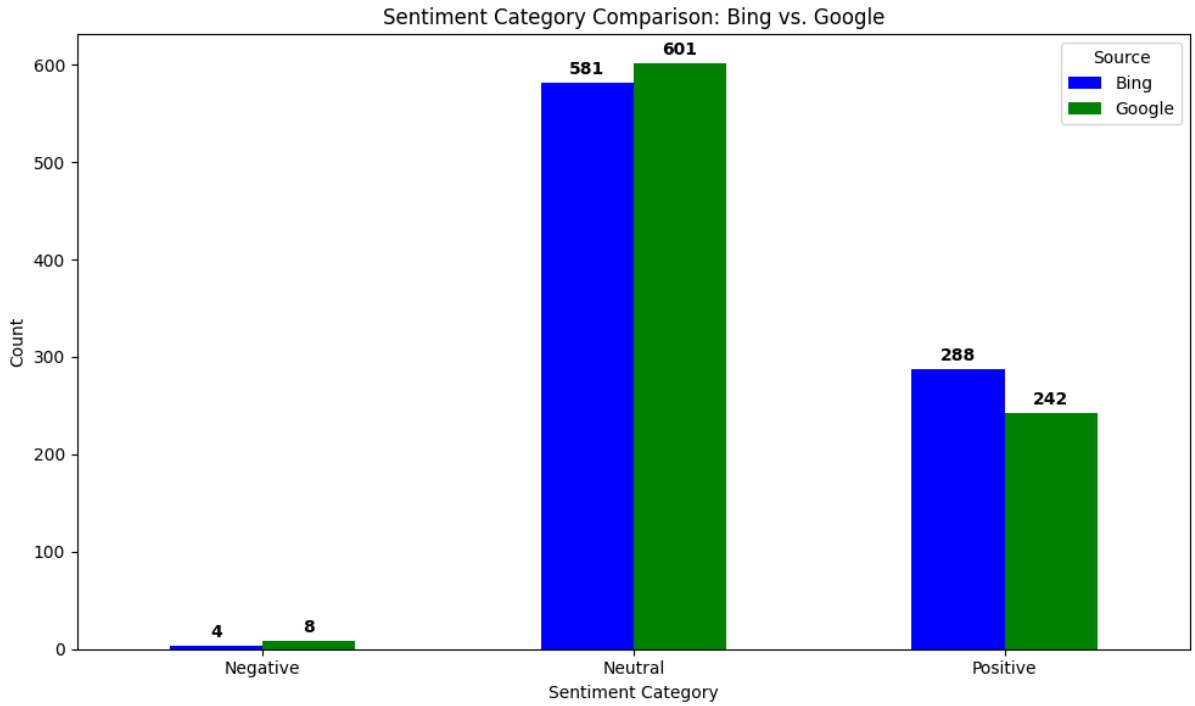


Šaltinis: Sudaryta autoriaus

11 pav. ROBERTA sudėtinių balų pasiskirstymas: „Bing“ ir „Google“

Rezultatai buvo vizualizuoti siekiant palyginti dviejų duomenų šaltinių nuotaikų pasiskirstymą. Pirmoji vizualizacija - dėžutės diagrama - iliustruoja RoBERTa sudėtinių nuotaikų balų pasiskirstymą (apskaičiuotą kaip teigiamas - neigiamas). „Bing“ rezultatų sudėtinio balo mediana didesnė (0,15), palyginti su ‚Google‘ (0,07), o tai rodo, kad ‚Bing‘ duomenys vidutiniškai pasižymi šiek tiek pozityvesnėmis nuotaikomis. Abiejų šaltinių tarpkvartilinis intervalas panašus, o tai rodo, kad dauguma jų turinio patenka į panašų nuotaikų intervalą. Tačiau „Bing“ teigiamų

nuotaikų balų sklaida yra šiek tiek didesnė (Q3 = 0,61) nei „Google“ (Q3 = 0,53), o tai gali atspindėti abiejų paieškos sistemų turinio prioritetų nustatymo ar paieškos skirtumus.



Šaltinis: Sudaryta autoriaus

12 pav. ROBERTA Sentimentų kategorijų palyginimas „Bing“ vs „Google“

Antroje vizualizacijoje - stulpelinėje diagramoje - pateikiamos sentimentų kategorijos, pagrįstos RoBERTa rezultatais. Neutralios nuotaikos dominuoja abiejuose šaltiniuose, o „Google“ jų skaičius didesnis (601), palyginti su „Bing“ (581). Palyginti su „Google“ (242), „Bing“ turi daugiau teigiamų įrašų (288), o tai sutampa su šiek tiek didesniais sudėtiniais balais, pastebėtais dėžutinėje diagramoje. Neigiamos nuotaikos abiejuose duomenų rinkiniuose pasitaiko itin retai: „Google“ yra 8 įrašai, o „Bing“ - tik 4. Tai rodo, kad abiejose paieškos sistemose daugiausia ieškoma neutralaus arba teigiamo turinio.

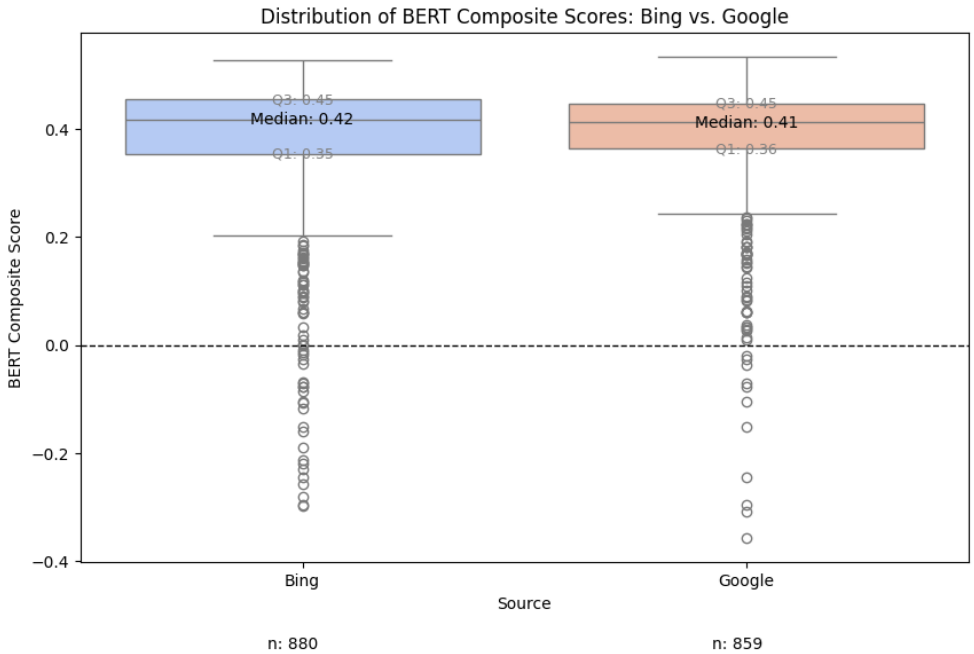
Ši analizė rodo, kad RoBERTa geba atspindėti niuansų skirtumus tarp tekstinių duomenų rinkinių. Palyginimas rodo, kad „Bing“ duomenys yra šiek tiek pozityvesni, o „Google“ pateikia daugiau neutralaus turinio. Šiems skirtumams įtakos gali turėti paieškos sistemų algoritmų ar turinio indeksavimo strategijų skirtumai, be to, jie leidžia suprasti, kaip interneto informacija pateikiama naudotojams priklausomai nuo platformos. Ši analizė suteikia vertingą pagrindą suprasti nuotaikomis grindžiamus interneto turinio modelius ir jų galimą poveikį suvokimui ir sprendimų priėmimui.

3.4.3. BERT

BERT (Bidirectional Encoder Representations from Transformers) modelis buvo taikomas nuotaikoms vertinti tekstiniuose duomenyse, surinktuose iš „Google“ ir „Bing“ paieškos sistemų. Skirtingai nei VADER ir RoBERTa, standartinis BERT įgyvendinimas nuotaikų klasifikavimui suteikia tik du balus: **teigiamą** ir **neigiamą**, be tiesioginio neutralaus ar sudėtinio balo. Norint įveikti šį apribojimą ir gauti labiau interpretuojamą analizės metriką, **sudėtinis balas** buvo apskaičiuotas iš teigiamo balo atimant neigiamą balą (BERT_Composite = BERT_Pos - BERT_Neg). Šis metodas atitinka mokslinių tyrimų praktiką, taikomą kuriant linijinę skalę iš modelių, kuriuose nėra aiškių neutralių nuotaikų rezultatų.

Tekstas iš „FullText“ arba „ Snippet“ laukų buvo simbolizuotas ir apdorotas naudojant BERT modelį, kad būtų sukurtos nuotaikų tikimybės. Kiekvienam įvesties įrašui buvo išskirtos teigiamos ir neigiamos tikimybės ir apskaičiuotas sudėtinis balas. Siekiant išlaikyti visų įrašų nuoseklumą, buvo tvarkomos pasikartojančios nuorodos, o prireikus sentimentų balai buvo vidutiniškai vertinami.

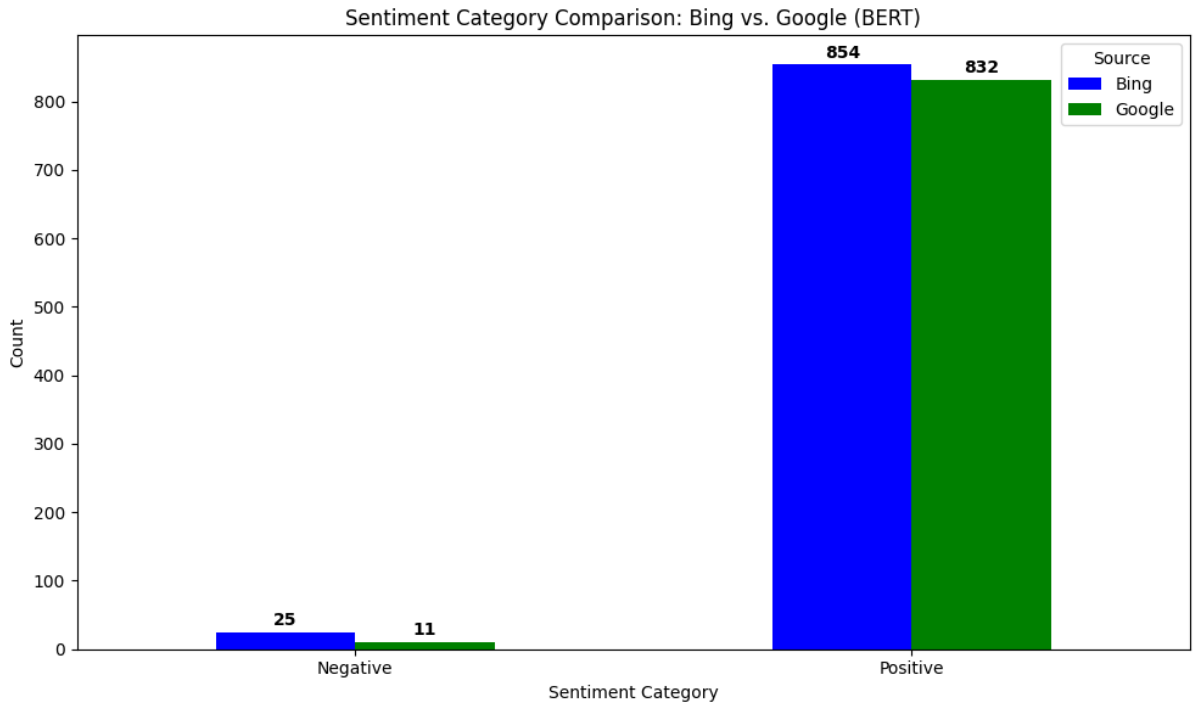
Sentimentų rezultatai buvo vizualizuoti siekiant palyginti „Google“ ir „Bing“ duomenų rinkinius:



Šaltinis: Sudaryta autoriaus

13 pav. BERT sudėtinių balų pasiskirstymas: „bing“ ir „google

BERT sudėtinių balų diagramoje (13 pav.) lyginamas BERT sudėtinių balų pasiskirstymas abiejuose šaltiniuose. Bing“ duomenų medianos sudėtinis balas šiek tiek didesnis (0,42), palyginti su ‚Google‘ (0,41), o abiejų duomenų rinkinių tarpkvartilinis intervalas yra siauras (I-III ketv.), o tai rodo, kad nuotaikų tendencijos sutampa. Nors abiejuose šaltiniuose kartais pasitaiko neigiamų nuokrypių, jų pasiskirstymas iš esmės yra teigiamas. Skirstymų panašumas reiškia, kad abi paieškos sistemos daugiausia pateikia panašaus sentimentų poliariškumo turinį.



Šaltinis: Sudaryta autoriaus

14 pav. Sentimentų kategorijų palyginimas „bing“ ir „google“ (BERT)

Sentimentų kategorijų stulpelinė diagrama (14 pav.) atskleidžia, kad abiejuose duomenų rinkiniuose dominuoja teigiami sentimentai: „Bing“ yra 854 teigiami įrašai, o „Google“ - 832. Neigiamų įrašų yra nedaug: „Bing“ - 25, o „Google“ - tik 11. Tai rodo, kad ETRI nuotaikų klasifikacija atitinka bendrą teigiamą interneto turinio, gaunamo iš paieškos sistemų, vertinimą.

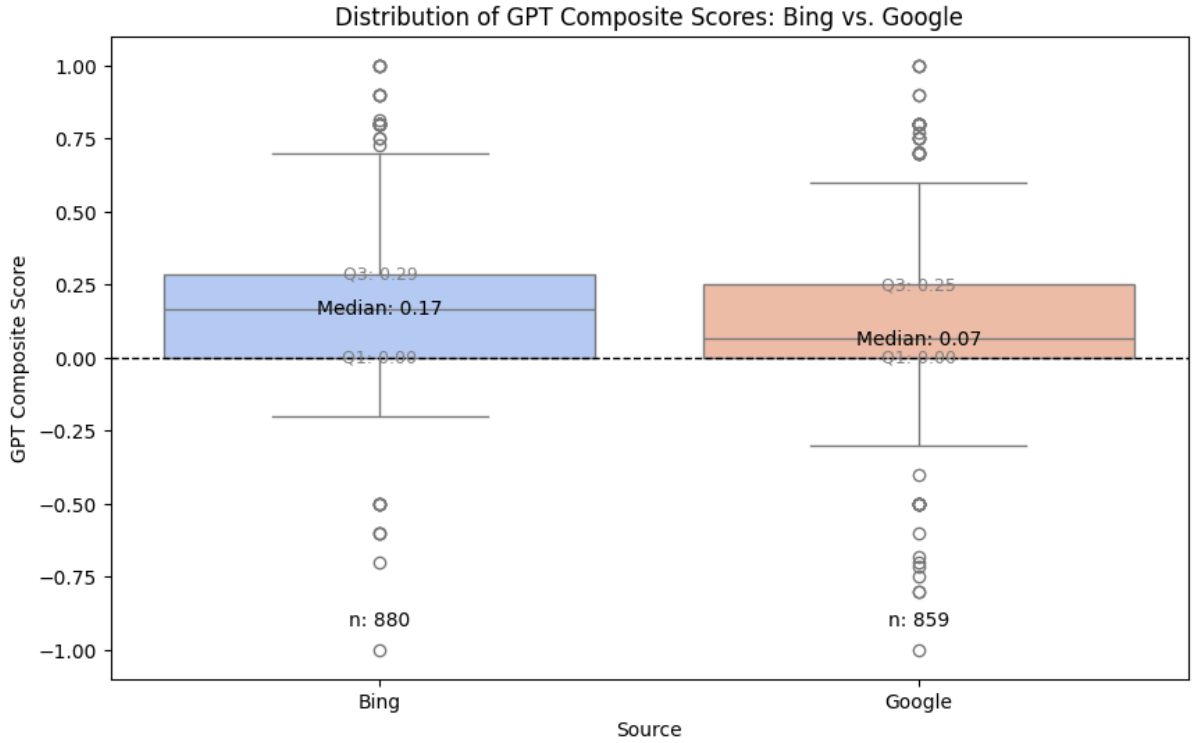
Naudojant ETRI sudėtinį balą, buvo galima atlikti prasmingą nuotaikų analizę, nepaisant to, kad nėra neutralaus rezultato. Šis koregavimas ne tik išsaugojo interpretacijos galimybes, bet ir leido palyginti įvairius duomenų šaltinius. Glaudus „Bing“ ir „Google“ rezultatų atitikimas atspindi bendrą interneto turinio pozityvumo tendenciją. Tačiau konkrečių subjektų atveju pastebėti skirtumai rodo, kad paieškos sistemos gali skirtingai reitinguoti ir teikti pirmenybę turiniui, o tai lemia nuotaikų atspindėjimo skirtumus.

Pritaikant BERT rezultatus ir atliekant lyginamąją analizę šiame tyrime parodyta, kaip iš anksto apmokyti NLP modeliai gali būti naudojami siekiant gauti naudingų įžvalgų, net jei jų pirminiai rezultatai yra riboti. Šis metodologinis lankstumas yra labai svarbus tiriant niuansuotus nuotaikų modelius įvairiuose duomenų rinkiniuose.

3.4.4. OpenAI Chat-GPT

Šiame skyriuje išsamiai aprašomas „OpenAI“ modelio „ChatGPT“ taikymas nuotaikų analizei iš „Google“ ir „Bing“ paieškos sistemų surinktiems tekstiniams duomenims. ChatGPT buvo panaudotas nuotaikoms klasifikuoti į tris kategorijas - neigiamas, neutralias ir teigiamas - tiesiogiai užklausiant modelį ir analizuojant jo struktūrizuotus atsakymus. Kad būtų lengviau atlikti analizę, buvo apskaičiuotas sudėtinis balas kaip teigiamo ir neigiamo balų skirtumas (GPT_Composite = GPT_Pos - GPT_Neg), kad būtų galima tiesiogiai palyginti duomenų šaltinius.

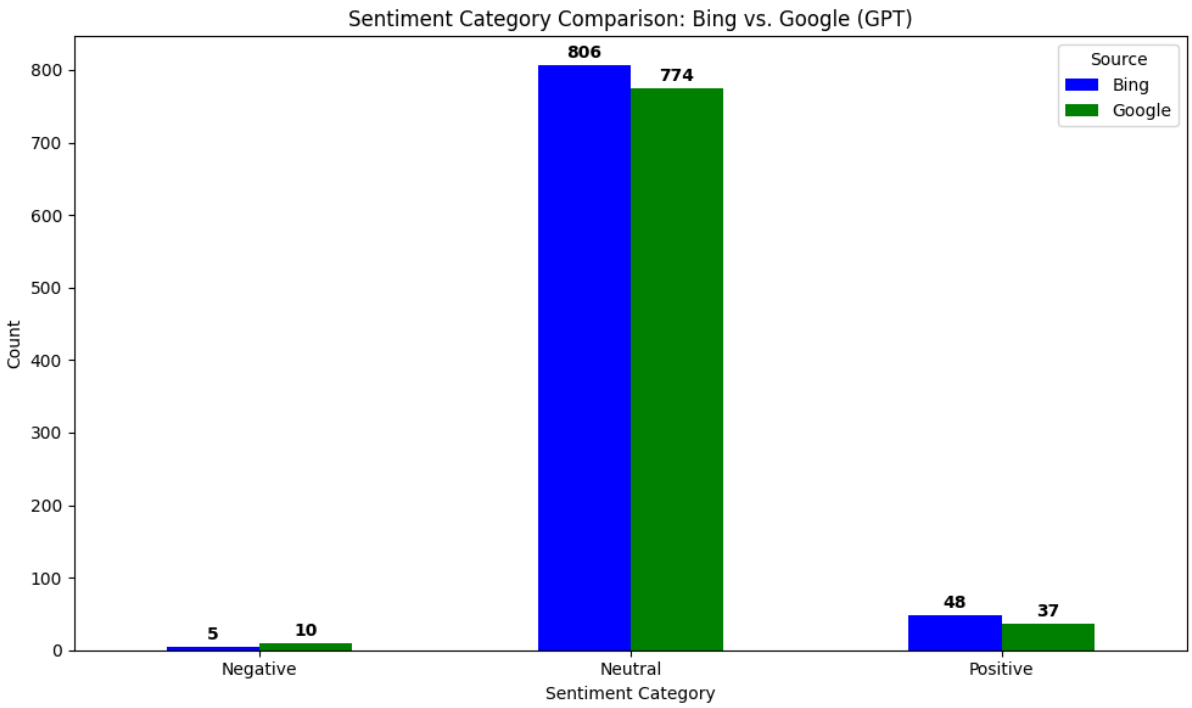
1. Teksto apdorojimas: Kiekvieno įrašo „FullText“ laukas buvo prioritetas analizei, o „Snippet“ laukas buvo naudojamas kaip atsarginis variantas. Tekstas, viršijantis modelio įvesties ribas, buvo sutrumpintas dėl suderinamumo.
2. Modelio sąveika: Užklausa ChatGPT buvo atliekama naudojant struktūrizuotą užklausą, nurodant jai iš anksto nustatytu formatu grąžinti sentimentų balus. Tada šie balai buvo išskirti naudojant reguliariąją išraišką ir suskirstyti į atskirus neigiamų, neutralių ir teigiamų nuotaikų stulpelius.
3. Sudėtinio balo apskaičiavimas: Siekiant apibendrinti nuotaikų poliškumą, buvo apskaičiuotas sudėtinis nuotaikų balas. Šis rodiklis padėjo tiesiogiai palyginti „Bing“ ir „Google“ duomenų rinkinius, todėl buvo galima suprasti, kaip skiriasi kiekvienos paieškos sistemos ieškomo turinio tonai.



Šaltinis: Sudaryta autoriaus

15 pav. GPT sudėtinių balų pasiskirstymas: „Bing“ ir “Google

GPT sudėtinių balų pasiskirstymas (15 pav.) atskleidė, kad „Bing“ duomenys pasižymėjo didesne medianine balo verte (0,17), palyginti su „Google“ (0,07), o tai rodo, kad „Bing“ rezultatuose vidutiniškai vyrauja pozityvesnės nuotaikos. Be to, „Bing“ rezultatų tarpkvartilinis intervalas buvo šiek tiek didesnis, o tai rodo didesnį nuotaikų poliariškumo kintamumą. Tačiau abiejuose šaltiniuose buvo pastebimos neutralios tendencijos, kurias rodo nedidelis balų skirtumas ir didesnis tankis aplink nulį.



Šaltinis: Sudaryta autoriaus

16 pav. Nuotaikų kategorijų palyginimas: „Google“ ir ‚Bing‘ (GPT)

Sentimentų kategorijų palyginimo diagramoje (16 pav.) sentimentai suskirstyti į neigiamus, neutralius ir teigiamus pagal GPT gautus balus. Neutralios nuotaikos vyrauja abiejuose duomenų rinkiniuose: „Bing“ buvo 806, o „Google“ - 774 įrašai. Teigiamų nuotaikų buvo mažiau (Bing - 48, Google - 37), o neigiamų - retai (Bing - 5, Google - 10). Tai rodo, kad abiejose paieškos sistemose pirmiausia paviršiuje pateikiamas neutraliai nuspalvintas turinys, o „Bing“ pasižymi šiek tiek didesniu pozityvumu.

Naudojant ChatGPT nuotaikų analizei, išryškėja jos lankstumas pateikiant išsamius sudėtingų tekstinių duomenų nuotaikų balus. Pastebėti skirtumai tarp „Bing“ ir „Google“ duomenų rinkinių rodo subtilius gauto turinio toninių charakteristikų skirtumus, kuriems gali turėti įtakos paieškos sistemos algoritmai arba turinio reitingavimo strategijos.

ChatGPT struktūrizuoti atsakymai leido atlikti išsamesnę nuotaikų pasiskirstymo analizę, palyginti su tradiciniais modeliais. Vyraujančios neutralios nuotaikos atitinka tipišką interneto turinio pobūdį, kuris dažnai skirtas informuoti arba pateikti subalansuotą požiūrį, o ne sukelti stiprias emocijas. Vis dėlto didesnė teigiama nuotaika „Bing“ rezultatuose gali rodyti tendenciją teikti pirmenybę optimistiškiau nusiteikusiam turiniui, o tai gali būti potencialus konkurencinis pranašumas naudotojų patirties srityje.

Pasinaudojus ChatGPT galimybėmis, ši analizė leidžia geriau suprasti nuotaikų modelius įvairiose platformose ir parodo pažangių NLP modelių naudą lyginamiesiems interneto duomenų analizės tyrimams.

3.5. NLP Metodų tobulinimas naudojant aviacijos kontekstą

Tradiciniai nuotaikų analizės metodai, nors ir veiksmingi bendrajame kontekste, dažnai neatspindi konkrečios srities kalbos niuansų, ypač tokiose labai specializuotose pramonės šakose kaip aviacija. Tokie nuotaikų analizės modeliai kaip VADER, RoBERTa, BERT ir ChatGPT yra iš anksto apmokyti bendriems duomenų rinkiniams, todėl jie nėra optimizuoti interpretuoti aviacijos terminologiją, teisės aktų nuorodas ar šio sektoriaus sankcijų pasekmes. Dėl šio apribojimo šiuos modelius reikia pritaikyti, kad jie geriau suprastų ir klasifikuotų aviacijos kontekste naudojamą tekstą.

Pavyzdžiui, tokius terminus kaip „įžemintas“, „skrydžio draudimas“ ar „juodajame sąraše“ tradiciniai nuotaikų analizės modeliai gali neteisingai interpretuoti kaip visuotinai neigiamai nusiteikusius, nors aviacijos sektoriuje šie terminai gali būti susiję su reguliavimo reikalavimų laikymusi ar saugos priemonėmis, o ne su bendra kritika. Panašiai dėl konkrečios srities mokymo duomenų trūkumo gali būti neatsižvelgta į nuorodas į aviacijos sankcijas ar saugos rekomendacijas arba jos gali būti neteisingai klasifikuojamos. Norint tai išspręsti, būtina iš naujo mokyti arba tikslinti NLP metodus naudojant konkrečios aviacijos srities duomenų rinkinius.

Norint tai pasiekti, labai svarbu įtraukti pasauliniu mastu pripažintų aviacijos reguliavimo institucijų duomenų rinkinius, kuriuose pateikiami išsamūs dokumentai, sankcijų sąrašai ir terminija. Šios įstaigos apima Užsienio turto kontrolės biurą (OFAC), Jungtines Tautas (JT), Europos Sąjungą (ES), Europos Sąjungos aviacijos saugos agentūrą (EASA), Federalinę aviacijos administraciją (FAA) ir Tarptautinę civilinės aviacijos organizaciją (ICAO). Šios organizacijos tvarko išsamius sankcijų sąrašus, skrydžių apribojimus ir atitikties patarimus, kurie yra neįkainojami šaltiniai kuriant konkrečios srities NLP modelius.

Integruojant šiuos konkrečiai aviacijai skirtus duomenis, pertvarkytais NLP metodais siekiama šių tikslų:

- Kontekstinį supratimą: Tiksliai interpretuoti aviacijai būdingus terminus ir frazes jų numatyto konteksto kontekste, sumažinant klaidingą nuotaikų klasifikavimą.
- Sankcijų analizė: Nustatyti ir išskirti aviacijos reguliavimo institucijų pažymėtus subjektus ar asmenis, užtikrinant atitiktį atitikties reikalavimams.

- Padidintas aktualumas: Pagerinkite teksto klasifikavimo ir nuotaikų analizės tikslumą, atsižvelgdami į unikalias aviacijos pramonės reguliavimo ir veiklos sistemas.

Įtraukus aviacijai būdingus žodynus, sankcijų sąrašus ir kontekstinius duomenų rinkinius, šie modeliai galės geriau atitikti aviacijos sektoriaus poreikius, todėl bus naudingesni siekiant užtikrinti atitiktį teisės aktų reikalavimams, įvertinti riziką ir atlikti reputacijos analizę. Šis žingsnis yra esminis NLP metodikų taikymo etapas, kuriuo užtikrinama, kad šios metodikos užtikrintų veiksmingas ir tikslias įžvalgas itin reguliuojamoje ir sudėtingoje srityje.

3.5.1. Aviacijos Sektoriui Aktualių Duomenų Rinkinio Paruošimas

Norint veiksmingai pritaikyti NLP modelius aviacijos sričiai, itin svarbu įtraukti duomenis, kurie tiksliai atspindėtų reguliavimo sistemas, atitikties standartus ir specifinei sričiai būdingą kalbą. Šis skyrius apima sankcijų sąrašų ir kontekstinių duomenų, gautų iš tarptautiniu mastu pripažintų aviacijos reguliavimo institucijų bei organizacijų, sudarymą. Duomenų rinkiniai apima OFAC, JT ir ES sankcijų sąrašus, taip pat Europos Sąjungos aviacijos saugos agentūros (EASA), Federalinės aviacijos administracijos (FAA) ir Tarptautinės civilinės aviacijos organizacijos (ICAO) pranešimus spaudai.

Šie šaltiniai yra labai svarbūs dėl dviejų pagrindinių priežasčių:

- Specifinis svarbumas sričiai: Tradiciniai sentimentų analizės modeliai gali neteisingai interpretuoti aviacijos terminologiją ar nepaisyti sankcionuotų subjektų konteksto. Tai kelia riziką atitikties ir rizikos analizei.
- Aviacijai būdingas kontekstualizavimas: Integruojant sankcijų sąrašus ir reguliavimo duomenis, NLP modeliai gali geriau suprasti specifinius aviacijos kontekstus, tokius kaip skrydžių apribojimai, sankcijos ir saugos rekomendacijos.

Pavyzdžiui, bendruosiuose modeliuose terminas „restricted entity“ gali būti neutralus, tačiau aviacijoje jis žymi didelės rizikos subjektus, kuriems taikomos sankcijos. Įtraukus specifinius žodynus ir kontekstinius duomenis, NLP modeliai gali tinkamai atpažinti reguliavimo terminus, kaip „skrydžio draudimas“ ar „tinkamumo skraidyti direktyva“.

Visi duomenys buvo apdoroti pagal struktūrizuotą darbo eigą, kurią sudaro šie pagrindiniai žingsniai:

4 lentelė

Aviacijos duomenų apdorojimo žingsniai

Duomenų šaltinis	**Paruošimo žingsniai**	**Rezultatas**
OFAC Sankcijų Sąrašas	- XML duomenų ištrauka su requests ir ElementTree. - Alternatyvūs pavadinimai (aka), adresai (add.csv) ir komentarai (sdn_comments.csv) sujungti naudojant pandas.merge. - Duomenys susieti per uid.	- Praturtinti OFAC įrašai su papildomais atributais, pvz., alternatyviais pavadinimais ir adresais. - Normalizuoti duomenys: pašalinti NaN, pertekliniai simboliai („-0-“).
JT Konsoliduotas Sąrašas	- Fizinių ir juridinių asmenų įrašai buvo išskirti iš XML struktūros. - Apdoroti atributai: pavadinimai, adresai, sankcijų paskyrimai. - Konvertuota į vieną DataFrame.	- Unifikuoti įrašai su specifine informacija apie kiekvieną subjektą.
ES Sankcijų Sąrašas	- XML duomenų analizė su namespace. - Išgauti alternatyvūs pavadinimai, gimimo datos, pilietybės ir reguliavimo nuorodos. - Normalizuoti duomenys į sąrašus.	- Struktūrizuoti įrašai su programų ir teisės aktų pagrindais.
Reguliavimo Pranešimai Spaudai	- **EASA**: Naudojant BeautifulSoup, buvo išskirtos ir išvalytos susijusios naujienos. - **FAA**: Panašiai apdoroti spaudos pranešimai, pašalinant pasikartojančius šablonus ir HTML. - **ICAO**: Apdorota keliais etapais, pašalinant nereikšmingą tekstą.	- Praturtinti straipsnių tekstai, atspindintys aktualius aviacijos reguliavimo duomenis.

Šaltinis: sudaryta autoriaus

Po duomenų surinkimo, buvo atlikti šie apdorojimo žingsniai:

1. Valymas ir Normalizavimas: Pašalintos nereikalingos reikšmės, žymos ir pasikartojimai. Tekstai buvo apdoroti siekiant išvengti formatavimo neatitikimų.

2. Duomenų Žymėjimas: Subjektai buvo pažymėti kaip „didelės rizikos“ arba „reguliavimo“ subjektai, priklausomai nuo jų konteksto.
3. Sujungimas į Vieningą Rinkinį: Visi duomenys buvo integruoti į bendrą struktūrą, siekiant užtikrinti suderinamumą ir paruošti NLP modelių mokymui.

Galutinis produktas – tai struktūrizuotas ir praturtintas duomenų rinkinys, apimantis pasaulinius sankcijų sąrašus, reguliavimo duomenis ir pranešimus spaudai. Šis rinkinys:

- Tiksliai identifikuoja sankcionuotus subjektus.
- Kontekstualizuoja aviacijai specifinius terminus ir frazes.
- Suteikia tvirtą pagrindą NLP modelių perkvalifikavimui, siekiant geresnės sentimentų analizės ir atitikties vertinimo aviacijos sektoriuje.

Tokiu būdu, šie duomenys leis tiksliai ir veiksmingai spręsti realias aviacijos atitikties problemas.

3.5.2. VADERS atnaujinimas

Šiame skyriuje VADER nuotaikų analizės modelis patobulinamas įtraukiant aviacijos terminologiją, gautą iš kuruojamų aviacijos srities duomenų rinkinių. Tikslas - išplėsti ir pritaikyti modelio leksikoną su konkrečiai pramonei būdingais raktažodžiais ir kontekstine informacija, taip pagerinant jo tinkamumą sentimentų analizės užduotims aviacijos sektoriuje.

Ankstesniuose skyriuose surinkti su aviacija susiję duomenys, įskaitant reguliavimo leidinius, sankcijų sąrašus ir pranešimus spaudai, sudaro VADER leksikono praturtinimo pagrindą. Šie duomenų rinkiniai yra du svarbūs komponentai:

EASA, FAA ir ICAO paskelbtuose dokumentuose pateikiami raktažodžiai ir terminai, taip pat OFAC, JT ir ES sąrašuose esančių subjektų, kuriems taikomos sankcijos, pavadinimai ir slapyvardžiai. Antra, jie apima kontekstinę informaciją, kuri yra labai svarbi aviacijos sričiai, pavyzdžiui, tokias frazes kaip „skrydžio draudimas“ arba „aviacijos saugos pažeidimas“, kurios yra labai svarbios nuotaikų analizei, tačiau standartiniai modeliai jų gali neatpažinti.

Pavyzdžiui, tokia bendra sąvoka kaip „riboto naudojimo subjektas“ tradiciniuose nuotaikų analizės modeliuose gali atrodyti neutrali, tačiau aviacijoje ji dažnai reiškia didelės rizikos organizacijas arba organizacijas, kurioms taikomos sankcijos. Panašiai, tokie terminai kaip „skrydžio draudimas“ arba „orlaivio nutupdymas“ šioje srityje sukelia daug neigiamų nuotaikų. Juos įtraukus į VADER modelį, galima gerokai padidinti jo analitines galimybes.

Siekiant pritaikyti VADER modelį konkrečioms aviacijos srities sentimentų analizės užduotims, buvo panaudota struktūruota darbo eiga:

Pirminiam duomenų apdorojimui buvo pasitelkti EASA, FAA, ICAO, OFAC, JT ir ES sankcijų sąrašų duomenų rinkiniai kurie buvo sujungti ir iš anksto apdoroti. Šis pirminis apdorojimas apėmė teksto normalizavimą - viso teksto konvertavimą į mažąsias raides, skyrybos ženklų pašalinimą ir terminijos nuoseklumo užtikrinimą. Naudojant TF-IDF vektorizatorių, buvo nustatyti pagrindiniai aviacijos terminai ir nustatyti jų svoriai pagal dažnumą ir svarbą duomenų rinkiniuose.

Leksikono kūrimas pagal užsakymą:

- Sankcijų sąrašo integravimas: Sankcijų sąrašuose esančių subjektų pavadinimai ir slapyvardžiai buvo tiesiogiai įtraukti į VADER leksikoną su numatytuoju neigiamu balu (-4,0). Šis koregavimas pabrėžia jų didelę riziką atitikties kontekste. Pavyzdžiui, su sankcijomis susiję subjektai buvo įvertinti neigiamu balu, taip užtikrinant tikslų atspindėjimą nuotaikų analizėje.
- TF-IDF svertinis vertinimas: Kiekvieno aviacijos termino reikšmė apskaičiuota naudojant normalizuotus TF-IDF balus. Tada šie balai buvo sujungti su esamais VADER sentimentų balais, kad būtų galima priskirti svorius, atspindinčius jų kontekstinę svarbą. Pavyzdžiui, tokiems terminams, kaip „aviacijos saugos pažeidimas“, dėl jų svarbos su aviacija susijusiuose tekstuose buvo suteikti didesni svoriai, o mažiau svarbūs terminai buvo koreguojami minimaliai.

Šio proceso metu VADER leksikonas buvo papildytas 582 naujais su aviacija susijusiais terminais. Tokiems terminams, kaip „ribojamas subjektas“, „skrydžio draudimas“ ir „orlaivio įžeminimas“, buvo priskirti sentimentų balai, sukalibruoti pagal VADER balų intervalą nuo -4 iki +4. Šie balai buvo perskaičiuoti, kad būtų užtikrintas suderinamumas su pradiniu VADER leksikonu.

Skirtingai nuo mašininio mokymosi modelių, kuriems reikia didelio perkvalifikavimo, VADER veikia kaip leksikonu pagrįstas modelis, kuris palaiko tiesioginį žodyno atnaujinimą. Ši savybė leido veiksmingai integruoti aviacijai būdingą terminiją. Pagrindiniai etapai buvo šie:

1. Žodyno atnaujinimas: praturtintas leksikonas buvo įtrauktas į esamą VADER žodyną, padidinant jo dydį nuo 7 502 pagrindinio žodyno įrašų iki 9 084 įrašų atnaujintame aviacijos srities žodyne.

2. Patvirtinimas ir testavimas: Atnaujintas VADER modelis buvo išbandytas su konkrečios aviacijos duomenų rinkiniais, kad būtų užtikrintas tikslus pagrindinių terminų aiškinimas. Pavyzdžiui, jis sėkmingai identifikavo tokius terminus kaip „skrydžio draudimas“ kaip neigiamą ir „reguliuojančių institucijų patvirtinimas“ kaip teigiamą, o tai rodo atnaujinimų veiksmingumą.

Patobulintas VADER nuotaikų analizės modelis dabar yra pritaikytas aviacijos sektoriui ir turi keletą skirtingų galimybių:

- Aviacijos terminų atpažinimas ir vertinimas balais: Išplėstas leksikonas leidžia VADER priskirti reikšmingus nuotaikos balus aviacijai būdingiems raktažodžiams, todėl galima tiksliai klasifikuoti nuotaikas.
- kontekstinis terminų vertinimas: TF-IDF svorių integravimas užtikrina, kad terminai būtų koreguojami kontekstualiai, atsižvelgiant į jų svarbą su aviacija susijusiuose duomenų rinkiniuose.
- Patobulinta konkrečios srities analizė: Dabar modelis žymiai tiksliau atlieka su aviacija susijusių tekstų, pavyzdžiui, reguliavimo dokumentų ir naujienų straipsnių, nuotaikų analizę.

Atnaujintas VADER modelis suteikia reikšmingų privalumų atliekant sentimentų analizės užduotis aviacijos pramonėje.

Pirma, patobulintas jo veikimas užtikrina didesnį tikslumą klasifikuojant su aviacija susijusias nuotaikas, o tai labai svarbu tokioms užduotims kaip rizikos vertinimas ir atitikties stebėsena. Antra, modelis išlieka labai lengvai pritaikomas, todėl į jį galima lanksčiai įtraukti naujus aviacijos terminus, kai jie atsiranda. Galiausiai, dėl pritaikytos konstrukcijos jį galima taikyti įvairiais aviacijos srities atvejais, pradedant reguliavimo tendencijų analize ir baigiant aviacijos naujienų ir pranešimų spaudai nuotaikų stebėsena.

Ši naujovė yra reikšmingas žingsnis į priekį konkrečios srities nuotaikų analizės srityje, leidžiantis aviacijos sektoriuje taikyti tikslesnes ir kontekstualesnes programas.

3.5.2. roBERTa atnaujinimas

Sentimentų analizės modelio RoBERTa pritaikymas ir perkvalifikavimas konkrečioms aviacijos užduotims buvo kruopščiai struktūruotas procesas, kuriame buvo akcentuojamas konkrečios srities duomenų integravimas ir griežtas išankstinis apdorojimas. Šis metodas užtikrino, kad modelis galėtų veiksmingai analizuoti ir klasifikuoti nuotaikas specializuotame

aviacijos pramonės kontekste. Norint pritaikyti RoBERTa konkrečioms aviacijos užduotims, pirmiausia reikėjo surinkti išsamų duomenų rinkinį, apimantį įvairius su aviacija susijusių tekstų šaltinius. Duomenys buvo gauti iš trijų pagrindinių kategorijų:

- Aviaciją reguliuojantys subjektai: EASA, FAA ir ICAO organizacijų tekstai sudarė duomenų rinkinio pagrindą. Juos sudarė pranešimai spaudai, reguliavimo atnaujinimai ir su sauga susiję pranešimai, kurie suteikė svarbų kontekstą ir aviacijos sričiai būdingą terminologiją.
- Sankcijų sąrašai: Buvo įtraukti didelės rizikos subjektai iš OFAC, JT ir ES tvarkomų sankcijų sąrašų. Šių subjektų pavadinimai ir slapyvardžiai buvo laikomi labai svarbiais nustatant ir klasifikuojant neigiamas nuotaikas aviacijos kontekste.
- Jungtinis korpusas: Su aviacija susiję tekstai iš reguliavimo institucijų ir sankcijų sąrašų buvo sujungti į vieną korpusą. Taip buvo gautas vientisas duomenų rinkinys, atspindintis ir konkrečiai pramonei būdingą kalbą, ir su rizika susijusią terminologiją.

Išankstinio apdorojimo etape buvo užtikrinta, kad surinkti tekstiniai duomenys būtų išvalyti, normalizuoti ir paruošti modelio mokymui. Buvo atlikti keli etapai:

- Teksto normalizavimas: Siekiant išlaikyti vienodumą visame duomenų rinkinyje, visas tekstas buvo paverstas mažosiomis raidėmis. Specialiosios raidės ir skyrybos ženklai buvo pašalinti naudojant reguliarias išraiškas, kad būtų pašalintas triukšmas.
- Konkatenavimas (angl. *concatenation*): Tekstai iš kelių sankcijų sąrašų stulpelių, pavyzdžiui, vardai, antrieji vardai ir slapyvardžiai, buvo sujungti į vieną eilutę. Taip užtikrinta, kad iš duomenų rinkinio nebūtų pašalinta jokia svarbi informacija.
- Duomenų rinkinio sudarymas: Su aviacija susiję tekstai iš reguliavimo institucijų ir su sankcijomis susiję tekstai buvo sujungti į vieningą duomenų rinkinį, atspindintį įvairius aviacijos scenarijus.

Duomenų rinkinio ženklinimas buvo labai svarbus žingsnis rengiant duomenų rinkinį prižiūrimam mokymui. Ženklinimo procesą sudarė du metodai:

- Aviacijos tekstų nuotaikų klasifikavimas: Reguliavimo subjektų tekstai buvo ženklinami naudojant iš anksto apmokytą RoBERTa modelį (cardiffnlp/twitter-roberta-base-sentiment). Kiekvienam tekstui buvo priskirta viena iš trijų nuotaikų kategorijų: neigiama, neutrali arba teigiama, remiantis modelio tikimybiniais rezultatais.
- Su sankcijomis susiję tekstai: Vardai ir slapyvardžiai iš sankcijų sąrašų buvo tiesiogiai žymimi kaip neigiami, nes jie iš esmės siejami su didelės rizikos subjektais ir veikla.

Šio ženklinimo proceso metu buvo gautas išsamus duomenų rinkinys, kuriame kiekvienas tekstas buvo suporuotas su nuotaikos etikete, taip palengvinant prižiūrimą mokymąsi. Siekiant paruošti duomenų rinkinį mokymui, paženklinti duomenys buvo suskirstyti į struktūrizuotą formatą:

- „Text“ skiltyje buvo pateikti išvalyti ir iš anksto apdoroti tekstiniai duomenys.
- „Label“ stulpelyje buvo koduojamos skaitmeninės nuotaikų etiketės: 0 - neigiamas, 1 - neutralus, 2 - teigiamas.

Siekiant užtikrinti atsitiktinumą, duomenų rinkinys buvo sumaišytas, o tada padalytas į mokymo (80 %) ir tikrinimo (20 %) pogrupius. Siekiant palaikyti klasių pusiausvyrą pogrupiuose, buvo naudojama stratifikuota atranka, užtikrinant, kad modelis mokymo metu būtų veikiamas visų nuotaikų kategorijų.

Mokymo ir tikrinimo duomenų rinkiniai buvo simbolizuoti, kad juos būtų galima įvesti į „RoBERTa“ modelį. Tokenizavimo procesą sudarė:

- Tokenizatoriaus įkėlimas Tekstui teksto teksto teksto teksto simbolizacijai buvo įkeltas „Hugging Face RobertaTokenizer“.

- Teksto kodavimas: Kiekvieno teksto žymėjimas: kiekvienas tekstas buvo žymimas, sutrumpintas iki maksimalaus 512 simbolių ilgio ir užpildytas, kad būtų užtikrintas vienodas įvesties dydis partijose.
- Duomenų rinkinio įpakavimas: Tokenizuoti duomenys buvo sudėti į pasirinktinę „PyTorch Dataset“ klasę, koduotus įvesties duomenis susiejant su atitinkamomis nuotaikų etiketėmis.

Taip užtikrinta, kad duomenys būtų tinkamo formato, kad juos būtų galima apdoroti „RoBERTa“ modeliu. RoBERTa modelis (roberta-base) buvo pritaikytas sentimentų klasifikavimui naudojant paruoštą duomenų rinkinį. Mokymo procesą sudarė šie etapai:

- Modelio inicijavimas: Iš anksto apmokytas „RoBERTa“ modelis buvo įkeltas ir sukonfigūruotas taip, kad išvestų tris sentimentų etiketes.
- Mokymo konfigūracija: Nustatyti mokymo argumentai, įskaitant:
  - 16 mokymo ir vertinimo partijų dydis.
  - Trys mokymo epochos, kad būtų suderintas skaičiavimo efektyvumas ir tinkamas mokymasis.
  - Mokymosi greičio planuotojas su 500 apšilimo žingsnių, kad būtų užtikrintas stabilumas ankstyvaisiais mokymo etapais.
  - Svorio mažėjimas 0,01, kad būtų sumažintas perteklinis pritaikymas.

- Mokymo vykdymas: Mokymo ciklui valdyti buvo naudojama „Hugging Face Trainer “ klasė, apimanti ir mokymo, ir vertinimo etapus. Modelis buvo vertinamas kiekvienos epochos pabaigoje, kad būtų galima stebėti, kaip jis veikia patikrinimo rinkinyje.

Mokymo procese, kur įmanoma, buvo naudojamas GPU pagreitinimas, todėl gerokai sutrumpėjo mokymo laikas. Baigus mokymą, iš naujo apmokytas „RoBERTa“ modelis parodė, kad sugeba tiksliai klasifikuoti su aviacija susijusias nuotaikas. Pagrindiniai rezultatai:

- Nuotaikų klasifikavimo tikslumas: Modelis veiksmingai atskyrė neigiamas, neutralias ir teigiamas nuotaikas aviacijos tekstuose, pavyzdžiui, „skrydžių draudimas“ buvo įvertintas neigiamai, o „reguliavimo patvirtinimas“ - teigiamai.
- Galimybė taikyti didelės rizikos kontekstuose: Integravus su sankcijomis susijusius duomenis, modelis tiksliai nustatė ir klasifikavo didelės rizikos subjektus kaip neigiamus.

Apmokytas modelis buvo išsaugotas diegimui, todėl jį buvo galima naudoti realiose nuotaikų analizės užduotyse. Šis procesas pabrėžia konkrečiai sričiai būdingo pritaikymo svarbą gerinant iš anksto apmokytų modelių veikimą. Įtraukiant konkrečiai aviacijai būdingus duomenis ir kruopščiai ženklinant duomenų rinkinį, „RoBERTa“ modelis buvo sėkmingai pritaikytas sentimentų analizei aviacijos sektoriuje. Šis pritaikytas metodas užtikrina, kad modelis gali spręsti unikalius su aviacija susijusių užduočių iššūkius ir tapti patikima sentimentų klasifikavimo, rizikos vertinimo ir reguliavimo atitikties užtikrinimo priemone.

3.5.3. BERT atnaujinimas

BERT modelio perkvalifikavimo procesas, skirtas konkrečiai aviacijai būdingų nuotaikų analizei, apėmė keletą kruopščių etapų, kurių kiekvienas buvo skirtas užtikrinti, kad modelis galėtų veiksmingai prisitaikyti prie niuansuotos ir labai specializuotos aviacijos srities. Toliau pateikiamas išsamus šių etapų ir jų reikšmės tyrimas.

Procesas prasidėjo nuo tekstinių duomenų iš įvairių autoritetingų šaltinių kaupimo. Tai buvo aviacijos institucijų, tokių kaip „EASA“, „FAA“ ir „ICAO“, reguliavimo dokumentai ir pranešimai spaudai, taip pat duomenys apie sankcijas iš „OFAC“, „JT“ ir „ES“ sankcijų sąrašų. Su sankcijomis susijusių duomenų įtraukimas buvo ypač svarbus siekiant užfiksuoti neigiamas nuotaikas, susijusias su didelės rizikos subjektais. Pavyzdžiui, sankcijų sąrašų atveju, siekiant išsaugoti išsamų subjektų atvaizdavimą, buvo naudojami keli pavadinimų laukai, pavyzdžiui, slapyvardžiai ir sujungtos pavadinimų dalys. Taip buvo užtikrinta, kad duomenų rinkinys

atspindėtų realius kontekstus, kuriuose šie subjektai gali atsirasti su aviacija susijusiose diskusijose.

Išankstinis duomenų apdorojimas buvo pagrindinis žingsnis standartizuojant tekstą. Tam reikėjo visą tekstą konvertuoti į mažąsias raides, kad būtų išvengta didžiųjų raidžių jautrumo problemų, ir pašalinti skyrybos ženklus, simbolius ir kitus ne raidinius skaitmeninius ženklus. Tokiu būdu duomenys buvo paversti švariu ir vienodu formatu, todėl jie atitiko ETRI žymeklio reikalavimus. Be to, šis veiksmas padėjo pašalinti neatitikimus, dėl kurių į modelį galėjo patekti triukšmo, ir taip pagerino jo mokymosi galimybes.

Sentimentų ženklinimas buvo atliekamas remiantis teksto šaltiniams būdingu pobūdžiu. Su aviacija susijusių tekstų nuotaikos buvo vertinamos rankiniu arba pusiau automatiniu būdu. Reguliavimo atnaujinimai ar teigiami pokyčiai buvo žymimi kaip „teigiami“, o diskusijos apie avarijas, pažeidimus ar draudimus - kaip „neigiami“. Su sankcijomis susijusiuose duomenų rinkiniuose visi įrašai buvo žymimi kaip „neigiami“, nes buvo siejami su subjektais, kuriems taikomi apribojimai, arba didelės rizikos veikla. Šis ženklinimo procesas buvo labai svarbus siekiant užtikrinti, kad mokymo duomenys modeliui suteiktų aiškius ir tikslius signalus nuotaikų skirstymui.

Kai duomenys buvo paženklinti, jie buvo sujungti į vieną duomenų rinkinį ir išmaišyti, kad jokie konkrečių šaltinių modeliai neturėtų įtakos mokymo procesui. Tada sujungtas duomenų rinkinys buvo padalytas į mokymo ir patvirtinimo poaibius, išlaikant 80:20 santykį. Šis sluoksninis padalijimas užtikrino, kad nuotaikų pasiskirstymas būtų nuoseklus abiejuose rinkiniuose, o tai svarbu vertinant modelio veikimą panašiomis sąlygomis į tas, kuriomis jis buvo apmokytas.

Teksto teksto paruošimui ETRI labai svarbus buvo teksto žymėjimas. Naudojant „bert-base-uncased“ tokenizatorių, tekstas buvo paverstas į įvesties ID ir dėmesio kaukes, kurių reikia, kad modelis galėtų apdoroti duomenis. Tokenizatorius pridėjo specialių ženklų, žyminčių sakinių pradžią ir pabaigą, ir sutrumpino tekstus, kurių ilgis viršijo didžiausią leistiną 512 ženklų ilgį. Šiuo veiksmu užtikrintas suderinamumas su ETRI modelio architektūra, kuri skirta fiksuoto ilgio įvestims apdoroti.

Tiksliam derinimo procesui reikėjo pritaikyti iš anksto apmokytą modelį dvejetainiam sekų klasifikavimui. Tam reikėjo modifikuoti modelio architektūrą, pridedant klasifikavimo galvutę, pritaikytą dviem nuotaikų klasėms: teigiamai ir neigiamai. Mokymo metu modelio parametrams koreguoti naudotas AdamW optimizatorius. Šis optimizatorius specialiai sukurtas darbui su

transformuojamaisiais modeliais, nes jame numatytas svorių mažėjimas, kad būtų išvengta perteklinio pritaikymo.

Mokymo ciklas buvo sukurtas taip, kad maksimaliai padidintų modelio mokymosi efektyvumą. Kiekvienai epochai modelis buvo nustatomas į mokymo režimą ir kiekvienai partijai apskaičiuojamos nuostolių vertės. Apskaičiavus gradientus, modelis atnaujindavo savo svorius naudodamas grįžtamąjį srautą. Kiekvienos epochos pabaigoje modelis pereidavo į vertinimo režimą, kad būtų apskaičiuoti patvirtinimo nuostoliai, iš anksto parodantys, kaip gerai jis apibendrina nematytus duomenis. Nuolatinis mokymo ir patvirtinimo nuostolių mažėjimas per epochas parodė, kad modelis efektyviai mokėsi iš duomenų nepermokėdamas.

Išsaugojus modelį po mokymo, buvo užtikrinta, kad tiksliai sureguliuotą BERT modelį būtų galima naudoti pakartotinai ir nereikėtų jo mokyti iš naujo. Į išsaugotą modelį buvo įtraukti apmokyti svoriai, tokenizatorius ir konfigūracijos failai, todėl jį buvo galima sklandžiai įdiegti nuotaikų analizės užduotims atlikti. Pasirinkus išsaugoti modelį „Google Drive“, buvo užtikrintas jo prieinamumas ir atkuriamumas.

Visas pakartotinio mokymo procesas buvo sukurtas atsižvelgiant į specifinius aviacijos srities reikalavimus. Į modelį įtraukus aviacijai būdingą terminologiją ir kruopščiai paženklintus duomenis, modelis įgijo gebėjimą įžvelgti subtilias nuotaikų užuominas, kurios šiame sektoriuje yra labai svarbios. Pavyzdžiui, į modelio mokymosi procesą buvo integruoti tokie terminai kaip „atitikties pažeidimas“ arba „skrydžio nutraukimas“, kurie aviacijoje turi didelę reikšmę, todėl modelis galėjo tiksliai interpretuoti tokias frazes.

Pakartotinai išmokytas BERT modelis yra galinga nuotaikų analizės aviacijos srityje priemonė. Dabar jis gali apdoroti sudėtingus tekstus, labai tiksliai klasifikuoti nuotaikas ir pateikti naudingų įžvalgų įvairiais atvejais - nuo rizikos vertinimo ir atitikties stebėsenos iki reguliavimo tendencijų analizės. Šis griežtas ir gerai struktūruotas metodas užtikrina, kad modelis būtų patikimas ir pritaikomas būsimiems aviacijos pramonės iššūkiams.

3.5.4. CHAT-GPT atnaujinimas

Šiame scenarijuje pateikiamas pažangus požiūris į nuotaikų analizę aviacijos srityje naudojant „OpenAI“ GPT-3.5 turbo modelį. Jis sistemingai apdoroja tekstinius duomenis iš paieškos sistemų („Google“ ir „Bing“) rezultatų, kad gautų reikšmingas, specialiai aviacijos kontekstui pritaikytas nuotaikų įžvalgas. Darbo eiga pradedama kuriant aplinką, į kurią importuojamos svarbiausios bibliotekos, įskaitant „OpenAI“, skirtą sąveikai su modeliu,

„Pandas“, skirtą duomenims tvarkyti, ir „TQDM“, skirtą pažangai stebėti. Inicijuojamas „OpenAI“ API raktas, kad būtų galima palaikyti ryšį su GPT modeliu.

Sentimentų analizės funkcija sukurta su aviacijai būdingomis instrukcijomis, kad modelis tiksliai interpretuotų šios specializuotos srities tekstą. Sistemos pranešime nustatomas kontekstas, nurodant modeliui veikti kaip aviacijai būdingam sentimentų analizatoriui. Naudotojo žinutėje pateikiama teksto įvestis ir prašoma pateikti neigiamų, neutralių ir teigiamų sentimentų balus struktūrizuotu formatu kartu su trumpu vertinimu. Toks pritaikytas metodas užtikrina, kad modelis pateikia aktualias ir konkrečiai sričiai pritaikytas įžvalgas.

Skriptas iteratyviai apdoroja duomenis, užtikrindamas, kad iš duomenų rinkinių „FullText“ arba „Snippet“ stulpelių būtų išskirtas tinkamas tekstas. Negaliojantys arba trūkstami teksto įrašai praleidžiami, kad būtų išsaugota analizės kokybė. Sentimentų rezultatai kartu su atitinkamomis nuorodomis saugomi naujame DataFrame, kad juos būtų galima toliau apdoroti.

Gavus nuotaikų rezultatus, jie suskirstomi į neigiamų, neutralių ir teigiamų nuotaikų skaitinius balus naudojant reguliarias išraiškas. Šis etapas yra labai svarbus, kad tekstinė GPT išvestis būtų transformuota į struktūrizuotą formatą, kurį galima analizuoti kiekybiškai. Sudėtinis balas apskaičiuojamas iš teigiamo balo atimant neigiamą balą ir gaunamas apibendrintas nuotaikų poliariškumo rodiklis.

Kad duomenys būtų paruošti analizei, scenarijuje pašalinamos pasikartojančios eilutės pagal lauką „Link“ ir pridedamas šaltinio identifikatorius, kad būtų galima atskirti, ar duomenys gauti iš „Google“, ar iš „Bing“. Taip užtikrinama, kad duomenų rinkinys būtų ir švarus, ir informatyvus, o visi sentimentų balai būtų suderinti su atitinkamais paieškos rezultatais.

Tada apdoroti nuotaikų duomenys vėl sujungiami su pradiniais duomenų rinkiniais, praturtinant juos nuotaikų įžvalgomis. Eilutėms, kuriose trūksta sentimentų duomenų, siekiant išlaikyti duomenų nuoseklumą, priskiriamos numatytoji nulinė reikšmė. Šis sujungtas duomenų rinkinys sujungia originalų tekstą su GPT sukurtais nuotaikų įvertinimais, todėl galima atlikti išsamią analizę.

Tarpiniai rezultatai įrašomi į atskirus „Google“ ir „Bing“ duomenų CSV failus, taip užtikrinant, kad pirminiai sentimentų rezultatai būtų išsaugoti ir būtų galima jais naudotis. Taip pat išsaugomi galutiniai apdoroti duomenų rinkiniai, kuriuose sentimentų duomenys sujungiami su originaliais paieškos rezultatais, todėl galima susidaryti išsamų praturtintų duomenų vaizdą.

Šis metodas turi keletą privalumų. Pritaikant GPT modelį su aviacijai būdingais nurodymais, užtikrinama, kad atliekant nuotaikų analizę būtų atsižvelgta į šios specializuotos srities niuansus. Struktūrizuotas nuotaikų balų pateikimas kartu su sudėtiniais balais palengvina tikslias ir interpretuojamas įžvalgas. Sentimentų duomenis integravus su originaliais paieškos rezultatais, išsaugomas prasmingai analizei būtinas kontekstas.

Apskritai, šis įgyvendinimas rodo sudėtingą GPT taikymą konkrečios srities sentimentų analizei. Jis veiksmingai mažina atotrūkį tarp bendrosios paskirties dirbtinio intelekto modelių ir specializuotų pramonės sektoriaus reikalavimų, todėl įžvalgos yra tikslios ir pritaikomos. Šį praturtintą duomenų rinkinį galima pritaikyti rizikos vertinimui, tendencijų stebėsenai ir sprendimų priėmimo procesams aviacijos sektoriuje.

3.5.5. CUDA taikymas

A100 GPU, naudojantis NVIDIA CUDA (*Compute Unified Device Architecture*) technologija, yra viena pažangiausių aparatinių priemonių, skirtų sudėtingoms skaičiavimams atlikti, įskaitant gilųjį mokymąsi, duomenų analizę ir dirbtinio intelekto (DI) modelių mokymą. Tai GPU, priklausantis NVIDIA „Ampere“ architektūrai, optimizuotas didelės apimties skaičiavimams ir lygiagrečioms užduotims, kuriose reikia didelės atminties pralaidumo ir skaičiavimo galios. Jis palaiko CUDA, kuri leidžia vykdyti didelio našumo ML bibliotekas, tokias kaip „PyTorch“ ir „TensorFlow“, maksimaliai išnaudojant aparatūros galimybes.

„Google Colab“ siūlo galimybę naudoti įvairias aparatūros spartintuvo parinktis, įskaitant CPU, T4 GPU, L4 GPU, TPU (Tensor Processing Units) ir A100 GPU. Kaip nurodyta dokumentacijoje, A100 GPU gali būti aktyvuojamas pasirenkant „Runtime > Change Runtime Type“ ir pažymint „A100 GPU“ skyriuje „Hardware accelerator“. Šis GPU išsiskiria iš kitų siūlomų variantų savo galia, kuri ypač aktuali mokant didelius ir sudėtingus ML modelius. Pasak oficialios dokumentacijos (Colaboratory: Frequently Asked Questions, 2024):

„Colab provides free access to GPUs and TPUs for accelerating compute-heavy operations. A100 GPUs offer the highest performance among the available options.“

Naudojant A100 GPU kartu su CUDA technologija, duomenų apdorojimas tampa labai efektyvus. CUDA leidžia vykdyti lygiagretinius skaičiavimus, todėl mokymo procesai, tokie kaip matricų daugyba ar grįžtamojo sklaidos algoritmo taikymas, vyksta daug greičiau (5 lentelė). Ši technologija taip pat leidžia išnaudoti didelę A100 GPU atmintį, kuri būtina didelio masto modeliams, tokiems kaip „RoBERTa“ ar BERT, efektyviai apdoroti.

5 lentelė

Sutaupytas vykdymo laikas naudojant CUDA

Modelio pav.	CPU laikas	A100 laikas	Skirtumas
VADERS	0.92 sek.	0.77 sek.	~0.15 sek.
roBERTa	~7-8 val.	44.56 min.	~7 val.
BERT	~5-6 val.	31.38 min	~4.5 - 5.5 val.
ChatGPT	-	-	N/A

Šaltinis: Sudaryta autoriaus

Naudojant A100 GPU, mokymo laikai skaičiuojami minutėmis, o ne valandomis. Pavyzdžiui:

- „RoBERTa“ modelis, kuris CPU aplinkoje užtruktų apie 7–8 valandas, su A100 GPU buvo išmokytas per 44,56 minutes.
- BERT modelio mokymas CPU truktų 5–6 valandas, tačiau su A100 GPU jis buvo atliktas per 31,38 minutes.
- Net ir maži sentimentų analizės modeliai, tokie kaip „VADERS“, pastebimai pagreitėjo – jų vykdymo laikas sumažėjo nuo 0.92 sekundžių iki 0.77 sekundžių.

Remiantis stebėtais laikais, kai naudojama A100 GPU, galima prognozuoti, kad CPU trukmė būtų apie 10 kartų ilgesnė. Tai rodo, jog CUDA technologijos nauda yra itin reikšminga:

- „RoBERTa“ modelis būtų mokomas apie 7–8 valandas vietoj 44,56 minutės.
- BERT modelis užtruktų 5–6 valandas vietoj 31,38 minutės.

„Google Colab“ platformoje siūlomi keli greitintuvai, skirti įvairiems skaičiavimo poreikiams:

- CPU: naudojamas pagrindinėms užduotims, tačiau gilusis mokymasis čia vyksta labai lėtai.
- T4 GPU: tinkamas mažesniems modeliams ar vidutinės apimties ML užduotims.
- L4 GPU: našesnis už T4, skirtas sudėtingesnėms ML užduotims.
- v2-8 TPU ir v5e-1 TPU: specializuotos „Tensor Processing Units“, naudojamos optimizuotoms ML bibliotekoms, tokioms kaip „TensorFlow“.

- A100 GPU: stipriausias pasirinkimas, užtikrinantis didžiausią našumą ir greitį, skirtas didelės apimties ML modelių mokymui.

A100 GPU, sujungtas su CUDA galimybėmis, žymiai paspartino šio tyrimo ML modelių mokymo procesą. Sutrumpėjęs skaičiavimo laikas leido efektyviai iteruoti modelių tobulinimą ir lengviau analizuoti aviacijos srities sentimentų analizės rezultatus. Be to, A100 GPU integravimas į „Google Colab“ demokratizuoja prieigą prie didelio našumo kompiuterijos ir leidžia pasaulio mokslininkams dirbti su pažangiausiais įrankiais neinvestuojant į brangią techninę įrangą.

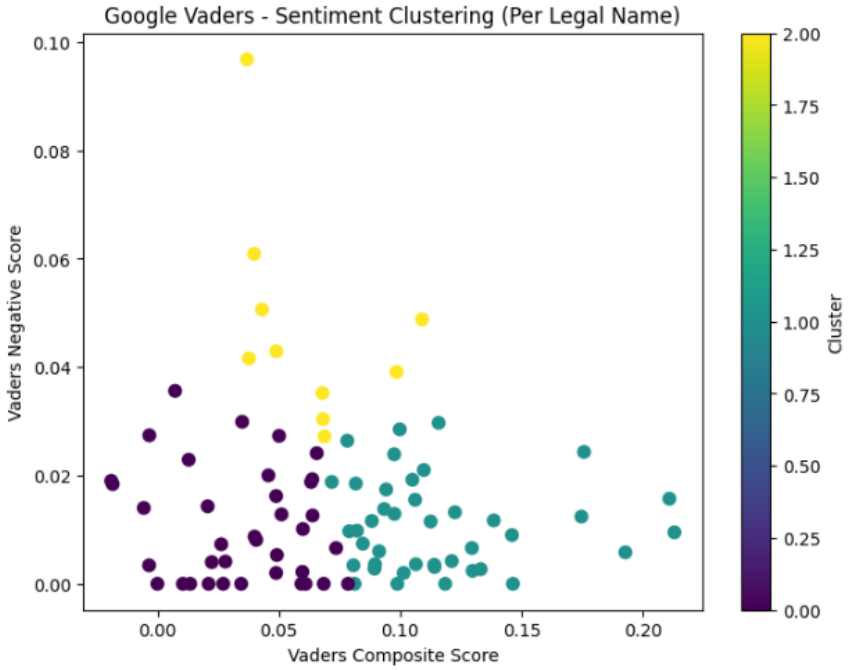
3.6. Rizikos lygio nustatymas naudojant k-klasterizavimo metodą

Rizikos lygio nustatymas yra labai svarbus nuotaikų analizės aspektas, kai ji taikoma tokiose jautriose srityse kaip atitikties stebėsena aviacijos sektoriuje. Šiame tyrime K-Means klasterizacija, plačiai paplitęs mašininio mokymosi algoritmas, skirtas nekontroliuojamam mokymuisi, buvo panaudotas nuotaikų vertinimams suskirstyti į atskirus rizikos lygius. Taikant šį metodą, nuotaikų duomenys iš įvairių šaltinių, įskaitant „Google“ ir „Bing“ VADER nuotaikų balus, buvo išanalizuoti ir sugrupuoti į klasterius, atspindinčius skirtingus rizikos lygius: aukštą, vidutinį ir žemą.

K-Means klasterizavimas veikia suskirstant duomenis į tam tikrą skaičių klasterių (šiuo atveju k = 3), remiantis klasterio vidinės dispersijos minimizavimu (Hartigan ir Wong, 1979). Šis metodas gerai tiko šiai problemai spręsti, nes juo galima apdoroti skaitinius požymius, pavyzdžiui, nuotaikų balus, ir užtikrinti aiškų segmentavimą pagal nuotaikų modelius. Toliau aprašoma, kaip K-Means klasterizacija buvo pritaikyta šiai problemai spręsti:

1. Požymių atranka: Pagrindiniais požymiais buvo pasirinkti tokie sentimentų balai, kaip neigiami (neg), teigiami (pos) ir sudėtiniai sentimentų balai (composite). Šie rodikliai buvo apibendrinti pagal kiekvieną subjektą (t. y. juridinį pavadinimą), kad atspindėtų jų bendrą nuotaikų pėdsaką.
2. Pirminis duomenų apdorojimas: Kad būtų užtikrintas požymių palyginamumas, sentimentų duomenys buvo normalizuoti naudojant „StandardScaler“. Šiuo veiksmu kiekvienas požymis buvo standartizuotas taip, kad jo vidurkis būtų lygus 0, o standartinis nuokrypis - 1. Taip išvengta šališkumo, atsirandančio dėl skirtingo reikšmių intervalo.
3. Klasterizavimo procesas: Taikytas K-Means algoritmas, kurio k=3, atitinkantis tris rizikos lygius: žemą, vidutinį ir aukštą. Klasterizavimas buvo pagrįstas Euklido atstumo tarp duomenų taškų ir jų klasterių centroidų minimizavimu.

4. Klasterių vertinimas ir rizikos žemėlapio sudarymas: Po klasterizavimo klasterių centroidai buvo išanalizuoti, siekiant nustatyti jų santykinę padėtį pagal sudėtinį nuotaikų balą. Klasteriai, kurių vidutinis sudėtinių nuotaikų balų vidurkis buvo priskirtas prie „mažos rizikos“, o klasteriai, kurių balų vidurkis buvo mažiausias, buvo pažymėti kaip „didelės rizikos“. Taip buvo užtikrinta, kad subjektams, pasižymintiems daugiausia neigiamomis nuotaikomis, būtų priskirti aukštesni rizikos lygiai.
5. Vizualizavimas ir aiškinimas: Klasterių pasiskirstymui vizualizuoti buvo sukurtos sklaidos diagramos. Pavyzdžiui, sudėtinis balas buvo nubraižytas pagal neigiamų nuotaikų balą, o spalvos žymėjo klasterius. Šios vizualizacijos leido suprasti pagrindinę sentimentų duomenų struktūrą ir padėjo patvirtinti klasterizavimo rezultatus.

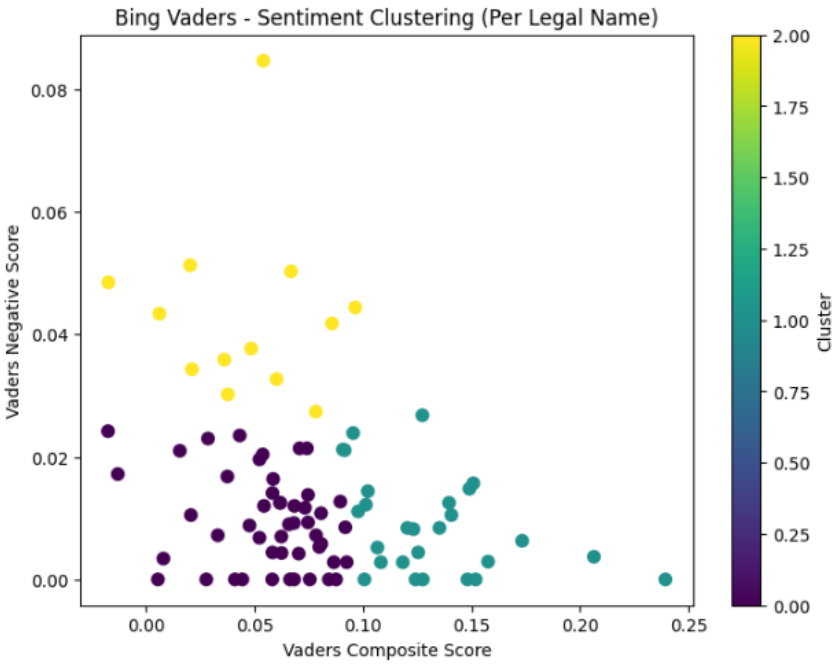


Šaltinis: Sudaryta autoriaus

17 pav. „Google Vaders“ – sentimentų grupavimas (pagal juridinį pavadinimą)

Google VADER sentimentų klasterizavimo diagramoje (17 pav.) išryškėja trys skirtingos spalvos žymimi klasteriai. X ašis rodo VADER sudėtinį balą, o y ašis - VADER neigiamą balą. Subjektai, kurių neigiami balai didesni, o sudėtiniai balai mažesni, buvo sugrupuoti į „Didelės rizikos“ klasterį, pavaizduotą violetine spalva. Ir atvirkščiai, subjektai, kurių sudėtiniai balai buvo didesni, o neigiami balai mažesni, pateko į „mažos rizikos“ klasterį, pavaizduotą arbatine spalva.

Pavyzdžiui, vienas subjektas, kurio sudėtinis balas buvo maždaug 0,05, o neigiamas balas - 0,08, buvo priskirtas „vidutinės rizikos“ grupei (geltonos spalvos klasteris). Tai atspindi teigiamų ir neigiamų vertinimų pusiausvyrą, rodančią vidutinio lygio susirūpinimą arba atitikties riziką.



Šaltinis: Sudaryta autoriaus

18 pav. „Bing Vaders“ – sentimentų grupavimas (pagal juridinį pavadinimą)

Bing VADER nuotaikų grupavimo diagramoje (18 pav.) stebimi panašūs grupavimo modeliai. Subjektai grupuojami pagal jų sudėtinių ir neigiamų balų sąveiką. Didelės rizikos klasteris apima subjektus, kurių neigiamų nuotaikų balai yra santykinai didesni, o sudėtiniai balai - mažesni, o tai rodo didelę riziką. Pavyzdžiui, subjektas, kurio sudėtinis balas yra 0,04, o neigiamas balas - 0,07, buvo priskirtas „didelės rizikos“ kategorijai. Tai rodo, kad klasteriai nuosekliai grupuojami įvairiuose duomenų rinkiniuose, o tai patvirtina metodo patikimumą.

Šiame tyrime rizikos nustatymui naudotas K-Means klasterizavimas rodo nekontroliuojamo mokymosi veiksmingumą atliekant nuotaikų analizę. Apibendrinus nuotaikų metrikas ir pritaikius patikimas klasterizavimo metodikas, rizikos lygių nustatymo procesas buvo automatizuotas ir standartizuotas. Šie rezultatai sudaro pagrindą integruoti nuotaikų analizę į platesnes atitikties stebėsenos sistemas, suteikiant organizacijoms galingą priemonę sistemingai vertinti ir mažinti riziką. Ateityje būtų galima ištirti pažangių klasterizavimo metodų arba hibridinių modelių naudojimą, siekiant dar labiau padidinti rizikos nustatymo galimybes.

3.7. Gautų rezultatų analizė

Naudotiems NLP modeliams nustačius rizikos lygius kiekvienam tyrime dalyvavusiam klientui, svarbu gautus rezultatus validuoti taip patikrinant modelio tikslumą bei aktualumą. Analizė apima tiek kiekybinius aspektus, tokius kaip tikslumo rodikliai, tiek kokybines įžvalgas, gautas lyginant pradinių ir naujų modelių veiklos rezultatus. Rezultatai pateikiami kontekstualiai, ypatingą dėmesį skiriant internetinės reputacijos vertinimui ir rizikos lygių klasifikavimui.

Pagrindinė metrika, naudojama modelio veikimui įvertinti, yra tikslumas. Toliau pateiktoje lentelėje (lent. 6) apibendrinti naudotų NLP modelių, suskirstytų į originalią ir atnaujintą (apmokytą aviacijos žodynu) versijas, atitikmens procentai su realiais klientų rizikos lygiais.

6 lentelė

NLP modelių tikslumo palyginimas

Modelio pav.	**Originalaus modelio tikslumas (%)**	**Atnaujinto modelio tikslumas (%)**	**Pokytis(%)**
Google GPT	38.10	52.38	+14.29
Bing GPT	38.10	42.86	+4.76
Google BERT	52.38	42.86	-9.52
Bing BERT	38.10	42.86	+4.76
Google RoBERTa	19.05	42.86	+23.81
Bing RoBERTa	23.81	35.00	+11.19
Google VADERs	42.86	52.38	+9.52
Bing VADERs	33.33	38.10	+4.76

Šaltinis: Sudaryta autoriaus

6-oje lentelėje pateikiama originalių ir atnaujintų mašininio mokymosi modelių tikslumo procentinių dalių lyginamoji analizė vertinant rizikos lygius KYC procese. Joje išryškinamas kiekvieno modelio našumo pagerėjimas arba pablogėjimas (išreikštas procentiniu pokyčiu). Pastebėtina, kad geriausia rezultatą demonstruoja „Google GPT“ modelis, kurio tikslumas padidėjo 14,29 % lyginant su originaliu modeliu ir dabar siekia 52.38 proc. bendrą tikslumą, kas

kartu su Google VADERS atnaujintu modeliu dalinasi geriausią pasiektą tikslumo procentą. Po to seka „Google RoBERTa“ atitikimo procentą padidinęs 23,81 % iki 42.86 proc. Tuo tarpu „Google BERT“ modelio tikslumas sumažėjo -9,52 %. „Bing“ pagrindu sukurtų modelių tikslumas iš esmės pagerėjo nedaug, o ‚Bing RoBERTa‘ tikslumas padidėjo +11,19 %. Apskritai lentelėje matyti įvairūs modelių našumo patobulinimai, kai kurie atnaujintieji modeliai gerokai lenkia savo pirminius modelius, ypač konkrečiose konfigūracijose, pavyzdžiui, „Google RoBERTa“ ir „Google GPT“. Tai rodo, kad galima patobulinti iš anksto parengtus modelius ir padidinti rizikos vertinimo užduočių tikslumą.Šiuos gautus rezultatus galima skaidyti ir pridedant rizikos lygio dimensiją į analizę (7 lentelė):

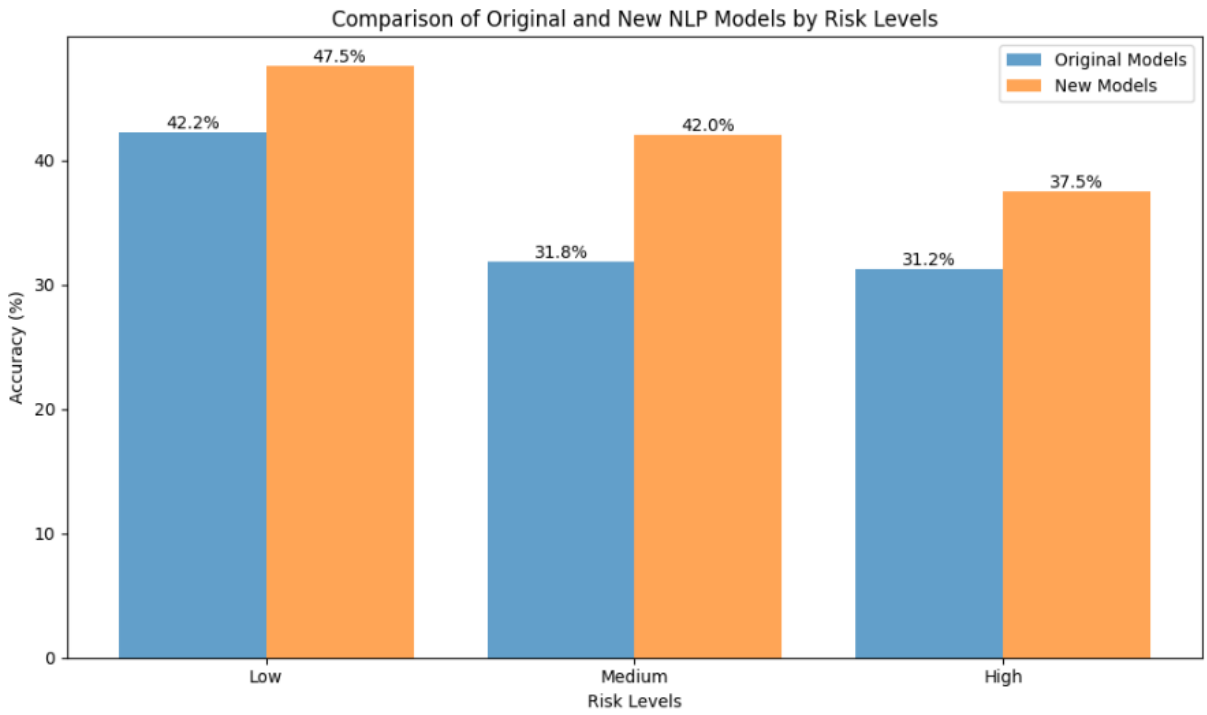
7 lentelė

NLP modelių tikslumo palyginimas išskiriant rizikos lygius

Modelio pav.	Originalaus modelio žemos rizikos atitikmuo (%)	Atnaujinto modelio žemos rizikos atitikmuo (%)	Originalaus modelio vidutinės rizikos atitikmuo (%)	Atnaujinto modelio vidutinės rizikos atitikmuo (%)	Originalaus modelio aukštos rizikos atitikmuo (%)	Atnaujinto modelio aukštos rizikos atitikmuo (%)
Google GPT	62.5	75.0	27.27	45.45	0.0	0.0
Bing GPT	12.5	75.0	54.55	18.18	50.0	50.0
Google BERT	62.5	25.0	45.45	54.55	50.0	50.0
Bing BERT	25.0	25.0	45.45	54.55	50.0	50.0
Google RoBERTa	37.5	37.5	9.09	45.45	0.0	50.0
Bing RoBERTa	25.0	42.9	9.09	27.27	100.0	50.0
Google VADERs	50.0	50.0	45.45	63.64	0.0	0.0
Bing VADERs	62.5	50.0	18.18	27.27	0.0	50.0

Šaltinis: Sudaryta autoriaus

Lentelėje pateikiama pirminio ir naujojo NLP modelių tikslumo lyginamoji analizė pagal tris rizikos lygius: Žemas, vidutinis ir aukštas. Daugumos modelių „žemos“ rizikos lygio tikslumas naujosiose versijose gerokai pagerėjo: „Google“ GPT padidėjo nuo 62,5 % iki 75 %, o „Bing“ GPT - nuo 12,5 % iki 50 %. Panašiai „vidutinės“ rizikos lygio tikslumas pastebimai pagerėjo tokiuose modeliuose kaip „Google VADER“ - nuo 45,45 % iki 63,64 %. Priešingai, kai kurių modelių, pavyzdžiui, „Google BERT“, „vidutinio“ tikslumo (45,45 %) rezultatai originaliose ir naujose versijose yra vienodi. „Aukšto“ rizikos lygio atveju ypač akivaizdžiai pagerėjo tokie modeliai kaip ‚Google RoBERTa‘ (nuo 0 % iki 42,86 %) ir ‚Bing RoBERTa‘ (nuo 0 % iki 50 %). Apskritai naujieji modeliai pasižymi geresniais arba panašiais rezultatais, ypač „mažos“ ir „didelės“ rizikos lygių atveju, ir tai rodo, kad jie patobulinti ir veiksmingi vertinant riziką. Dar labiau apibendrinant duomenis galima matyti aiški teigiama tendencija (19 pav.):



Šaltinis: Sudaryta autoriaus

19 pav. Pradinio ir naujojo NLP modelių palyginimas pagal rizikos lygius

Stulpelinėje diagramoje pavaizduotas pirminio ir naujojo NLP modelių lyginamasis tikslumas trimis rizikos lygiais: Žemas, vidutinis ir aukštas. Naujieji modeliai buvo pranašesni už originaliuosius visose rizikos kategorijose. Mažos rizikos lygių atveju tikslumas padidėjo nuo 42,2 % (originalus) iki 47,5 % (naujas), o tai rodo žymų pagerėjimą. Panašiai ir vidutinės rizikos kategorijoje naujųjų modelių tikslumas padidėjo iki 42,0 %, palyginti su 31,8 % pirminių modelių tikslumu. Galiausiai, didelės rizikos lygio atveju naujieji modeliai pasiekė 37,5 % tikslumą,

viršydami 31,2 % pirminių modelių tikslumą. Šie rezultatai rodo, kad naujieji NLP modeliai teikia nuoseklesnes ir tikslesnes prognozes visuose rizikos lygiuose, ypač mažos ir vidutinės rizikos kategorijose, o tai rodo didesnį jų veiksmingumą atliekant rizikos klasifikavimo užduotis.

Palyginus gautų rezultatų „Google“ ir „Bing“ pagrįstų NLP modelių vidutinį tikslumą, matyti, kad „Google“ modeliai nuolat lenkia „Bing“ modelius: vidutinis tikslumas yra 42,86 %, o „Bing“ - 36,52 %. Šis skirtumas pabrėžia „Google“ pagrįstų modelių veiksmingumą ir tikslumą vertinant rizikos lygį ir reputaciją. Geresnį „Google“ modelių veikimą galima paaiškinti jų pažangia iš anksto apmokyta architektūra ir optimizavimo metodais, todėl jie geriau tinka niuansuotiems KYC ir deramo patikrinimo procesų reikalavimams aviacijos sektoriuje.

Palyginus pradinių sujungtų modelių ir naujai sugrupuotų modelių tikslumą, pastebimas nedidelis našumo sumažėjimas. Pirminių sujungtų modelių tikslumas siekė 56,82 %, o tai rodo jų veiksmingumą tiksliai nustatant ir klasifikuojant rizikos lygius. Tuo tarpu galutinis naujų sugrupuotų modelių tikslumas sumažėjo iki 52,38 %, t. y. 4,44 procentinio punkto. Šį sumažėjimą galima paaiškinti duomenų grupavimo strategijų pokyčiais, modelių optimizavimu arba sunkumais prisitaikant prie naujų duomenų pasiskirstymų. Nors naujieji modeliai vis dar suteikia vertingų įžvalgų, rezultatai rodo, kaip svarbu tobulinti metodikas, kad tikslumas nuosekliai didėtų.

IŠVADOS

1. Mašininiu mokymusi paremtos reputacijos vertinimo metodikos gerokai patobulina tradicinius, žmogiškaisiais ištekliais paremtus procesus, nes leidžia atlikti dinamišką ir didelės apimties duomenų analizę, o tai labai svarbu tokiame sudėtingame ir griežtai reguliuojamame sektoriuje kaip aviacija.
2. Integruojant natūralios kalbos apdorojimo (NLP) modelius ir pažangius klasterizavimo metodus, atliekant tyrimą pavyko automatizuotai klasifikuoti rizikos lygius į „žemą“, „vidutinį“ ir „aukštą“. Iš jų „Google“ pagrįsti modeliai, ypač patobulinti papildomu aviacijos sektoriaus kontekstu, savo tikslumu nuolat lenkė „Bing“ pagrįstus analogiškus modelius.
3. K klasterizavimo (angl. *K-Clustering*) metodika pasirodė esanti veiksminga verčiant nuotaikų analizės balus į praktiškai pritaikomus rizikos vertinimus, o tai yra labai svarbus žingsnis atitikties ir deramo patikrinimo procesuose.
4. Po papildomo modelių mokymo „žemos“ rizikos klasifikavimo tikslumas padidėjo nuo 42,2 % iki 47,5 %, „vidutinės“ rizikos – nuo 31,8 % iki 42,0 %, o „aukštos“ rizikos – nuo 31,2 % iki 37,5 %. Tai rodo, kad visi rizikos lygiai buvo geriau atpažįstami po papildomo aviacijos duomenų įtraukimo į mokymosi procesą.
5. „Google RoBERTa“ modelis pasižymėjo reikšmingiausiu atitikimo tikslumo pagerėjimu – nuo 19,05 % iki 42,86 %, kas sudaro +23,81 % pokytį. „Bing RoBERTa“ taip pat pastebimai patobulėjo (+11,19 %), o kiti modeliai, kaip „Google GPT“ ir „Google VADERs“, taip pat pasiekė reikšmingų rezultatų. Šie skaičiai aiškiai pabrėžia, kad NLP modelių pritaikymas konkrečiam sektoriui, tokiam kaip aviacija, padeda pagerinti jų efektyvumą.
6. Galutiniai rezultatai parodė trūkumus. Bendras atnaujintų sugrupuotų modelių tikslumas (52,38 %) buvo mažesnis nei pirminių modelių (56,82 %), o tai rodo, kad tobulinant tam tikras sritis atsirado kompromisų. Šis neatitikimas pabrėžia būtinybę toliau optimizuoti ML algoritmus ir modelių integravimo procesus. Be to, NLP modeliai, nors ir buvo patobulinti, pasižymėjo skirtingu efektyvumo lygiu, ypač atskiriant vidutinės ir didelės rizikos lygius, o tai rodo, kad reikalingi labiau specializuoti konkrečios srities mokymo duomenys.

LITERATŪROS SĄRAŠAS

1. *About EASA*. (2024). From easa.europa.eu: https://www.easa.europa.eu/en/the-agency/faqs/about-easa
2. *About FAA*. (2024). From FAA.gov: https://www.faa.gov/about
3. *About ICAO*. (2024). From ICAO.int: https://www.icao.int/about-icao/Pages/default.aspx
4. *About Us - Birdeye*. (2024). From Birdeye.com: https://birdeye.com/aboutus/
5. Alqwadri, A., Azzeh, M., & Almasalha, F. (2021). Application of Machine Learning for Online Reputation. *International Journal of Automation and Computing*.
6. Alzubi, J., Nayyar, A., & Kumar, A. (2018). Machine Learning from Theory to Algorithms: An Overview. *Journal of Physics: Conference Series*.
7. Amadeus. (2014). *Amadeus Fraud Management for Airlines: Fraud prevention customised for the travel industry.* From Amadeus.com: https://amadeus.com/documents/en/travel-industry/sales-sheet/amadeus-fraud-management-for-airlines.pdf
8. Aras, T. M., & Güvensan, M. A. (2021). Challenges and Key Points for Fraud Detection in Aviation. Istanbul, Turkey.
9. *Bank of Lithuania: KYC information collected by banks helps to strengthen the country's resilience to threats of money laundering and circumvention of sanctions*. (2023). From lb.lt: https://www.lb.lt/en/news/bank-of-lithuania-kyc-information-collected-by-banks-helps-to-strengthen-the-country-s-resilience-to-threats-of-money-laundering-and-circumvention-of-sanctions
10. Biersteker, T., & Portela, C. (2015). *EU sanctions in context: three types.* European Union Institute for Security Studies (EUISS).
11. Biliri, E., Pertselakis, M., Phinikettos, M., Zacharias, M., Lampathaki, F., & Alexandrou, D. (2019). Designing a Trusted Data Brokerage. *Collaborative Networks and Digital Transformation.* Turin: Springer, Cham.
12. *Bing Web Search API*. (2024). From Microsoft.com: https://www.microsoft.com/en-us/bing/apis/bing-web-search-api
13. *BrandYourself.com: Boosting Your Personal Brand Online*. (2012). From Foxbusiness.com: https://www.foxbusiness.com/video/1437231505001
14. *Business listings - Reputation*. (2024). From Reputation.com: https://reputation.com/products/business-listings/

15. Chen, T.-H. (2020). Do you know your customer? Bank risk assessment based on machine. *Applied Soft Computing Journal*.
16. Chowdhary, K. R. (2020). Fundamentals of Artificial Intelligence. Jodhpur, Rajasthan, India.
17. Ciccozzi, F., Addazi, L., Asadollah, S. A., Lisper, B., Masud, A. N., & Mubeen, S. (2022). A Comprehensive Exploration of Languages for Parallel Computing. *ACM Computing Surveys (CSUR), Volume 55*.
18. *Colaboratory: Frequently Asked Questions*. (2024). From research.google.com: https://research.google.com/colaboratory/faq.html
19. *Consolidated list of persons, groups and entities subject to eu financial sanctions*. (2024). From ofac.treasury.gov: https://ofac.treasury.gov/ https://data.europa.eu/data/datasets/consolidated-list-of-persons-groups-and-entities-subject-to-eu-financial-sanctions?locale=en
20. Corbet, S., Nicolau, J. L., & Oxley, L. (2023). Spillover Effects of Sanctions on Airlines. *Journal of Travel Research*.
21. Cortright, D., & Lopez, G. A. (2018). Economic sanctions: Panacea or peacebuilding in a post-cold war world? Abingdon, UK.
22. *Danske Bank Pleads Guilty to Fraud on U.S. Banks in Multi-Billion Dollar Scheme to Access the U.S. Financial System*. (2022). From justice.gov: https://www.justice.gov/opa/pr/danske-bank-pleads-guilty-fraud-us-banks-multi-billion-dollar-scheme-access-us-financial
23. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for.
24. Dogucu, M., & Çetinkaya-Rundel, M. (2020). Web Scraping in the Statistics and Data Science. *Journal of Statistics and Data Science Education*.
25. *Endorsed Regulatory Fines And Enforcement Actions Relating To CDD/KYC Failures*. (2024). From Financialcrimeacademy.org: https://financialcrimeacademy.org/fines-and-enforcement-actions-relating-to-cdd-kyc-failures/
26. *Features - Repbot*. (2024). From Repbot.ai: https://repbot.ai/features
27. Feldman, A. (2024). *Sky-High Stakes: Combating Cyber Fraud in the Aviation Industry.* From BlueVoyant.com: https://www.bluevoyant.com/blog/combating-cyber-fraud-in-the-aviation-industry
28. Fitria, T. N. (2023). Artificial intelligence (AI) technology in OpenAI ChatGPT application: A review of ChatGPT in writing English essay. *Journal of English Language Teaching*.
29. Fuchs, K. (2023). Exploring the opportunities and challenges of NLP models in higher education: is Chat GPT a blessing or a curse? Phuket, Thailand.

30. *Google APIs Explorer*. (2024). From developers.google.com: https://developers.google.com/apis-explorer
31. Haque, M. U., Dharmadasa, I., Sworna, Z. T., Rajapakse, R. N., & Ahmad, H. (2022). I think this is the most disruptive technology": Exploring Sentiments of ChatGPT Early Adopters using Twitter Data.
32. Hartigan, J. A., & Wong, M. A. (1979). Algorithm AS 136: A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society*.
33. Hentosh, L., Tsikalo, Y., Kustra, N., & Kutucu, H. (2023). ML-based Approach for Credit Risk Assessment Using. Lviv, Ukraine.
34. *Home - Grade.us*. (2024). From Grade.us: https://www.grade.us/home/
35. *HSBC Holdings Plc. and HSBC Bank USA N.A. Admit to Anti-Money Laundering and Sanctions Violations, Forfeit $1.256 Billion in Deferred Prosecution Agreement*. (2012). From justice.gov: https://www.justice.gov/opa/pr/hsbc-holdings-plc-and-hsbc-bank-usa-na-admit-anti-money-laundering-and-sanctions-violations
36. Hutto, C., & Gilbert, E. (2014). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. *Proceedings of the Eighth International AAAI Conference on Weblogs and Social Media.* Atlanta: ICWSM.
37. IATA. (2020). *Fraud in the airline industry: why carriers need to think of themselves as crimefighters.* IATA.
38. Janiesch, C., Zschech, P., & Heinrich, K. (2021). Machine learning and deep learning. *Electronic Markets: The International Journal on Networked Business*.
39. Khder, M. A. (2021). Web Scraping or Web Crawling: State of Art Techniques, Approaches and Application. *International Journal of Advances in Soft Computing and its Applications*.
40. Koroteev, M. (2021). BERT: A Review of Applications in Natural Language Processing and Understanding. Moscow, Russia.
41. Krotov, V., & Johnson, L. (2020). Legality and Ethics of Web Scraping. *Communications of the Association for Information Systems*.
42. MacQueen, J. (1967). Some Methods for Classification and Analysis of Multivariate Observations .
43. Mahesh, B. (2019). Machine Learning Algorithms -A Review. *International Journal of Science and Research (IJSR).*
44. Makortoff, K. (2019). *Standard Chartered fined $1.1bn for money-laundering and sanctions breaches*. From theguardian.com: https://www.theguardian.com/business/2019/apr/09/standard-chartered-fined-money-laundering-sanctions-breaches

45. Mansoor, N., Antora, K. F., Priyata, D., Arman, T. A., Manaf, A. A., & Zareei, A. M. (2023). A Review of Blockchain Approaches for KYC. *IEEE Access*.
46. *Market share of leading desktop search engines worldwide from January 2015 to January 2024*. (2024). From Statista.com: https://www.statista.com/statistics/216573/worldwide-market-share-of-search-engines/
47. *Media Intelligence - Meltwater*. (2024). From Meltwater.com: https://www.meltwater.com/en/suite/media-intelligence
48. Metz, C. A., Goli, M., & Drechsler, R. (2021). Pick the Right Edge Device: Towards Power and Performance Estimation of CUDA-based CNNs on GPGPUs. Bremen, Germany.
49. Miliauskas, M. (2024). *AML Fines: Recent Most Famous Cases*. From amlyze.com: https://amlyze.com/aml-fines/
50. Min, B., Ross, H., Sulem, E., Pouran, A., Veyseh, B., Nguyen, T. H., . . . Roth, D. (2023). Recent Advances in Natural Language Processing via Large Pre-trained Language Models: A Survey. *ACM Computing Surveys*.
51. *Monitor - Mention*. (2024). From Mention.com: https://mention.com/en/
52. Müller, M., Salathé, M., & Kummervold, E. P. (2023). COVID-Twitter-BERT: A natural language processing model to analyse COVID-19 content on Twitter. *Frontiers in Artificial Intelligence*.
53. Nadkarni, P. M., Ohno-Machado, L., & Chapman, W. W. (2011). Natural language processing: an introduction. *Journal of the American Medical Informatics Association*.
54. Ofer, D., Brandes, N., & Linial, M. (2021). The language of proteins: NLP, machine learning & protein sequences. *Computational and Structural Biotechnology Journal 19*.
55. *Online reputation grader - Podium*. (2024). From Podium.com: https://www.podium.com/online-reputation-grader/
56. *Online reputation management - Brand24*. (2024). From Brand24.com: https://brand24.com/use-case/online-reputation-management/
57. Osborne, H. (2018). *Malta's Pilatus Bank has European licence withdrawn*. From theguardian.com: https://www.theguardian.com/world/2018/nov/05/pilatus-bank-malta-european-banking-licence-withdrawn
58. Perlman, A. (2024). The Implications of ChatGPT For Legal Services and Society.
59. Persson, E. (2019). Evaluating tools and techniques for web scraping. Stockholm.
60. Petrik, M. (2017). *Machine Learning: Introduction to Machine Learning.* From www.cs.unh.edu: https://www.cs.unh.edu/~mpetrik/teaching/intro_ml_17_files/class1.pdf
61. Poudel, A., & Weninger, T. (2024). Navigating the Post-API Dilemma: Search Engine Results Pages Present a Biased View of Social Media Data.

62. Rajput, V. U. (2013). Research on Know Your Customer (KYC). *International Journal of Scientific and Research Publications*.
63. *Reputation management - Yext*. (2024). From Yext.com: https://www.yext.com/solutions/marketing/reputation-management
64. Seyfi, S., & Hall, C. M. (2020). Sanctions and tourism: effects, complexities and research. *An International Journal of Tourism Space, Place and Environment*.
65. Sinha, P., & Kaul, A. (2018). Decentralized KYC System. *International Research Journal of Engineering and Technology (IRJET)*.
66. Skirnevskiy, I. P., Pustovit, A. V., & Abdrashitova, M. O. (2017). Digital image processing using parallel computing based on CUDA technology. *Journal of Physics: Conference series*.
67. Stankevičiūtė, G. (2023). *KYC Onboarding: How to Achieve KYC/AML Compliance?* From Idenfy.com: https://www.idenfy.com/blog/kyc-onboarding/
68. Tarunesh, I., Aditya, S., & Choudhury, M. (2021). Trusting RoBERTa over BERT: Insights from CheckListing the Natural Language Inference Task.
69. *The Most Sanctioned Countries*. (2024). From forbes.ge: https://forbes.ge/en/the-most-sanctioned-countries/
70. Turksen, U., Benson, V., & Adamyk, B. (2024). Legal implications of automated suspicious transaction monitoring: enhancing integrity of AI. *Journal of Banking Regulation*.
71. *Understanding the Steps of a "Know Your Customer" Process*. (2024). From Dowjones.com: https://www.dowjones.com/professional/risk/glossary/know-your-customer/
72. *United Nations Security Council Consolidated List*. (2024). From main.un.org: https://main.un.org/securitycouncil/en/content/un-sc-consolidated-list
73. *What is NLP and how it is implemented in our lives*. (2024). From Amazinum.com: https://amazinum.com/insights/what-is-nlp-and-how-it-is-implemented-in-our-lives/
74. *What is the Bing Web Search API?* (2022). From learn.microsoft.com: https://learn.microsoft.com/en-us/bing/search-apis/bing-web-search/overview
75. Xu, Y., Lee, H., Chen, D., Hechtman, B., Huang, Y., Joshi, R., . . . Chen, Z. (2021). GSPMD: General and Scalable Parallelization for ML.
76. Yin, Y., & Habernal, I. (2022). Privacy-Preserving Models for Legal Natural Language Processing. Abu Dhabi, UAE.
77. Zhao, B. (2022). Web Scraping. *Encyclopedia of Big Data*.
78. Zhou, Z.-H. (2022). Open-environment machine learning. *National Science Review*.

PRIEDAI

1 PRIEDAS

```
# -*- coding: utf-8 -*-
"""MTD1222.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1bZkNCEyla8uQSTHojoc-bV7U0k32yumI
"""

!pip install pandas
!pip install beautifulsoup4
!pip install requests
!pip install selenium
!pip install tensorflow
!pip install torch
!pip install --upgrade tensorflow
!pip install - U spacy
!pip install - U spacy-lookups-data
!pip install transformers
!pip install nltk
!pip install matplotlib
!pip install seaborn
!pip install --upgrade --user torch transformers
!pip install transformers[torch]
!pip install accelerate -U

import pandas as pd
from bs4 import BeautifulSoup
import requests
import tensorflow as tf
import warnings
import tensorflow as tf
```

```
from transformers import AutoTokenizer
from transformers import AutoModelForSequenceClassification
from scipy.special import softmax
import os
from tqdm import tqdm  # For progress tracking
import csv  # Import csv module for quoting options

from google.colab import drive
drive.mount('/content/drive')

"""# 1. KYC data FLT"""

# Full file path
file_path = '/content/drive/My Drive/MTD/KYC data/MTD_Companies.csv'

# Load the file
df_kyc_data = pd.read_csv(file_path)

# Drop rows with missing values in critical columns
df_kyc_data.dropna(subset=['id', 'legal_name', 'risk_level', 'updated_at'], inplace=True)

# Ensure the 'id' column is numeric
df_kyc_data['id'] = pd.to_numeric(df_kyc_data['id'], errors='coerce').astype(int)

# Ensure values in 'risk_level' are consistent
valid_risk_levels = ['Low', 'Medium', 'High']
df_kyc_data = df_kyc_data[df_kyc_data['risk_level'].isin(valid_risk_levels)]
df_kyc_data['risk_level'] = df_kyc_data['risk_level'].str.title()

# Convert 'updated_at' to datetime
df_kyc_data['updated_at'] = pd.to_datetime(df_kyc_data['updated_at'], errors='coerce')

# Standardize 'legal_name'
df_kyc_data['legal_name'] = df_kyc_data['legal_name'].str.strip().str.title()
```

```
# Save the cleaned data
df_kyc_data.to_csv('cleaned_kyc_data_with_id.csv', index=False)

# Display a preview
print(f"Number of rows after cleaning: {len(df_kyc_data)}")

# Creating a new DataFrame with only 'id' and 'legal_name' columns, removing duplicates,
# and limiting it to the top 10 rows
df_companies = df_kyc_data[['id', 'legal_name']].drop_duplicates().head(100)
df_companies

# Limiting DataFrame
#df_companies = df_companies.head(5)

# Display the limited DataFrame
#print(df_companies)

"""# 2. Web scraping

## 2.1. Google Custom Search JSON API
"""

google_api_key = ''
cse_id = ''
# https://console.cloud.google.com/apis/credentials?project=webscrape-405719
# https://programmablesearchengine.google.com/controlpanel/overview?cx=e12d601fb17f345c0

# Function to perform a Google search
def google_search(legal_name, google_api_key, cse_id, **kwargs):
    query = {
        'q': legal_name,
        'key': google_api_key,
        'cx': cse_id,
        'num': 10  # Limit to the first 10 results
    }
```

```
    query.update(kwargs)
    response = requests.get('https://www.googleapis.com/customsearch/v1', params=query)
    return response.json()

# Function to scrape the full text from a URL
def get_full_text(url):
    try:
        response = requests.get(url, timeout=10)
        if response.status_code == 200:
            soup = BeautifulSoup(response.text, 'html.parser')
            paragraphs = soup.find_all('p')  # Find all paragraph tags
            all_text = ' '.join(para.get_text(separator=' ', strip=True) for para in paragraphs)
            return all_text if all_text.strip() else "None"  # Return "None" if all_text is empty
    except Exception:
        return "None"  # Return "None" in case of an exception

# Initialize a list to store the scraped data
scraped_data = []

# Use tqdm for tracking progress
for index, row in tqdm(df_companies.iterrows(), total=len(df_companies), desc="Processing Companies"):
    legal_name_to_search = row['legal_name']
    company_id = row['id']

    # Perform Google search
    results = google_search(legal_name_to_search, google_api_key, cse_id)

    # Check if results exist
    if 'items' in results:
        for item in results['items']:
            full_text = get_full_text(item['link'])  # Fetch the full text from the link
            row_data = {
                'id': company_id,
                'legal_name': legal_name_to_search,
```

```
                'Title': item['title'],
                'Link': item['link'],
                'Snippet': item['snippet'],  # Include the snippet
                'FullText': full_text
            }
            # Append to scraped_data
            scraped_data.append(row_data)
    else:
        print(f"No results found for {legal_name_to_search}.")

# Convert the scraped data to a DataFrame
if scraped_data:
    google_data = pd.DataFrame(scraped_data)

    # Ensure the output directory exists
    output_dir = '/content/drive/My Drive/MTD/KYC data/Output'
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    # Save the results to a CSV file with escapechar
    output_path = os.path.join(output_dir, 'google_search_results.csv')
    google_data.to_csv(output_path, index=False, quoting=csv.QUOTE_ALL, escapechar='\\')

    print(f"Data collection complete. Results saved to '{output_path}'.")
else:
    print("No data was collected. Please check your API settings or input data.")

"""## 2.2. Bing search API"""

bing_api_key = ''

# Function to perform a Bing search
def bing_search(legal_name, bing_api_key, **kwargs):
    print(f"Searching for {legal_name}")  # Debugging log
    headers = {"Ocp-Apim-Subscription-Key": bing_api_key}
```

```
    params = {"q": legal_name, "textDecorations": True, "textFormat": "HTML"}
    params.update(kwargs)
    response = requests.get("https://api.bing.microsoft.com/v7.0/search", headers=headers, params=params)
    print(f"Response status code: {response.status_code}")  # Check response status
    if response.status_code == 200:
        return response.json()
    else:
        print(f"Error fetching Bing results: {response.text}")
        return {}

# Function to scrape the full text from a URL
def get_full_text(url):
    print(f"Fetching full text from {url}")  # Debugging log
    try:
        response = requests.get(url, timeout=10)  # Added timeout
        if response.status_code == 200:
            soup = BeautifulSoup(response.text, 'html.parser')
            paragraphs = soup.find_all('p')
            all_text = ' '.join(para.get_text(separator=' ', strip=True) for para in paragraphs)
            return all_text if all_text.strip() else "None"
    except Exception as e:
        print(f"Error fetching full text: {e}")  # Log any errors
        return "None"

# Initialize a list to store the scraped data
scraped_data = []

# Use tqdm for tracking progress
for index, row in tqdm(df_companies.iterrows(), total=len(df_companies), desc="Processing Companies"):
    legal_name_to_search = row['legal_name']
    company_id = row['id']

    # Perform Bing search
```

```
    results = bing_search(legal_name_to_search, bing_api_key)

    # Check if results exist
    if 'webPages' in results and 'value' in results['webPages']:
        for item in results['webPages']['value']:
            full_text = get_full_text(item['url'])
            clean_snippet = BeautifulSoup(item['snippet'], 'html.parser').get_text()
            row_data = {
                'id': company_id,
                'legal_name': legal_name_to_search,
                'Title': item['name'],
                'Link': item['url'],
                'Snippet': clean_snippet,  # Clean snippet without HTML tags
                'FullText': full_text
            }
            scraped_data.append(row_data)
    else:
        print(f"No results found for {legal_name_to_search}.")

# Convert the scraped data to a DataFrame if it's not empty
if scraped_data:
    bing_data = pd.DataFrame(scraped_data)
    # Check if FullText column exists before applying transformation
    if 'FullText' in bing_data.columns:
        bing_data['FullText'] = bing_data['FullText'].apply(lambda x: "None" if not x else x)

    # Display a preview of the data
    print(bing_data.head())

    # Ensure the output directory exists
    output_dir = '/content/drive/My Drive/MTD/KYC data/Output'
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    # Save the results to a CSV file
```

```
    output_path = os.path.join(output_dir, 'bing_search_results.csv')
    bing_data.to_csv(output_path, index=False, quoting=1, escapechar='\\')  # Quoting and escape for special characters
    print(f"Data collection complete. Results saved to '{output_path}'.")
else:
    print("No valid data was collected. The DataFrame is empty.")

"""# 3. Natural language processing (NLP)

## 3.1. VADER

### 3.1.1. Google
"""

# Define file paths
google_file_path = '/content/drive/My Drive/MTD/KYC data/Output/google_search_results.csv'
bing_file_path = '/content/drive/My Drive/MTD/KYC data/Output/bing_search_results.csv'

# Load CSV files into DataFrames
google_data = pd.read_csv(google_file_path)
bing_data = pd.read_csv(bing_file_path)

# Display first few rows
print("Google Data:")
print(google_data.head())

print("\nBing Data:")
print(bing_data.head())

print(len(gvaders))
print(len(source_indicator))

import nltk
from nltk.sentiment import SentimentIntensityAnalyzer
from tqdm.notebook import tqdm
```

```
# Download the VADER lexicon
nltk.download('vader_lexicon')

# Initialize the Sentiment Intensity Analyzer
sia = SentimentIntensityAnalyzer()

from nltk.sentiment import SentimentIntensityAnalyzer
from tqdm import tqdm
import pandas as pd

# Initialize SentimentIntensityAnalyzer
sia = SentimentIntensityAnalyzer()

res = {}
source_indicator = []  # List to store whether 'FullText' or 'Snippet' was used

# Set a maximum character limit for FullText
MAX_TEXT_LENGTH = 500  # Adjust this limit as needed

# Loop through each row in google_data
for i, row in tqdm(google_data.iterrows(), total=google_data.shape[0]):
    source_used = None
    text = ""

    # Use 'FullText' if it's valid, truncate if necessary
    if pd.notnull(row['FullText']) and row['FullText'] != "None":
        text = row['FullText'][:MAX_TEXT_LENGTH]  # Truncate text
        source_used = 'FullText'
    elif pd.notnull(row['Snippet']) and row['Snippet'] != "None":
        text = row['Snippet']
        source_used = 'Snippet'

    # Only add to res and source_indicator if text is not empty
    if text.strip():  # Check if valid text exists
```

```
        myid = row['Link']
        res[myid] = sia.polarity_scores(text)
        source_indicator.append(source_used)

# Convert results dictionary to DataFrame
gvaders = pd.DataFrame(res).T

# Ensure source_indicator matches the DataFrame length
source_indicator = source_indicator[:len(gvaders)]

# Add the source indicator as a new column to the DataFrame
gvaders['SourceUsed'] = source_indicator

# Display the final DataFrame
print(gvaders)

# Reset the index of vaders_df and keep the current index as a column
gvaders = gvaders.reset_index()

# Rename the new column to 'Link' if the index was 'Link'
gvaders = gvaders.rename(columns={'index': 'vaders_link'})

# Now, 'Link' is a regular column in vaders_df

# Assuming vaders is the DataFrame with sentiment analysis results
gvaders = gvaders.rename(columns={
    'neg': 'vaders_neg',
    'neu': 'vaders_neu',
    'pos': 'vaders_pos',
    'compound': 'vaders_compound',
    'SourceUsed': 'vaders_source'
})

# Now, vaders_df has the columns renamed as specified
gvaders['vaders_composite'] = gvaders['vaders_pos'] - gvaders['vaders_neg']
```

```
gvaders

# Merge google_data with vaders on the 'Link' column
google_data_vaders = pd.merge(google_data, gvaders, left_on='Link', right_on='vaders_link', how='left')

# Now, merged_data contains information from both google_data and vaders
google_data_vaders

"""### 3.1.2. Bing"""

import nltk
from nltk.sentiment import SentimentIntensityAnalyzer
from tqdm.notebook import tqdm

# Download the VADER lexicon
nltk.download('vader_lexicon')

# Initialize the Sentiment Intensity Analyzer
sia = SentimentIntensityAnalyzer()

from nltk.sentiment import SentimentIntensityAnalyzer
from tqdm import tqdm
import pandas as pd

# Initialize SentimentIntensityAnalyzer
sia = SentimentIntensityAnalyzer()

res = {}
source_indicator = []  # List to store whether 'FullText' or 'Snippet' was used

# Set a maximum character limit for FullText
MAX_TEXT_LENGTH = 500  # Adjust as needed

# Loop through each row in bing_data
```

```
for i, row in tqdm(bing_data.iterrows(), total=bing_data.shape[0]):
    source_used = None
    text = ""

    # Use 'FullText' if it's a valid string, otherwise use 'Snippet'
    if pd.notnull(row['FullText']) and row['FullText'] != "None":
        text = row['FullText'][:MAX_TEXT_LENGTH]  # Truncate FullText
        source_used = 'FullText'
    elif pd.notnull(row['Snippet']) and row['Snippet'] != "None":
        text = row['Snippet']
        source_used = 'Snippet'

    # Only add to res and source_indicator if text is not empty
    if text.strip():
        myid = row['Link']
        res[myid] = sia.polarity_scores(text)
        source_indicator.append(source_used)

# Convert results dictionary to DataFrame
bvaders = pd.DataFrame(res).T

# Ensure source_indicator matches the DataFrame length
source_indicator = source_indicator[:len(bvaders)]

# Add the source indicator as a new column to the DataFrame
bvaders['SourceUsed'] = source_indicator

# Display the final DataFrame
print(bvaders)

# Reset the index of vaders_df and keep the current index as a column
bvaders = bvaders.reset_index()

# Rename the new column to 'Link' if the index was 'Link'
bvaders = bvaders.rename(columns={'index': 'vaders_link'})
```

```
# Now, 'Link' is a regular column in vaders_df

# Assuming vaders is the DataFrame with sentiment analysis results
bvaders = bvaders.rename(columns={
    'neg': 'vaders_neg',
    'neu': 'vaders_neu',
    'pos': 'vaders_pos',
    'compound': 'vaders_compound',
    'SourceUsed': 'vaders_source'
})

# Now, vaders_df has the columns renamed as specified
bvaders['vaders_composite'] = bvaders['vaders_pos'] - bvaders['vaders_neg']

# Merge bing_data with vaders on the 'Link' column
bing_data_vaders = pd.merge(bing_data, bvaders, left_on='Link', right_on='vaders_link', how='left')

# Now, merged_data contains information from both google_data and vaders
bing_data_vaders

"""### 3.1.3 Vaders comparison Google VS Bing"""

import matplotlib.pyplot as plt
import seaborn as sns

# Combine Bing and Google data
bing_data_vaders['Source'] = 'Bing'
google_data_vaders['Source'] = 'Google'

combined_data = pd.concat([
    bing_data_vaders[['vaders_compound', 'Source']],
    google_data_vaders[['vaders_compound', 'Source']]
])
```

```
# Plot boxplot comparison
plt.figure(figsize=(10, 6))
sns.boxplot(x='Source', y='vaders_compound', data=combined_data, palette='coolwarm')

# Add annotations for each box
stats = combined_data.groupby('Source')['vaders_compound'].describe()

for i, source in enumerate(stats.index):
    median = stats.loc[source, '50%']  # Median
    q1 = stats.loc[source, '25%']     # 1st quartile
    q3 = stats.loc[source, '75%']     # 3rd quartile
    count = stats.loc[source, 'count']  # Count of data points

    # Annotate statistics on the boxplot
    plt.text(i, median, f'Median: {median:.2f}', ha='center', va='center', fontsize=10, color='black')
    plt.text(i, q1, f'Q1: {q1:.2f}', ha='center', va='center', fontsize=9, color='gray')
    plt.text(i, q3, f'Q3: {q3:.2f}', ha='center', va='center', fontsize=9, color='gray')
    plt.text(i, -0.9, f'n: {int(count)}', ha='center', va='center', fontsize=10, color='black')  # Add count
at bottom

# Customize plot
plt.title('Distribution of VADER Compound Scores: Bing vs. Google')
plt.xlabel('Source')
plt.ylabel('VADER Compound Score')
plt.axhline(0, color='black', linestyle='--', linewidth=1)

plt.show()

# Plot bar chart
ax = sentiment_comparison.plot(kind='bar', figsize=(10, 6), color=['blue', 'green'])

# Customize plot
plt.title('Sentiment Category Comparison: Bing vs. Google')
plt.xlabel('Sentiment Category')
plt.ylabel('Count')
```

```
plt.xticks(rotation=0)
plt.legend(title='Source')

# Annotate the exact values on the bars
for p in ax.patches:
    # Get the height (value) of each bar
    value = int(p.get_height())
    if value > 0:  # Only display non-zero values
        # Place the text at the top of the bar
        ax.text(p.get_x() + p.get_width() / 2, p.get_height() + 5,
                f'{value}', ha='center', va='bottom', fontsize=10, fontweight='bold')

plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Aggregate vaders_compound by legal_name for Bing
bing_grouped = bing_data_vaders.groupby('legal_name')['vaders_compound'].mean().reset_index()

# Aggregate vaders_compound by legal_name for Google
google_grouped =
google_data_vaders.groupby('legal_name')['vaders_compound'].mean().reset_index()

# Merge the two datasets on legal_name
merged_data = pd.merge(bing_grouped, google_grouped, on='legal_name', how='outer',
suffixes=('_bing', '_google'))

# Fill NaN values with 0 (assuming no score as neutral sentiment)
merged_data.fillna(0, inplace=True)

# Calculate the absolute discrepancy between Bing and Google scores
```

```
merged_data['discrepancy'] = abs(merged_data['vaders_compound_bing'] - merged_data['vaders_compound_google'])

# Sort the data by the largest discrepancies
top_discrepancies = merged_data.sort_values(by='discrepancy', ascending=False).head(20)

# Set figure size
plt.figure(figsize=(12, 6))

# Setting the positions for the bars
pos = np.arange(len(top_discrepancies['legal_name']))
width = 0.35

# Plot Bing scores
plt.bar(pos - width/2, top_discrepancies['vaders_compound_bing'], width, alpha=0.7, color='blue', label='Bing')

# Plot Google scores
plt.bar(pos + width/2, top_discrepancies['vaders_compound_google'], width, alpha=0.7, color='green', label='Google')

# Add discrepancy values above the bars
for i, (bing, google, disc) in enumerate(zip(top_discrepancies['vaders_compound_bing'],
                                             top_discrepancies['vaders_compound_google'],
                                             top_discrepancies['discrepancy'])):
    plt.text(i - width/2, bing + 0.01, f'{bing:.2f}', ha='center', va='bottom', fontsize=8, color='black')
    plt.text(i + width/2, google + 0.01, f'{google:.2f}', ha='center', va='bottom', fontsize=8, color='black')
    plt.text(i, max(bing, google) + 0.05, f'Diff: {disc:.2f}', ha='center', fontsize=9, fontweight='bold', color='red')

# Customize the plot
plt.xlabel('Legal Name')
plt.ylabel('Average VADER Compound Score')
plt.title('Top 20 Discrepancies in VADER Compound Scores: Bing vs. Google')
```

```
plt.xticks(pos, top_discrepancies['legal_name'], rotation=90)
plt.legend()

# Show the plot
plt.tight_layout()
plt.show()

# Plot the results for clarity
plt.figure(figsize=(12, 6))
bar_width = 0.25
index = np.arange(len(discrepancy_df))

# Plot total Bing links
plt.bar(index, discrepancy_df['Bing_Links'], bar_width, label='Bing Links', color='blue')

# Plot total Google links
plt.bar(index + bar_width, discrepancy_df['Google_Links'], bar_width, label='Google Links', color='green')

# Plot matched links
bars_matched = plt.bar(index + 2 * bar_width, discrepancy_df['Matched_Links'], bar_width, label='Matched Links', color='orange')

# Annotate the matched links count on top of the orange bars
for bar, value in zip(bars_matched, discrepancy_df['Matched_Links']):
    plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height() + 0.5,
             f'{value}', ha='center', va='bottom', fontsize=10, fontweight='bold', color='red')

# Customize the plot
plt.xlabel('Company', fontsize=12)
plt.ylabel('Number of Links', fontsize=12)
plt.title('Link Comparison for Top Discrepancies')
plt.xticks(index + bar_width, discrepancy_df['Company'], rotation=90, fontsize=10)
plt.legend()
```

```
plt.tight_layout()
plt.show()

# Relaxed condition for no discrepancies
threshold = 0.01
no_discrepancy_data = merged_data[
    abs(merged_data['vaders_compound_bing'] - merged_data['vaders_compound_google']) < threshold
]

# Initialize a list to store results
no_discrepancy_analysis = []

# Analyze links for these companies
for company in no_discrepancy_data['legal_name']:
    # Get Bing and Google links for the company
    bing_links = bing_data_vaders[bing_data_vaders['legal_name'] == company]['cleaned_link'].unique()
    google_links = google_data_vaders[google_data_vaders['legal_name'] == company]['cleaned_link'].unique()

    # Calculate matched links
    matched_links = set(bing_links).intersection(set(google_links))

    # Append results
    no_discrepancy_analysis.append({
        'Company': company,
        'Bing_Links': len(bing_links),
        'Google_Links': len(google_links),
        'Matched_Links': len(matched_links)
    })

# Convert results to a DataFrame
no_discrepancy_df = pd.DataFrame(no_discrepancy_analysis)
```

```
# Check if the DataFrame is not empty
if not no_discrepancy_df.empty:
    # Plot the results for clarity
    plt.figure(figsize=(12, 6))
    bar_width = 0.3
    index = np.arange(len(no_discrepancy_df))

    # Plot total Bing links
    plt.bar(index, no_discrepancy_df['Bing_Links'], bar_width, label='Bing Links', color='blue')

    # Plot total Google links
    plt.bar(index + bar_width, no_discrepancy_df['Google_Links'], bar_width, label='Google Links', color='green')

    # Plot matched links
    bars_matched = plt.bar(index + 2 * bar_width, no_discrepancy_df['Matched_Links'], bar_width, label='Matched Links', color='orange')

    # Annotate matched links count on top of the orange bars
    for bar, value in zip(bars_matched, no_discrepancy_df['Matched_Links']):
        plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height() + 0.5,
                 f'{value}', ha='center', va='bottom', fontsize=10, fontweight='bold', color='red')

    # Customize the plot
    plt.xlabel('Company', fontsize=12)
    plt.ylabel('Number of Links', fontsize=12)
    plt.title('Link Comparison for Companies with No Discrepancies')
    plt.xticks(index + bar_width, no_discrepancy_df['Company'], rotation=90, fontsize=10)
    plt.legend()

    plt.tight_layout()
    plt.show()
else:
    print("No companies found with matching VADER compound scores.")
```

```
"""## 3.2. Roberta Pretrained Model

"""

from transformers import AutoTokenizer, AutoModelForSequenceClassification
from scipy.special import softmax

MODEL = "cardiffnlp/twitter-roberta-base-sentiment"
tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModelForSequenceClassification.from_pretrained(MODEL)

def get_sentiment_scores(row):
    # Use 'Snippet' if 'FullText' is "None" or not available; otherwise, use 'FullText'
    if row['FullText'] == "None" or pd.isna(row['FullText']):
        text = row['Snippet']
        source = 'Snippet'
    else:
        text = row['FullText']
        source = 'FullText'

    # Truncate the text to fit the model's maximum input size
    inputs = tokenizer.encode_plus(
        text,
        add_special_tokens=True,
        max_length=512,
        truncation=True,
        return_tensors='pt'
    )

    outputs = model(**inputs)
    scores = outputs[0][0].detach().numpy()
    sentiment_scores = softmax(scores)

    return sentiment_scores, source
```

```
# First, filter out the necessary columns and create new dataframes
groberta = google_data.filter(['Link', 'Snippet', 'FullText']).copy()
broberta = bing_data.filter(['Link', 'Snippet', 'FullText']).copy()

# Apply the function to each dataframe and split the results into two columns
groberta[['SentimentScores', 'roberta_source']] = groberta.apply(get_sentiment_scores, axis=1, result_type='expand')
broberta[['SentimentScores', 'roberta_source']] = broberta.apply(get_sentiment_scores, axis=1, result_type='expand')

# For groberta dataframe
groberta['roberta_neg'] = groberta['SentimentScores'].apply(lambda scores: scores[0])
groberta['roberta_neu'] = groberta['SentimentScores'].apply(lambda scores: scores[1])
groberta['roberta_pos'] = groberta['SentimentScores'].apply(lambda scores: scores[2])

# For broberta dataframe
broberta['roberta_neg'] = broberta['SentimentScores'].apply(lambda scores: scores[0])
broberta['roberta_neu'] = broberta['SentimentScores'].apply(lambda scores: scores[1])
broberta['roberta_pos'] = broberta['SentimentScores'].apply(lambda scores: scores[2])

# Rename 'Link' to 'roberta_link' and filter the required columns for groberta
groberta = groberta.rename(columns={'Link': 'roberta_link'}).filter(['roberta_link', 'roberta_neg', 'roberta_neu', 'roberta_pos', 'roberta_source'])

# Rename 'Link' to 'roberta_link' and filter the required columns for broberta
broberta = broberta.rename(columns={'Link': 'roberta_link'}).filter(['roberta_link', 'roberta_neg', 'roberta_neu', 'roberta_pos', 'roberta_source'])

groberta['roberta_composite'] = groberta['roberta_pos'] - groberta['roberta_neg']
broberta['roberta_composite'] = broberta['roberta_pos'] - broberta['roberta_neg']

# Display the updated DataFrames
print("Updated groberta:")
display(groberta)
```

```
print("Updated broberta:")
display(broberta)

print("Google_data:")
display(google_data)

print("bing_data:")
display(bing_data)

# Step 1: Normalize link columns in all dataframes
def normalize_links(df, column):
    return df[column].str.strip().str.lower().str.rstrip('/')

# Normalize 'Link' in google_data, bing_data and 'roberta_link' in groberta, broberta
google_data['Link'] = normalize_links(google_data, 'Link')
bing_data['Link'] = normalize_links(bing_data, 'Link')
groberta['roberta_link'] = normalize_links(groberta, 'roberta_link')
broberta['roberta_link'] = normalize_links(broberta, 'roberta_link')

# Step 2: Remove duplicates from groberta and broberta
groberta = groberta.drop_duplicates(subset=['roberta_link'])
broberta = broberta.drop_duplicates(subset=['roberta_link'])

# Step 3: Perform Left Join to Keep All Rows
google_data_roberta = google_data.merge(
    groberta,
    left_on='Link',
    right_on='roberta_link',
    how='left'
)

bing_data_roberta = bing_data.merge(
    broberta,
    left_on='Link',
    right_on='roberta_link',
```

```
    how='left'
)

# Step 4: Validate Row Counts
print(f"Row count in google_data: {google_data.shape[0]}")
print(f"Row count in google_data_roberta: {google_data_roberta.shape[0]}")
print(f"Row count in bing_data: {bing_data.shape[0]}")
print(f"Row count in bing_data_roberta: {bing_data_roberta.shape[0]}")

# Step 5: Display Results
print("Google Data with Roberta Linked:")
display(google_data_roberta)

print("Bing Data with Roberta Linked:")
display(bing_data_roberta)

import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

# Combine Bing and Google Roberta data
bing_data_roberta['Source'] = 'Bing'
google_data_roberta['Source'] = 'Google'

# Concatenate the datasets
combined_roberta_data = pd.concat([
    bing_data_roberta[['roberta_composite', 'Source']],
    google_data_roberta[['roberta_composite', 'Source']]
])

# Drop NaN values and ensure numeric type
combined_roberta_data = combined_roberta_data.dropna(subset=['roberta_composite'])
combined_roberta_data['roberta_composite'] =
pd.to_numeric(combined_roberta_data['roberta_composite'], errors='coerce')
```

```
# Check for NaNs after numeric conversion and drop them
combined_roberta_data = combined_roberta_data.dropna(subset=['roberta_composite'])

# Plot boxplot comparison
plt.figure(figsize=(10, 6))
sns.boxplot(x='Source', y='roberta_composite', data=combined_roberta_data, palette='coolwarm')

# Add annotations for each box
stats = combined_roberta_data.groupby('Source')['roberta_composite'].describe()

for i, source in enumerate(stats.index):
    median = stats.loc[source, '50%']  # Median
    q1 = stats.loc[source, '25%']     # 1st quartile
    q3 = stats.loc[source, '75%']     # 3rd quartile
    count = stats.loc[source, 'count']  # Count of data points

    # Annotate statistics on the boxplot
    plt.text(i, median, f'Median: {median:.2f}', ha='center', va='center', fontsize=10, color='black')
    plt.text(i, q1, f'Q1: {q1:.2f}', ha='center', va='center', fontsize=9, color='gray')
    plt.text(i, q3, f'Q3: {q3:.2f}', ha='center', va='center', fontsize=9, color='gray')
    plt.text(i, -0.9, f'n: {int(count)}', ha='center', va='center', fontsize=10, color='black')

# Customize plot
plt.title('Distribution of Roberta Composite Scores: Bing vs. Google')
plt.xlabel('Source')
plt.ylabel('Roberta Composite Score')
plt.axhline(0, color='black', linestyle='--', linewidth=1)

plt.show()

import matplotlib.pyplot as plt

# Step 1: Aggregate sentiment counts
bing_sentiment_counts = {
```

```
    'Negative': (bing_data_roberta['roberta_neg'] > 0.5).sum(),
    'Neutral': (bing_data_roberta['roberta_neu'] > 0.5).sum(),
    'Positive': (bing_data_roberta['roberta_pos'] > 0.5).sum()
}

google_sentiment_counts = {
    'Negative': (google_data_roberta['roberta_neg'] > 0.5).sum(),
    'Neutral': (google_data_roberta['roberta_neu'] > 0.5).sum(),
    'Positive': (google_data_roberta['roberta_pos'] > 0.5).sum()
}

# Step 2: Combine data into a DataFrame
import pandas as pd

sentiment_comparison = pd.DataFrame({
    'Bing': bing_sentiment_counts,
    'Google': google_sentiment_counts
})

# Step 3: Plot bar chart
ax = sentiment_comparison.plot(kind='bar', figsize=(10, 6), color=['blue', 'green'])

# Customize plot
plt.title('Sentiment Category Comparison: Bing vs. Google')
plt.xlabel('Sentiment Category')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.legend(title='Source')

# Annotate the exact values on the bars
for p in ax.patches:
    # Get the height (value) of each bar
    value = int(p.get_height())
    if value > 0:  # Only display non-zero values
        # Place the text at the top of the bar
```

```
        ax.text(p.get_x() + p.get_width() / 2, p.get_height() + 5,
                f'{value}', ha='center', va='bottom', fontsize=10, fontweight='bold')

plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Step 1: Aggregate roberta_composite by legal_name for Bing
bing_grouped = bing_data_roberta.groupby('legal_name')['roberta_composite'].mean().reset_index()

# Step 2: Aggregate roberta_composite by legal_name for Google
google_grouped = google_data_roberta.groupby('legal_name')['roberta_composite'].mean().reset_index()

# Step 3: Merge the two datasets on legal_name
merged_data = pd.merge(bing_grouped, google_grouped, on='legal_name', how='outer', suffixes=('_bing', '_google'))

# Fill NaN values with 0 (assuming no score as neutral sentiment)
merged_data.fillna(0, inplace=True)

# Step 4: Calculate the absolute discrepancy between Bing and Google scores
merged_data['discrepancy'] = abs(merged_data['roberta_composite_bing'] - merged_data['roberta_composite_google'])

# Sort the data by the largest discrepancies
top_discrepancies = merged_data.sort_values(by='discrepancy', ascending=False).head(20)

# Step 5: Visualization
plt.figure(figsize=(12, 6))

# Setting the positions for the bars
```

```
pos = np.arange(len(top_discrepancies['legal_name']))
width = 0.35

# Plot Bing scores
plt.bar(pos - width/2, top_discrepancies['roberta_composite_bing'], width, alpha=0.7, color='blue', label='Bing')

# Plot Google scores
plt.bar(pos + width/2, top_discrepancies['roberta_composite_google'], width, alpha=0.7, color='green', label='Google')

# Add discrepancy values above the bars
for i, (bing, google, disc) in enumerate(zip(top_discrepancies['roberta_composite_bing'],
                                             top_discrepancies['roberta_composite_google'],
                                             top_discrepancies['discrepancy'])):
    plt.text(i - width/2, bing + 0.01, f'{bing:.2f}', ha='center', va='bottom', fontsize=8, color='black')
    plt.text(i + width/2, google + 0.01, f'{google:.2f}', ha='center', va='bottom', fontsize=8, color='black')
    plt.text(i, max(bing, google) + 0.05, f'Diff: {disc:.2f}', ha='center', fontsize=9, fontweight='bold', color='red')

# Customize the plot
plt.xlabel('Legal Name')
plt.ylabel('Average Roberta Composite Score')
plt.title('Top 20 Discrepancies in Roberta Composite Scores: Bing vs. Google')
plt.xticks(pos, top_discrepancies['legal_name'], rotation=90)
plt.legend()

# Show the plot
plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt
import numpy as np
```

```
# Calculate discrepancies using roberta_composite
merged_data['discrepancy'] = abs(merged_data['roberta_composite_bing'] - merged_data['roberta_composite_google'])

# Select top discrepancies
top_discrepancies = merged_data.sort_values(by='discrepancy', ascending=False).head(10)

# Initialize results
discrepancy_analysis = []

# Analyze links for these companies
for company in top_discrepancies['legal_name']:
    # Get Bing and Google links for the company
    bing_links = bing_data_roberta[bing_data_roberta['legal_name'] == company]['roberta_link'].unique()
    google_links = google_data_roberta[google_data_roberta['legal_name'] == company]['roberta_link'].unique()

    # Calculate matched links
    matched_links = set(bing_links).intersection(set(google_links))

    # Append results
    discrepancy_analysis.append({
        'Company': company,
        'Bing_Links': len(bing_links),
        'Google_Links': len(google_links),
        'Matched_Links': len(matched_links)
    })

# Convert results to a DataFrame
discrepancy_df = pd.DataFrame(discrepancy_analysis)

# Plot the results
plt.figure(figsize=(12, 6))
bar_width = 0.25
```

```
index = np.arange(len(discrepancy_df))

# Plot Bing links
plt.bar(index, discrepancy_df['Bing_Links'], bar_width, label='Bing Links', color='blue')

# Plot Google links
plt.bar(index + bar_width, discrepancy_df['Google_Links'], bar_width, label='Google Links', color='green')

# Plot Matched links
bars_matched = plt.bar(index + 2 * bar_width, discrepancy_df['Matched_Links'], bar_width, label='Matched Links', color='orange')

# Annotate matched links
for bar, value in zip(bars_matched, discrepancy_df['Matched_Links']):
    plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height() + 0.5,
             f'{value}', ha='center', va='bottom', fontsize=10, fontweight='bold', color='red')

# Customize plot
plt.xlabel('Company', fontsize=12)
plt.ylabel('Number of Links', fontsize=12)
plt.title('Link Comparison for Top Discrepancies (Roberta Composite Scores)')
plt.xticks(index + bar_width, discrepancy_df['Company'], rotation=90)
plt.legend()

plt.tight_layout()
plt.show()

# Relaxed condition for no discrepancies
threshold = 0.01
no_discrepancy_data = merged_data[
    abs(merged_data['roberta_composite_bing'] - merged_data['roberta_composite_google']) < threshold
]
```

```
# Initialize analysis list
no_discrepancy_analysis = []

# Analyze links for companies with no discrepancies
for company in no_discrepancy_data['legal_name']:
    # Get Bing and Google links for the company
    bing_links = bing_data_roberta[bing_data_roberta['legal_name'] == company]['roberta_link'].unique()
    google_links = google_data_roberta[google_data_roberta['legal_name'] == company]['roberta_link'].unique()

    # Calculate matched links
    matched_links = set(bing_links).intersection(set(google_links))

    # Append results
    no_discrepancy_analysis.append({
        'Company': company,
        'Bing_Links': len(bing_links),
        'Google_Links': len(google_links),
        'Matched_Links': len(matched_links)
    })

# Convert results to DataFrame
no_discrepancy_df = pd.DataFrame(no_discrepancy_analysis)

# Plot if data is available
if not no_discrepancy_df.empty:
    plt.figure(figsize=(12, 6))
    bar_width = 0.3
    index = np.arange(len(no_discrepancy_df))

    # Plot Bing links
    plt.bar(index, no_discrepancy_df['Bing_Links'], bar_width, label='Bing Links', color='blue')

    # Plot Google links
```

```
    plt.bar(index + bar_width, no_discrepancy_df['Google_Links'], bar_width, label='Google Links', color='green')

    # Plot Matched links
    bars_matched = plt.bar(index + 2 * bar_width, no_discrepancy_df['Matched_Links'], bar_width, label='Matched Links', color='orange')

    # Annotate matched links
    for bar, value in zip(bars_matched, no_discrepancy_df['Matched_Links']):
        plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height() + 0.5,
                 f'{value}', ha='center', va='bottom', fontsize=10, fontweight='bold', color='red')

    # Customize plot
    plt.xlabel('Company', fontsize=12)
    plt.ylabel('Number of Links', fontsize=12)
    plt.title('Link Comparison for Companies with No Discrepancies (Roberta Composite Scores)')
    plt.xticks(index + bar_width, no_discrepancy_df['Company'], rotation=90)
    plt.legend()

    plt.tight_layout()
    plt.show()
else:
    print("No companies found with matching Roberta composite scores.")

from google.colab import drive
import pandas as pd

# Mount Google Drive
drive.mount('/content/drive')

# Define output directory
output_dir = '/content/drive/My Drive/MTD/KYC data/Output/'

# Save the DataFrames as CSV files
google_data_vaders.to_csv(f'{output_dir}google_data_vaders.csv', index=False)
```

```
google_data_roberta.to_csv(f'{output_dir}google_data_roberta.csv', index=False)
bing_data_vaders.to_csv(f'{output_dir}bing_data_vaders.csv', index=False)
bing_data_roberta.to_csv(f'{output_dir}bing_data_roberta.csv', index=False)

print("Files saved successfully to Google Drive:")
print(f"- {output_dir}google_data_vaders.csv")
print(f"- {output_dir}google_data_roberta.csv")
print(f"- {output_dir}bing_data_vaders.csv")
print(f"- {output_dir}bing_data_roberta.csv")

"""### BERT"""

import pandas as pd
from transformers import BertTokenizer, BertForSequenceClassification
import torch
from tqdm import tqdm

# Load pre-trained model and tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert-base-uncased')

def bert_sentiment_analysis(text):
    # Tokenize and prepare data for BERT
    inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True, max_length=512)
    # Predict sentiment
    with torch.no_grad():
        outputs = model(**inputs)
        predictions = torch.nn.functional.softmax(outputs.logits, dim=-1)
    return predictions.numpy()[0].tolist()

# Preparing data for sentiment analysis
bert_results = []
source_indicator = []
links = []  # List to store the link of each row
```

```
# Loop through each row in google_data
for i, row in tqdm(google_data.iterrows(), total=google_data.shape[0]):
    # Store the link
    links.append(row['Link'])  # Assuming 'Link' is the column name for links in google_data

    # Use 'FullText' if it's valid, otherwise use 'Snippet'
    if pd.notnull(row['FullText']) and row['FullText'] != "None":
        text = row['FullText']
        source_indicator.append('FullText')
    else:
        text = row['Snippet']
        source_indicator.append('Snippet')

    text = text if pd.notnull(text) and text != "None" else ""
    bert_result = bert_sentiment_analysis(text)
    bert_results.append(bert_result)

# Creating the DataFrame
gbert = pd.DataFrame(bert_results, columns=['BERT_Neg', 'BERT_Pos'])
gbert['SourceUsed'] = source_indicator
gbert['Link'] = links  # Add the links to the DataFrame

bert_results_bing = []
source_indicator_bing = []
links_bing = []  # List to store the link of each row in bing_data

# Loop through each row in bing_data
for i, row in tqdm(bing_data.iterrows(), total=bing_data.shape[0]):
    # Store the link
    links_bing.append(row['Link'])  # Assuming 'Link' is the column name for links in bing_data

    # Use 'FullText' if it's valid, otherwise use 'Snippet'
    if pd.notnull(row['FullText']) and row['FullText'] != "None":
        text = row['FullText']
        source_indicator_bing.append('FullText')
```

```
    else:
        text = row['Snippet']
        source_indicator_bing.append('Snippet')

    text = text if pd.notnull(text) and text != "None" else ""
    bert_result = bert_sentiment_analysis(text)
    bert_results_bing.append(bert_result)

# Creating the DataFrame for Bing data
bbert = pd.DataFrame(bert_results_bing, columns=['BERT_Neg', 'BERT_Pos'])
bbert['SourceUsed'] = source_indicator_bing
bbert['Link'] = links_bing  # Add the links to the DataFrame
bbert

# Aggregate BERT results by 'Link' to ensure uniqueness
gbert = gbert.groupby('Link', as_index=False).agg({
    'BERT_Neg': 'mean',
    'BERT_Pos': 'mean',
    'SourceUsed': 'first'  # Retain one source
})

bbert = bbert.groupby('Link', as_index=False).agg({
    'BERT_Neg': 'mean',
    'BERT_Pos': 'mean',
    'SourceUsed': 'first'
})

# Merge google_data with gbert
google_data_bert = pd.merge(google_data, gbert, on='Link', how='left')
google_data_bert['BERT_Composite'] = google_data_bert['BERT_Pos'] - google_data_bert['BERT_Neg']

# Merge bing_data with bbert
bing_data_bert = pd.merge(bing_data, bbert, on='Link', how='left')
bing_data_bert['BERT_Composite'] = bing_data_bert['BERT_Pos'] - bing_data_bert['BERT_Neg']
```

```
# Verify row counts
print("Row count in google_data:", len(google_data))
print("Row count in google_data_bert:", len(google_data_bert))
print("Row count in bing_data:", len(bing_data))
print("Row count in bing_data_bert:", len(bing_data_bert))

from google.colab import drive
import pandas as pd

# Mount Google Drive
drive.mount('/content/drive')

# Define output directory
output_dir = '/content/drive/My Drive/MTD/KYC data/Output/'

# Save BERT DataFrames as CSV files
google_data_bert.to_csv(f'{output_dir}google_data_bert.csv', index=False)
bing_data_bert.to_csv(f'{output_dir}bing_data_bert.csv', index=False)

print("BERT files saved successfully to Google Drive:")
print(f"- {output_dir}google_data_bert.csv")
print(f"- {output_dir}bing_data_bert.csv")

import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

# Calculate BERT composite score for Bing and Google
bing_data_bert['bert_composite'] = bing_data_bert['BERT_Pos'] - bing_data_bert['BERT_Neg']
google_data_bert['bert_composite'] = google_data_bert['BERT_Pos'] -
google_data_bert['BERT_Neg']

# Add Source column to identify Bing and Google data
```

```
bing_data_bert['Source'] = 'Bing'
google_data_bert['Source'] = 'Google'

# Concatenate the datasets
combined_bert_data = pd.concat([
    bing_data_bert[['bert_composite', 'Source']],
    google_data_bert[['bert_composite', 'Source']]
])

# Drop NaN values and ensure numeric type
combined_bert_data = combined_bert_data.dropna(subset=['bert_composite'])
combined_bert_data['bert_composite'] = pd.to_numeric(combined_bert_data['bert_composite'],
errors='coerce')

# Plot boxplot comparison
plt.figure(figsize=(10, 6))
sns.boxplot(x='Source', y='bert_composite', data=combined_bert_data, palette='coolwarm')

# Add annotations for each box
stats = combined_bert_data.groupby('Source')['bert_composite'].describe()

for i, source in enumerate(stats.index):
    median = stats.loc[source, '50%']  # Median
    q1 = stats.loc[source, '25%']     # 1st quartile
    q3 = stats.loc[source, '75%']     # 3rd quartile
    count = stats.loc[source, 'count']  # Count of data points

    # Annotate statistics on the boxplot
    plt.text(i, median, f'Median: {median:.2f}', ha='center', va='center', fontsize=10, color='black')
    plt.text(i, q1, f'Q1: {q1:.2f}', ha='center', va='center', fontsize=9, color='gray')
    plt.text(i, q3, f'Q3: {q3:.2f}', ha='center', va='center', fontsize=9, color='gray')
    plt.text(i, -0.9, f'n: {int(count)}', ha='center', va='center', fontsize=10, color='black')

# Customize plot
plt.title('Distribution of BERT Composite Scores: Bing vs. Google')
```

```
plt.xlabel('Source')
plt.ylabel('BERT Composite Score')
plt.axhline(0, color='black', linestyle='--', linewidth=1)

plt.show()

import matplotlib.pyplot as plt
import pandas as pd

# Step 1: Aggregate sentiment counts
bing_sentiment_counts = {
    'Negative': (bbert['BERT_Neg'] > 0.5).sum(),
    'Positive': (bbert['BERT_Pos'] > 0.5).sum()
}

google_sentiment_counts = {
    'Negative': (gbert['BERT_Neg'] > 0.5).sum(),
    'Positive': (gbert['BERT_Pos'] > 0.5).sum()
}

# Step 2: Combine data into a DataFrame
sentiment_comparison = pd.DataFrame({
    'Bing': bing_sentiment_counts,
    'Google': google_sentiment_counts
})

# Step 3: Plot bar chart
ax = sentiment_comparison.plot(kind='bar', figsize=(10, 6), color=['blue', 'green'])

# Customize plot
plt.title('Sentiment Category Comparison: Bing vs. Google (BERT)')
plt.xlabel('Sentiment Category')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.legend(title='Source')
```

```
# Annotate the exact values on the bars
for p in ax.patches:
    value = int(p.get_height())
    if value > 0:
        ax.text(p.get_x() + p.get_width() / 2, p.get_height() + 5,
                f'{value}', ha='center', va='bottom', fontsize=10, fontweight='bold')

plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Step 1: Aggregate bert_composite by legal_name for Bing
bing_grouped = bing_data_bert.groupby('legal_name')['bert_composite'].mean().reset_index()

# Step 2: Aggregate bert_composite by legal_name for Google
google_grouped = google_data_bert.groupby('legal_name')['bert_composite'].mean().reset_index()

# Step 3: Merge the two datasets on legal_name
merged_data = pd.merge(bing_grouped, google_grouped, on='legal_name', how='outer',
suffixes=('_bing', '_google'))

# Fill NaN values with 0 (assuming no score as neutral sentiment)
merged_data.fillna(0, inplace=True)

# Step 4: Calculate the absolute discrepancy between Bing and Google scores
merged_data['discrepancy'] = abs(merged_data['bert_composite_bing'] -
merged_data['bert_composite_google'])

# Sort the data by the largest discrepancies
top_discrepancies = merged_data.sort_values(by='discrepancy', ascending=False).head(20)
```

```
# Step 5: Visualization
plt.figure(figsize=(12, 6))

# Setting the positions for the bars
pos = np.arange(len(top_discrepancies['legal_name']))
width = 0.35

# Plot Bing scores
plt.bar(pos - width/2, top_discrepancies['bert_composite_bing'], width, alpha=0.7, color='blue', label='Bing')

# Plot Google scores
plt.bar(pos + width/2, top_discrepancies['bert_composite_google'], width, alpha=0.7, color='green', label='Google')

# Add discrepancy values above the bars
for i, (bing, google, disc) in enumerate(zip(top_discrepancies['bert_composite_bing'],
                                             top_discrepancies['bert_composite_google'],
                                             top_discrepancies['discrepancy'])):
    plt.text(i - width/2, bing + 0.01, f'{bing:.2f}', ha='center', va='bottom', fontsize=8, color='black')
    plt.text(i + width/2, google + 0.01, f'{google:.2f}', ha='center', va='bottom', fontsize=8, color='black')
    plt.text(i, max(bing, google) + 0.05, f'Diff: {disc:.2f}', ha='center', fontsize=9, fontweight='bold', color='red')

# Customize the plot
plt.xlabel('Legal Name')
plt.ylabel('Average BERT Composite Score')
plt.title('Top 20 Discrepancies in BERT Composite Scores: Bing vs. Google')
plt.xticks(pos, top_discrepancies['legal_name'], rotation=90)
plt.legend()

# Show the plot
plt.tight_layout()
plt.show()
```

```
import matplotlib.pyplot as plt
import numpy as np

# Step 1: Calculate discrepancies using bert_composite
merged_data['discrepancy'] = abs(merged_data['bert_composite_bing'] -
merged_data['bert_composite_google'])

# Step 2: Select top discrepancies
top_discrepancies = merged_data.sort_values(by='discrepancy', ascending=False).head(10)

# Step 3: Initialize results
discrepancy_analysis = []

# Analyze links for these companies
for company in top_discrepancies['legal_name']:
    # Get Bing and Google links for the company
    bing_links = bing_data_bert[bing_data_bert['legal_name'] == company]['Link'].unique()
    google_links = google_data_bert[google_data_bert['legal_name'] == company]['Link'].unique()

    # Calculate matched links
    matched_links = set(bing_links).intersection(set(google_links))

    # Append results
    discrepancy_analysis.append({
        'Company': company,
        'Bing_Links': len(bing_links),
        'Google_Links': len(google_links),
        'Matched_Links': len(matched_links)
    })

# Step 4: Convert results to a DataFrame
discrepancy_df = pd.DataFrame(discrepancy_analysis)

# Step 5: Plot the results
```

```
plt.figure(figsize=(12, 6))
bar_width = 0.25
index = np.arange(len(discrepancy_df))

# Plot Bing links
plt.bar(index, discrepancy_df['Bing_Links'], bar_width, label='Bing Links', color='blue')

# Plot Google links
plt.bar(index + bar_width, discrepancy_df['Google_Links'], bar_width, label='Google Links', color='green')

# Plot Matched links
bars_matched = plt.bar(index + 2 * bar_width, discrepancy_df['Matched_Links'], bar_width, label='Matched Links', color='orange')

# Annotate matched links
for bar, value in zip(bars_matched, discrepancy_df['Matched_Links']):
    plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height() + 0.5,
             f'{value}', ha='center', va='bottom', fontsize=10, fontweight='bold', color='red')

# Customize plot
plt.xlabel('Company', fontsize=12)
plt.ylabel('Number of Links', fontsize=12)
plt.title('Link Comparison for Top Discrepancies (BERT Composite Scores)')
plt.xticks(index + bar_width, discrepancy_df['Company'], rotation=90)
plt.legend()

plt.tight_layout()
plt.show()

# Relaxed condition for no discrepancies
threshold = 0.01
no_discrepancy_data = merged_data[
    abs(merged_data['bert_composite_bing'] - merged_data['bert_composite_google']) < threshold
]
```

```
# Step 1: Initialize analysis list
no_discrepancy_analysis = []

# Step 2: Analyze links for companies with no discrepancies
for company in no_discrepancy_data['legal_name']:
    # Get Bing and Google links for the company
    bing_links = bing_data_bert[bing_data_bert['legal_name'] == company]['Link'].unique()
    google_links = google_data_bert[google_data_bert['legal_name'] == company]['Link'].unique()

    # Calculate matched links
    matched_links = set(bing_links).intersection(set(google_links))

    # Append results
    no_discrepancy_analysis.append({
        'Company': company,
        'Bing_Links': len(bing_links),
        'Google_Links': len(google_links),
        'Matched_Links': len(matched_links)
    })

# Step 3: Convert results to DataFrame
no_discrepancy_df = pd.DataFrame(no_discrepancy_analysis)

# Step 4: Plot if data is available
if not no_discrepancy_df.empty:
    plt.figure(figsize=(12, 6))
    bar_width = 0.3
    index = np.arange(len(no_discrepancy_df))

    # Plot Bing links
    plt.bar(index, no_discrepancy_df['Bing_Links'], bar_width, label='Bing Links', color='blue')

    # Plot Google links
```

```
    plt.bar(index + bar_width, no_discrepancy_df['Google_Links'], bar_width, label='Google Links', color='green')

    # Plot Matched links
    bars_matched = plt.bar(index + 2 * bar_width, no_discrepancy_df['Matched_Links'], bar_width, label='Matched Links', color='orange')

    # Annotate matched links
    for bar, value in zip(bars_matched, no_discrepancy_df['Matched_Links']):
        plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height() + 0.5,
                 f'{value}', ha='center', va='bottom', fontsize=10, fontweight='bold', color='red')

    # Customize plot
    plt.xlabel('Company', fontsize=12)
    plt.ylabel('Number of Links', fontsize=12)
    plt.title('Link Comparison for Companies with No Discrepancies (BERT Composite Scores)')
    plt.xticks(index + bar_width, no_discrepancy_df['Company'], rotation=90)
    plt.legend()

    plt.tight_layout()
    plt.show()
else:
    print("No companies found with matching BERT composite scores.")

"""# CHAT GPT"""

### Chat-GPT API:

pip install openai

pip install --upgrade openai

pip install openai==0.28

import openai
```

```
import pandas as pd
from tqdm import tqdm

# Replace with your OpenAI API key
openai.api_key = "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"

# Function to get sentiment analysis from GPT-3.5-turbo
def gpt_sentiment_analysis(text):
    try:
        # Call the GPT-3.5-turbo API
        response = openai.ChatCompletion.create(
            model="gpt-3.5-turbo",
            messages=[
                {"role": "system", "content": "You are an AI that performs sentiment analysis."},
                {"role": "user", "content": f"Please analyze the sentiment of this text and provide scores strictly in this format: [neg:<negative_score>,neu:<neutral_score>,pos:<positive_score>]. The values should be between 0 and 1.\nText: {text}"}
            ],
            max_tokens=100,
            temperature=0.2
        )
        # Extract and return the response
        result = response['choices'][0]['message']['content'].strip()
        return result
    except Exception as e:
        print(f"Error processing text: {e}")
        return None

# Function to process a dataset and get GPT sentiment analysis
def process_data_for_sentiment(data, data_name):
    gpt_results = []
    for i, row in tqdm(data.iterrows(), total=data.shape[0], desc=f"Processing {data_name}"):
        # Use 'FullText' if available, otherwise use 'Snippet'
        text = row['FullText'] if pd.notnull(row['FullText']) and row['FullText'] != "None" else row['Snippet']
```

```
        if pd.notnull(text) and text != "None":  # Ensure the text is valid
            sentiment_result = gpt_sentiment_analysis(text)
            gpt_results.append((row['Link'], sentiment_result))
        else:
            gpt_results.append((row['Link'], None))

    # Convert results to DataFrame
    gpt_sentiment_df = pd.DataFrame(gpt_results, columns=['Link', 'GPT_Sentiment'])
    return gpt_sentiment_df

# Process Google Data
gpt_google_sentiment = process_data_for_sentiment(google_data, "Google Data")
gpt_google_sentiment.to_csv("/content/drive/My Drive/MTD/KYC data/Output/gpt_google_sentiment_results.csv", index=False)
print("Saved GPT sentiment analysis for Google Data.")

# Process Bing Data
gpt_bing_sentiment = process_data_for_sentiment(bing_data, "Bing Data")
gpt_bing_sentiment.to_csv("/content/drive/My Drive/MTD/KYC data/Output/gpt_bing_sentiment_results.csv", index=False)
print("Saved GPT sentiment analysis for Bing Data.")

import pandas as pd
import re

# Function to split GPT sentiment into neg, neu, pos
def split_gpt_sentiment(sentiment):
    if isinstance(sentiment, str):
        matches = re.findall(r'neg:(\d*\.?\d*)|neu:(\d*\.?\d*)|pos:(\d*\.?\d*)', sentiment)
        # Extract values by position in matches
        neg = float(matches[0][0] or 0) if matches and len(matches) > 0 else 0.0
        neu = float(matches[1][1] or 0) if matches and len(matches) > 1 else 0.0
        pos = float(matches[2][2] or 0) if matches and len(matches) > 2 else 0.0
        return neg, neu, pos
    return 0.0, 0.0, 0.0
```

```
# Process GPT sentiment data and prepare for merge
def process_gpt_sentiment(df, source_name):
    # Split GPT sentiment
    df[['gpt_neg', 'gpt_neu', 'gpt_pos']] = pd.DataFrame(
        df['GPT_Sentiment'].apply(split_gpt_sentiment).tolist(), index=df.index
    )

    # Compute the composite score
    df['gpt_composite'] = df['gpt_pos'] - df['gpt_neg']

    # Deduplicate links to avoid merge duplication
    df = df.drop_duplicates(subset=['Link'], keep='first')

    # Add Source column
    df['Source'] = source_name

    return df

# Process Google and Bing GPT sentiment data
gpt_google_sentiment_processed = process_gpt_sentiment(gpt_google_sentiment, 'Google')
gpt_bing_sentiment_processed = process_gpt_sentiment(gpt_bing_sentiment, 'Bing')

# Define the columns to keep after merging
columns_to_keep = ['Link', 'gpt_neg', 'gpt_neu', 'gpt_pos', 'gpt_composite', 'Source']

# Merge GPT sentiment back to Google and Bing data
google_data_gpt = pd.merge(
    google_data,
    gpt_google_sentiment_processed[columns_to_keep],
    on='Link',
    how='left'
)

bing_data_gpt = pd.merge(
```

```
    bing_data,
    gpt_bing_sentiment_processed[columns_to_keep],
    on='Link',
    how='left'
)

# Fill missing values for rows without GPT sentiment
for col in ['gpt_neg', 'gpt_neu', 'gpt_pos', 'gpt_composite']:
    google_data_gpt[col] = google_data_gpt[col].fillna(0.0)
    bing_data_gpt[col] = bing_data_gpt[col].fillna(0.0)

# Set Source column for unmatched rows
google_data_gpt['Source'] = google_data_gpt['Source'].fillna('Google')
bing_data_gpt['Source'] = bing_data_gpt['Source'].fillna('Bing')

# Reorganize columns for consistency
google_data_gpt = google_data_gpt[['id', 'legal_name', 'Title', 'Link', 'Snippet', 'FullText', 'gpt_neg',
'gpt_neu', 'gpt_pos', 'gpt_composite', 'Source']]
bing_data_gpt = bing_data_gpt[['id', 'legal_name', 'Title', 'Link', 'Snippet', 'FullText', 'gpt_neg',
'gpt_neu', 'gpt_pos', 'gpt_composite', 'Source']]

# Display Results
print("Google Data with GPT Sentiment:")
display(google_data_gpt)

print("Bing Data with GPT Sentiment:")
display(bing_data_gpt)

# Validate row counts
print(f"Row count in google_data: {len(google_data)}")
print(f"Row count in google_data_gpt: {len(google_data_gpt)}")
print(f"Row count in bing_data: {len(bing_data)}")
print(f"Row count in bing_data_gpt: {len(bing_data_gpt)}")

import matplotlib.pyplot as plt
```

```
import seaborn as sns
import pandas as pd
import numpy as np

# Combine Bing and Google GPT data
bing_data_gpt['Source'] = 'Bing'
google_data_gpt['Source'] = 'Google'

# Concatenate the datasets
combined_gpt_data = pd.concat([
    bing_data_gpt[['gpt_composite', 'Source']],
    google_data_gpt[['gpt_composite', 'Source']]
])

# Drop NaN values and ensure numeric type
combined_gpt_data = combined_gpt_data.dropna(subset=['gpt_composite'])
combined_gpt_data['gpt_composite'] = pd.to_numeric(combined_gpt_data['gpt_composite'],
errors='coerce')

# Check for NaNs after numeric conversion and drop them
combined_gpt_data = combined_gpt_data.dropna(subset=['gpt_composite'])

# Plot boxplot comparison
plt.figure(figsize=(10, 6))
sns.boxplot(x='Source', y='gpt_composite', data=combined_gpt_data, palette='coolwarm')

# Add annotations for each box
stats = combined_gpt_data.groupby('Source')['gpt_composite'].describe()

for i, source in enumerate(stats.index):
    median = stats.loc[source, '50%']  # Median
    q1 = stats.loc[source, '25%']     # 1st quartile
    q3 = stats.loc[source, '75%']     # 3rd quartile
    count = stats.loc[source, 'count']  # Count of data points
```

```
    # Annotate statistics on the boxplot
    plt.text(i, median, f'Median: {median:.2f}', ha='center', va='center', fontsize=10, color='black')
    plt.text(i, q1, f'Q1: {q1:.2f}', ha='center', va='center', fontsize=9, color='gray')
    plt.text(i, q3, f'Q3: {q3:.2f}', ha='center', va='center', fontsize=9, color='gray')
    plt.text(i, -0.9, f'n: {int(count)}', ha='center', va='center', fontsize=10, color='black')

# Customize plot
plt.title('Distribution of GPT Composite Scores: Bing vs. Google')
plt.xlabel('Source')
plt.ylabel('GPT Composite Score')
plt.axhline(0, color='black', linestyle='--', linewidth=1)

plt.show()

import matplotlib.pyplot as plt
import pandas as pd

# Step 1: Aggregate sentiment counts
bing_sentiment_counts = {
    'Negative': (bing_data_gpt['gpt_neg'] > 0.5).sum(),
    'Neutral': (bing_data_gpt['gpt_neu'] > 0.5).sum(),
    'Positive': (bing_data_gpt['gpt_pos'] > 0.5).sum()
}

google_sentiment_counts = {
    'Negative': (google_data_gpt['gpt_neg'] > 0.5).sum(),
    'Neutral': (google_data_gpt['gpt_neu'] > 0.5).sum(),
    'Positive': (google_data_gpt['gpt_pos'] > 0.5).sum()
}

# Step 2: Combine data into a DataFrame
sentiment_comparison = pd.DataFrame({
    'Bing': bing_sentiment_counts,
    'Google': google_sentiment_counts
})
```

```
# Step 3: Plot bar chart
ax = sentiment_comparison.plot(kind='bar', figsize=(10, 6), color=['blue', 'green'])

# Customize plot
plt.title('Sentiment Category Comparison: Bing vs. Google (GPT)')
plt.xlabel('Sentiment Category')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.legend(title='Source')

# Annotate values on the bars
for p in ax.patches:
    value = int(p.get_height())
    if value > 0:
        ax.text(p.get_x() + p.get_width() / 2, p.get_height() + 5,
                f'{value}', ha='center', va='bottom', fontsize=10, fontweight='bold')

plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Step 1: Aggregate gpt_composite by legal_name for Bing and Google
bing_grouped = bing_data_gpt.groupby('legal_name')['gpt_composite'].mean().reset_index()
google_grouped = google_data_gpt.groupby('legal_name')['gpt_composite'].mean().reset_index()

# Step 2: Merge the datasets on 'legal_name'
merged_data = pd.merge(bing_grouped, google_grouped, on='legal_name', how='outer',
suffixes=('_bing', '_google'))
merged_data.fillna(0, inplace=True)

# Step 3: Calculate discrepancies
```

```
merged_data['discrepancy'] = abs(merged_data['gpt_composite_bing'] - merged_data['gpt_composite_google'])
top_discrepancies = merged_data.sort_values(by='discrepancy', ascending=False).head(20)

# Step 4: Visualization
plt.figure(figsize=(12, 6))
pos = np.arange(len(top_discrepancies['legal_name']))
width = 0.35

plt.bar(pos - width/2, top_discrepancies['gpt_composite_bing'], width, alpha=0.7, color='blue', label='Bing')
plt.bar(pos + width/2, top_discrepancies['gpt_composite_google'], width, alpha=0.7, color='green', label='Google')

# Annotate bars
for i, (bing, google, diff) in enumerate(zip(top_discrepancies['gpt_composite_bing'],
                                             top_discrepancies['gpt_composite_google'],
                                             top_discrepancies['discrepancy'])):
    plt.text(i - width/2, bing + 0.01, f'{bing:.2f}', ha='center', fontsize=8)
    plt.text(i + width/2, google + 0.01, f'{google:.2f}', ha='center', fontsize=8)
    plt.text(i, max(bing, google) + 0.05, f'Diff: {diff:.2f}', ha='center', fontsize=9, color='red')

plt.xlabel('Company')
plt.ylabel('Average GPT Composite Score')
plt.title('Top 20 Discrepancies in GPT Composite Scores: Bing vs. Google')
plt.xticks(pos, top_discrepancies['legal_name'], rotation=90)
plt.legend()
plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt
import numpy as np

# Analyze links for companies with top discrepancies
discrepancy_analysis = []
```

```
for company in top_discrepancies['legal_name']:
    bing_links = bing_data_gpt[bing_data_gpt['legal_name'] == company]['Link'].unique()
    google_links = google_data_gpt[google_data_gpt['legal_name'] == company]['Link'].unique()
    matched_links = set(bing_links).intersection(set(google_links))

    discrepancy_analysis.append({
        'Company': company,
        'Bing_Links': len(bing_links),
        'Google_Links': len(google_links),
        'Matched_Links': len(matched_links)
    })

discrepancy_df = pd.DataFrame(discrepancy_analysis)

# Plot the results
plt.figure(figsize=(12, 6))
bar_width = 0.25
index = np.arange(len(discrepancy_df))

plt.bar(index, discrepancy_df['Bing_Links'], bar_width, label='Bing Links', color='blue')
plt.bar(index + bar_width, discrepancy_df['Google_Links'], bar_width, label='Google Links', color='green')
bars_matched = plt.bar(index + 2 * bar_width, discrepancy_df['Matched_Links'], bar_width, label='Matched Links', color='orange')

# Annotate matched links
for bar, value in zip(bars_matched, discrepancy_df['Matched_Links']):
    plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height() + 0.5,
             f'{value}', ha='center', fontsize=10, color='red')

plt.xlabel('Company')
plt.ylabel('Number of Links')
plt.title('Link Comparison for Top Discrepancies (GPT Composite Scores)')
plt.xticks(index + bar_width, discrepancy_df['Company'], rotation=90)
plt.legend()
```

```
plt.tight_layout()
plt.show()

# Relaxed condition for no discrepancies
threshold = 0.01
no_discrepancy_data = merged_data[abs(merged_data['gpt_composite_bing'] - merged_data['gpt_composite_google']) < threshold]

no_discrepancy_analysis = []
for company in no_discrepancy_data['legal_name']:
    bing_links = bing_data_gpt[bing_data_gpt['legal_name'] == company]['Link'].unique()
    google_links = google_data_gpt[google_data_gpt['legal_name'] == company]['Link'].unique()
    matched_links = set(bing_links).intersection(set(google_links))

    no_discrepancy_analysis.append({
        'Company': company,
        'Bing_Links': len(bing_links),
        'Google_Links': len(google_links),
        'Matched_Links': len(matched_links)
    })

no_discrepancy_df = pd.DataFrame(no_discrepancy_analysis)

if not no_discrepancy_df.empty:
    plt.figure(figsize=(12, 6))
    bar_width = 0.3
    index = np.arange(len(no_discrepancy_df))

    plt.bar(index, no_discrepancy_df['Bing_Links'], bar_width, label='Bing Links', color='blue')
    plt.bar(index + bar_width, no_discrepancy_df['Google_Links'], bar_width, label='Google Links', color='green')
    bars_matched = plt.bar(index + 2 * bar_width, no_discrepancy_df['Matched_Links'], bar_width, label='Matched Links', color='orange')

    for bar, value in zip(bars_matched, no_discrepancy_df['Matched_Links']):
```

```
        plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height() + 0.5, f'{value}', ha='center', color='red')

    plt.xlabel('Company')
    plt.ylabel('Number of Links')
    plt.title('Link Comparison for Companies with No Discrepancies (GPT Composite Scores)')
    plt.xticks(index + bar_width, no_discrepancy_df['Company'], rotation=90)
    plt.legend()
    plt.tight_layout()
    plt.show()
else:
    print("No companies found with matching GPT composite scores.")

"""### Rules application (negative news keywords)"""

import pandas as pd
from google.colab import drive

# Step 1: Mount Google Drive
drive.mount('/content/drive')

# Step 2: File paths
base_path = '/content/drive/My Drive/MTD/KYC data/Output/'

files = {
    "google_data": base_path + "googledata.csv",  # Updated with exact name
    "bing_data": base_path + "bingdata.csv",     # Updated with exact name
    "google_data_vaders": base_path + "google_data_vaders.csv",
    "google_data_roberta": base_path + "google_data_roberta.csv",
    "bing_data_vaders": base_path + "bing_data_vaders.csv",
    "bing_data_roberta": base_path + "bing_data_roberta.csv",
    "google_data_bert": base_path + "google_data_bert.csv",
    "bing_data_bert": base_path + "bing_data_bert.csv",
    "gpt_google_sentiment_results": base_path + "gpt_google_sentiment_results.csv",
    "gpt_bing_sentiment_results": base_path + "gpt_bing_sentiment_results.csv"
```

```
}


# Step 3: Load all CSVs into DataFrames
dataframes = {}
for name, path in files.items():
    try:
        dataframes[name] = pd.read_csv(path)
        print(f"Loaded: {name} ({len(dataframes[name])} rows)")
    except Exception as e:
        print(f"Failed to load {name}: {e}")

# Step 4: Verify DataFrames
for name, df in dataframes.items():
    print(f"\n{name} Preview:")
    print(df.head())

# Assign individual variables dynamically for each DataFrame
for name, df in dataframes.items():
    globals()[name] = df
    print(f"DataFrame '{name}' loaded into a variable.")

# Example: Accessing google_data_vaders directly
print(google_data_vaders.head())

# Debug: List all files in the directory
print("Files in the directory:")
for f in os.listdir(base_path):
    print(f)

# Print all loaded DataFrames with their names
for name, df in dataframes.items():
    print(f"\nDataFrame: {name}")
    print(df.head())  # Print the first few rows of each DataFrame
    print(f"Shape: {df.shape}")  # Optionally, print the number of rows and columns
```

```
import pandas as pd

# Assuming google_data is already loaded, otherwise load it as needed
# google_data = pd.read_csv('path_to_your_file.csv')

# Make a copy of the DataFrame
google_data_copy = google_data.copy()

# Define the keyword checking function
def check_keywords(row):
    # List of keywords
    keywords = [
        "corruption", "corrupt", "corrupted", "corrupting",
        "fraud", "fraudulent", "defraud", "defrauding",
        "bribery", "bribe", "bribing", "bribed",
        "laundering", "launder", "laundered", "money launder",
        "trading", "trade", "traded", "insider trading",
        "tax", "taxing", "taxed", "tax evasion",
        "avoiding", "avoid", "avoided", "evasion",
        "fixing", "fix", "fixed", "rigging",
        "violations", "violate", "violated", "violating",
        "bankruptcy", "bankrupt", "bankrupted", "bankrupting",
        "insolvency", "insolvent",
        "misconduct", "misconducting", "misconducted",
        "accounting", "account", "accounted", "misaccounting",
        "ponzi", "ponzi scheme",
        "scheme", "scheming", "schemed",
        "scams", "scam", "scammed", "scamming",
        "labor", "labored", "laboring", "labour",
        "harassment", "harass", "harassed", "harassing",
        "unethical", "unethically",
        "conflict", "conflicting", "conflicted",
        "nepotism", "nepotistic",
        "whistleblower", "whistleblowing", "whistleblowed",
```

"sanctions", "sanction", "sanctioned", "sanctioning",
"issues", "issue", "issued", "issuing",
"failures", "failure", "failed", "failing",
"theft", "thief", "stealing", "stole",
"breach", "breached", "breaching",
"scandal", "scandalous",
"damage", "damaged", "damaging",
"negative", "negativity", "negatively",
"controversy", "controversial", "controversially",
"embezzlement", "embezzle", "embezzled", "embezzling",
"kickback", "kickbacks", "kickbacked",
"extortion", "extort", "extorted", "extorting",
"racketeering", "racketeer", "racketeered",
"malfeasance", "misfeasance",
"misappropriation", "misappropriate", "misappropriated", "misappropriating",
"exploitation", "exploit", "exploited", "exploiting",
"dishonesty", "dishonest", "dishonestly",
"deception", "deceive", "deceived", "deceiving",
"manipulation", "manipulate", "manipulated", "manipulating",
"coercion", "coerce", "coerced", "coercing",
"abuse of power", "abuse", "abused", "abusing",
"ethical breach", "ethics breach", "ethical violation",
"conflict of interest", "conflicts of interest",
"financial impropriety", "financially improper",
"illegal activity", "illegality", "illegalities",
"regulatory violation", "regulate", "regulated", "regulating",
"fiscal misconduct", "fiscal misdeed", "fiscally irresponsible",
"unlawful", "unlawfulness", "unlawfully",
"illicit", "illicitly",
"unscrupulous", "unscrupulously"
"collusion", "collude", "colluding",
"conspiracy", "conspire", "conspiring",
"falsification", "falsify", "falsifying",
"misrepresentation", "misrepresent", "misrepresenting",
"price fixing", "price rigging",

```
        "unfair practice", "unfair practices",
        "unauthorized", "unauthorized access", "unauthorized use",
        "unethical practice", "unethical practices",
        "whistle-blowing", "whistleblow", "whistleblower",
        "wrongdoing", "wrongdoings", "wrongful", "wrongfully",

        # Aviation-specific negative terms
        "air safety violation", "safety violation", "safety violations",
        "aircraft maintenance negligence", "maintenance negligence",
        "aviation fraud", "aviation scam",
        "bribery in aviation", "aviation bribery",
        "cargo theft", "aviation cargo theft",
        "counterfeit aircraft parts", "counterfeit parts",
        "flight risk", "flight risks",
        "grounding aircraft", "aircraft grounding",
        "illegal flight operation", "illegal operations",
        "pilot intoxication", "pilot negligence",
        "regulatory non-compliance", "non-compliance",
        "safety non-compliance", "safety negligence",
        "skyjacking", "aircraft hijacking", "hijacking",
        "smuggling via aircraft", "aircraft smuggling",
        "terrorist threat", "aviation terrorist threat",
        "unauthorized airspace entry", "airspace violation",
        "unsafe flying practices", "unsafe practices"
]

    # Check FullText first, then fallback to Snippet if FullText is blank, null, or 'None'
    text = row['FullText'] if pd.notna(row['FullText']) and row['FullText'].strip() not in ['', 'None'] else
row['Snippet']

    # Converting the text to lowercase for case-insensitive matching
    text_lower = text.lower()

    # Initialize a list to hold matched keywords
    matched_keywords = []
```

```
    # Check if any keyword is in the text and add to matched_keywords
    for keyword in keywords:
        if keyword in text_lower:
            matched_keywords.append(keyword)

    # Return matched keywords as a string if any, otherwise return an empty string
    return ', '.join(matched_keywords) if matched_keywords else ''

# Apply the function to each row of the copied DataFrame
google_data_copy['Keywords_Matched'] = google_data_copy.apply(check_keywords, axis=1)

# Add a new column to count the number of matched keywords
google_data_copy['Keyword_Count'] = google_data_copy['Keywords_Matched'].apply(lambda x:
len(x.split(', ')) if x else 0)

import pandas as pd

# File paths
files = {
    "google_search_results": "google_search_results.csv",
    "bing_search_results": "bing_search_results.csv",
    "google_data_vaders": "google_data_vaders.csv",
    "google_data_roberta": "google_data_roberta.csv",
    "bing_data_vaders": "bing_data_vaders.csv",
    "bing_data_roberta": "bing_data_roberta.csv",
    "google_data_bert": "google_data_bert.csv",
    "bing_data_bert": "bing_data_bert.csv",
    "gpt_google_sentiment_results": "gpt_google_sentiment_results.csv",
    "gpt_bing_sentiment_results": "gpt_bing_sentiment_results.csv"
}

# Load all CSVs into DataFrames
dataframes = {}
for name, path in files.items():
```

```
    try:
        dataframes[name] = pd.read_csv(path)
        print(f"Loaded: {name} ({len(dataframes[name])} rows)")
    except Exception as e:
        print(f"Failed to load {name}: {e}")

# Access DataFrames as dataframes['name']

"""### OFAC Sanctions list"""

import requests
import xml.etree.ElementTree as ET
import pandas as pd

# Fetch the XML data
url = "https://www.treasury.gov/ofac/downloads/sdn.xml"
response = requests.get(url)
xml_data = response.content

# Parse the XML data and get the namespace dynamically
tree = ET.fromstring(xml_data)
namespace = tree.tag.split('}')[0].strip('{')  # Extract namespace dynamically

# Define the namespace
ns = {'ns': namespace}

# Extract data from each sdnEntry
data = []
for sdn_entry in tree.findall('ns:sdnEntry', ns):
    entry_data = {}
    for child in sdn_entry:
        tag = child.tag.split('}')[-1]  # Remove namespace prefix
        entry_data[tag] = child.text.strip() if child.text else None
    data.append(entry_data)
```

```
# Convert to DataFrame
ofac = pd.DataFrame(data)
print(ofac.head())

import pandas as pd

# Correct path in Google Drive
add_file_path = '/content/drive/My Drive/MTD/SDN/add.csv'

# Define the column names
column_names = ['sdnEntry', 'addressID', 'address1', 'city', 'country', 'placeholder']

# Load the CSV into a DataFrame with specified column names
add = pd.read_csv(add_file_path, names=column_names)
add

import pandas as pd

# Specify the path to the "aka.csv" file
aka_file_path = r'/content/drive/My Drive/MTD/SDN/alt.csv'

# Define the column names
column_names = ['sdnEntry', 'akaID', 'akaType', 'alternateName', 'placeholder']

# Load the CSV into a DataFrame with specified column names
aka = pd.read_csv(aka_file_path, names=column_names)
aka

import pandas as pd

# Specify the path to the "sdn_comments.csv" file
comments_file_path = r'/content/drive/My Drive/MTD/SDN/sdn_comments.csv'

# Define the column names
column_names = ['sdnEntry', 'comments']
```

```
# Load the CSV into a DataFrame with specified column names
comments = pd.read_csv(comments_file_path, names=column_names, header=None)
comments

# Join the 'add' DataFrame
ofac = pd.merge(ofac, add[['sdnEntry', 'address1', 'city', 'country']],
                left_on='uid', right_on='sdnEntry', how='left')

# Join the 'aka' DataFrame
ofac = pd.merge(ofac, aka[['sdnEntry', 'alternateName']],
                left_on='uid', right_on='sdnEntry', how='left')

# Join the 'comments' DataFrame
ofac = pd.merge(ofac, comments[['sdnEntry', 'comments']],
                left_on='uid', right_on='sdnEntry', how='left')

# Optionally, you might want to drop the duplicate 'sdnEntry' columns after the merge
ofac = ofac.drop(columns=['sdnEntry_x', 'sdnEntry_y', 'sdnEntry'])
ofac

# Number of unique values in each column
unique_values = ofac.nunique()

# Identifying completely empty columns (all values are NaN)
empty_columns = ofac.isna().all()

# Display the results
print("Unique values in each column:\n", unique_values)
print("\nCompletely empty columns:\n", empty_columns[empty_columns == True])

# Columns with only 1 unique value
columns_with_one_unique_value = unique_values[unique_values == 1].index

# Display the unique value for each of these columns
```

```
for column in columns_with_one_unique_value:
    print(f"Unique value in '{column}': {ofac[column].unique()}")

# List of columns to be dropped
columns_to_drop = ['programList', 'akaList', 'addressList', 'idList', 'dateOfBirthList',
                   'placeOfBirthList', 'nationalityList', 'vesselInfo', 'citizenshipList']

# Dropping the columns from the DataFrame
ofac = ofac.drop(columns=columns_to_drop)

for col in ofac.columns:
    if ofac[col].dtype == 'object':  # Checking if the column is of string type
        ofac[col] = ofac[col].map(lambda x: x.strip() if isinstance(x, str) else x)

ofac = ofac.replace('-0-', '')  # Replacing '-0-' with empty strings
ofac = ofac.fillna('')  # Replacing NaN values with empty strings

ofac

"""### UN Sanctions list"""

import xml.etree.ElementTree as ET
import pandas as pd

# Path to your downloaded XML file
file_path = r'/content/drive/My Drive/MTD/xml/consolidated.xml'

# Parse the XML file
tree = ET.parse(file_path)
root = tree.getroot()

# Extract data
data = []

# Process INDIVIDUAL entries
```

```
for individual in root.findall('.//INDIVIDUALS/INDIVIDUAL'):
    entry_data = {}
    for attribute in individual:
        entry_data[attribute.tag] = attribute.text
    data.append(entry_data)

# Process ENTITY entries
for entity in root.findall('.//ENTITIES/ENTITY'):
    entry_data = {}
    for attribute in entity:
        entry_data[attribute.tag] = attribute.text
    data.append(entry_data)

# Convert to DataFrame
un_sanctions = pd.DataFrame(data)

# Replace NaN values with empty strings
un_sanctions = un_sanctions.fillna('')

un_sanctions

"""### EU Sanction list"""

import xml.etree.ElementTree as ET
import pandas as pd

# Path to your downloaded XML file
file_path = r'/content/drive/My Drive/MTD/xml/eulist.xml'

# Parse the XML file
tree = ET.parse(file_path)
root = tree.getroot()

# Extract data
data = []
```

```
ns = {'ns': 'http://eu.europa.ec/fpi/fsd/export'}  # Namespace

for entity in root.findall('ns:sanctionEntity', ns):
    entry_data = {
        'euReferenceNumber': entity.get('euReferenceNumber', ''),
        'remark': entity.findtext('ns:remark', default='', namespaces=ns),
        'nameAliases': [],
        'birthDates': [],
        'citizenships': []
    }

    # Extract the first regulation for programme and legalBasis
    regulation = entity.find('ns:regulation', ns)
    if regulation is not None:
        entry_data['programme'] = regulation.get('programme', '')
        entry_data['legalBasis'] = regulation.get('numberTitle', '')
        entry_data['publicationUrl'] = regulation.findtext('ns:publicationUrl', default='',
namespaces=ns)

    # Extract multiple name aliases
    for name_alias in entity.findall('ns:nameAlias', ns):
        alias = name_alias.get('wholeName', '')
        if alias:
            entry_data['nameAliases'].append(alias)

    # Extract multiple birth dates and places
    for birthdate in entity.findall('ns:birthdate', ns):
        birth_info = f"{birthdate.get('birthdate', '')}, {birthdate.get('city', '')}"
        if birth_info.strip(', '):
            entry_data['birthDates'].append(birth_info)

    # Extract multiple citizenships
    for citizenship in entity.findall('ns:citizenship', ns):
        country = citizenship.get('countryDescription', '')
        if country:
```

```
            entry_data['citizenships'].append(country)

    data.append(entry_data)

# Convert to DataFrame
eu_sanctions = pd.DataFrame(data)
eu_sanctions

"""### Press releases for training ML model

### EASA
"""

import requests
from bs4 import BeautifulSoup
import pandas as pd
import re

def clean_text(text):
    # Normalize whitespace and newlines
    text = re.sub(r'\s+', ' ', text).strip()
    return text

def easa_scrape_articles(base_url, page_path):
    response = requests.get(base_url + page_path)
    soup = BeautifulSoup(response.text, 'html.parser')

    item_list = soup.find('div', class_='item-list')
    article_h3_tags = item_list.find_all('h3') if item_list else []

    article_links = [base_url + h3.find('a')['href'] for h3 in article_h3_tags]

    data_for_df = []

    for link in article_links:
```

```
        article_response = requests.get(link)
        article_soup = BeautifulSoup(article_response.text, 'html.parser')

        paragraphs = article_soup.find_all('p')
        relevant_paragraphs = []

        for p in paragraphs:
            if "About European Union Aviation Safety Agency - EASA" in p.get_text():
                break
            relevant_paragraphs.append(p.get_text())

        text = ' '.join(relevant_paragraphs)
        text = clean_text(text)  # Clean the text
        data_for_df.append({'source': 'EASA', 'link': link, 'text': text})

    return data_for_df

# Base URL and page path
base_url = 'https://www.easa.europa.eu'
page_path = '/en/newsroom-and-events/press-releases?page=1'

# Scrape the articles
data = easa_scrape_articles(base_url, page_path)

# Creating a DataFrame
easa_df = pd.DataFrame(data)

# Resetting the index
easa_df.reset_index(inplace=True)

easa_df

"""### FAA"""

import requests
```

```
from bs4 import BeautifulSoup
import pandas as pd
import re

def remove_unwanted_text(text):
    patterns = [
        r"Federal Aviation Administration\s+800 Independence Avenue, SW\s+Washington, DC 20591\s+866\.835\.5322 \(866-TELL-FAA\)Contact Us",
        r"If you are deaf, hard of hearing, or have a speech disability, please dial 7-1-1 to access telecommunications relay services\.",
        r"Federal Aviation Administration Press Office\s+800 Independence Avenue, SW\s+Washington, DC 20591\s+United States\s+800 Independence Avenue, SW\s+Washington, DC 20591\s+United States\s+Email:pressoffice@faa\.gov",
        r"An official website of the United States government Here's how you know Official websites use \.govA \.gov website belongs to an official government organization in the United States\. Secure \.gov websites use HTTPS\s+A lock \( LockA locked padlock \) or https:// means you've safely connected to the \.gov website\. Share sensitive information only on official, secure websites\."
    ]
    for pattern in patterns:
        text = re.sub(pattern, '', text)

    # Replace HTML entities
    text = re.sub(r'&[a-zA-Z0-9#]+;', ' ', text)

    # Normalize whitespace
    text = re.sub(r'\s+', ' ', text).strip()

    return text

def faa_scrape_articles(base_url, page_path):
    response = requests.get(base_url + page_path)
    soup = BeautifulSoup(response.text, 'html.parser')

    article_divs = soup.find_all('div', class_='views-row')
    data_for_df = []
```

```
    for div in article_divs:
        link_tag = div.find('a')
        if link_tag and 'href' in link_tag.attrs:
            link = base_url + link_tag['href']

            article_response = requests.get(link)
            article_soup = BeautifulSoup(article_response.text, 'html.parser')
            paragraphs = article_soup.find_all('p')
            text = ' '.join(p.get_text() for p in paragraphs)

            # Remove the unnecessary text
            text = remove_unwanted_text(text)

            data_for_df.append({'source': 'FAA', 'link': link, 'text': text})

    return data_for_df

# Base URL and page path
base_url = 'https://www.faa.gov'
page_path = '/newsroom/press_releases'

# Scrape the articles
data = faa_scrape_articles(base_url, page_path)

# Creating a DataFrame
faa_df = pd.DataFrame(data)

# Resetting the index
faa_df.reset_index(inplace=True)

faa_df

"""### ICAO"""
```

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
import re

def clean_icao_text(text):
    text = re.sub(r'\s+', ' ', text).strip()
    return text

def icao_scrape_articles(base_url, page_path):
    response = requests.get(base_url + page_path)
    soup = BeautifulSoup(response.text, 'html.parser')

    dfwp_items = soup.find_all('li', class_='dfwp-item')
    data_for_df = []

    for item in dfwp_items:
        link_tag = item.find('a')
        if link_tag and 'href' in link_tag.attrs:
            link = link_tag['href']

            article_response = requests.get(link)
            article_soup = BeautifulSoup(article_response.text, 'html.parser')
            paragraphs = article_soup.find_all('p')
            text = ' '.join(p.get_text() for p in paragraphs)

            text = clean_icao_text(text)

            data_for_df.append({'source': 'ICAO', 'link': link, 'text': text})

    return data_for_df

def check_and_scrape_additional(base_url, current_data, page_number):
    if any(not row['text'] for row in current_data):
```

```
        additional_data = icao_scrape_articles(base_url,
f'/Newsroom/Pages/pressrelease.aspx?p={page_number}&year=2023')
        return current_data + additional_data
    else:
        return current_data

# Base URL and initial page path
base_url = 'https://www.icao.int'
initial_page_path = '/Newsroom/Pages/pressrelease.aspx?p=1&year=2023'

# Scrape the first page
data = icao_scrape_articles(base_url, initial_page_path)

# Check and possibly scrape the second page
data = check_and_scrape_additional(base_url, data, 2)

# Check and possibly scrape the third page
data = check_and_scrape_additional(base_url, data, 3)

# Creating a DataFrame
icao_df = pd.DataFrame(data)

# Function to check if the text is effectively empty
def is_effectively_empty(text):
    # Check for the presence of any alphanumeric characters
    return not any(char.isalnum() for char in text)

# Apply this function to filter out rows with effectively empty 'text'
icao_df = icao_df[~icao_df['text'].apply(is_effectively_empty)]

# Resetting the index after filtering
icao_df.reset_index(drop=True, inplace=True)

icao_df
```

```
"""--- Check text

# Training ML
"""

import nltk
nltk.download('vader_lexicon')  # Download the VADER lexicon
from nltk.sentiment.vader import SentimentIntensityAnalyzer

"""### VADER"""

import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Assuming easa_df, faa_df, icao_df, ofac, un_sanctions, eu_sanctions are preloaded dataframes

# Merge easa_df, faa_df, and icao_df vertically
merged_df = pd.concat([easa_df, faa_df, icao_df])

# Preprocessing
merged_df['cleaned_text'] = merged_df['text'].str.lower().str.replace('[^\w\s]', '', regex=True)

# TF-IDF Vectorization for aviation-related data
vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = vectorizer.fit_transform(merged_df['cleaned_text'])
feature_names = vectorizer.get_feature_names_out()

# Convert to DataFrame
dense = tfidf_matrix.todense()
denselist = dense.tolist()
df_tfidf = pd.DataFrame(denselist, columns=feature_names)

# Initialize VADER
sid = SentimentIntensityAnalyzer()
```

```
# Function to add names from sanctions lists to VADER lexicon
def add_sanctions_to_vader_lexicon(df_list, columns_list, negative_score):
    for df, col in zip(df_list, columns_list):
        # Ensure each name is processed as a string
        names = df[col].astype(str).dropna().unique()
        for name in names:
            sid.lexicon[name.lower()] = negative_score

# Define sanctions dataframes and their relevant columns
sanctions_dfs = [ofac, un_sanctions, eu_sanctions]
sanctions_columns = ['lastName', 'FIRST_NAME', 'nameAliases']  # Adjust these column names as needed
add_sanctions_to_vader_lexicon(sanctions_dfs, sanctions_columns, -4.0)

# Sentiment Analysis for aviation data
# Dictionary to store sentiment scores
keyword_sentiments = {}

for keyword in df_tfidf.columns:
    scores = sid.polarity_scores(keyword)
    keyword_sentiments[keyword] = scores['compound']  # Using compound score

# Calculate average TF-IDF score for each term
avg_tfidf_scores = df_tfidf.mean(axis=0)

# Normalize TF-IDF scores
normalized_tfidf = (avg_tfidf_scores - avg_tfidf_scores.min()) / (avg_tfidf_scores.max() - avg_tfidf_scores.min())

# Weight VADER sentiment scores by normalized TF-IDF scores
weighted_scores = {keyword: keyword_sentiments[keyword] * (1 + normalized_tfidf[keyword])
                   for keyword in keyword_sentiments}

# Find the maximum absolute value among the weighted scores
```

```
max_abs_weighted_score = max(abs(score) for score in weighted_scores.values())

# Function to scale scores
def scale_score(value, max_abs_val, scale_range):
    return (value / max_abs_val) * scale_range if max_abs_val != 0 else 0

# Scale each score to fit within -4 to +4, preserving the original sign
rescaled_scores = {keyword: scale_score(score, max_abs_weighted_score, 4)
                   for keyword, score in weighted_scores.items()}

# Now, rescaled_scores contains the final sentiment scores for each keyword
rescaled_scores

# Number of keywords in rescaled_scores
num_keywords = len(rescaled_scores)
print("Number of keywords in rescaled_scores:", num_keywords)

def preprocess_text(df, text_column):
    """
    Preprocess the text column by:
    - Converting to lowercase
    - Removing punctuation and non-word characters
    - Dropping rows with empty or invalid text
    """
    # Convert the column to string type before applying string operations
    df[text_column] = df[text_column].astype(str)
    # Preprocess the text
    processed_text = df[text_column].str.lower().str.replace('[^\w\s]', '', regex=True)
    # Filter rows with empty text after preprocessing
    processed_text = processed_text[processed_text.str.strip() != '']
    return processed_text

def count_unique_terms(df, text_column):
    """
    Vectorize text and count unique terms using TF-IDF.
```

```
    If the vocabulary is empty, return 0.
    """
    vectorizer = TfidfVectorizer(stop_words='english')

    # Preprocess the text and filter valid rows
    cleaned_text = preprocess_text(df, text_column)

    # Check if cleaned text has any valid data
    if cleaned_text.empty:
        print(f"No valid text in column '{text_column}'. Returning 0.")
        return 0, []

    # Vectorize the text
    tfidf_matrix = vectorizer.fit_transform(cleaned_text)
    feature_names = vectorizer.get_feature_names_out()
    return len(feature_names), feature_names

# Count unique terms in each dataframe
easa_keywords_count, easa_keywords = count_unique_terms(easa_df, 'text')
faa_keywords_count, faa_keywords = count_unique_terms(faa_df, 'text')
icao_keywords_count, icao_keywords = count_unique_terms(icao_df, 'text')
ofac_keywords_count, ofac_keywords = count_unique_terms(ofac, 'lastName')  # Assuming 'lastName' is the text column
un_sanctions_keywords_count, un_sanctions_keywords = count_unique_terms(un_sanctions, 'FIRST_NAME')  # Similarly for others
eu_sanctions_keywords_count, eu_sanctions_keywords = count_unique_terms(eu_sanctions, 'nameAliases')

# Print the counts
print("EASA keywords count:", easa_keywords_count)
print("FAA keywords count:", faa_keywords_count)
print("ICAO keywords count:", icao_keywords_count)
print("OFAC keywords count:", ofac_keywords_count)
print("UN Sanctions keywords count:", un_sanctions_keywords_count)
print("EU Sanctions keywords count:", eu_sanctions_keywords_count)
```

```
print("EASA Preprocessed Text Sample:")
print(preprocess_text(easa_df, 'text').head())

print("FAA Preprocessed Text Sample:")
print(preprocess_text(faa_df, 'text').head())

from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Initialize VADER
sid = SentimentIntensityAnalyzer()

# Get the number of keywords in the basic VADER lexicon
basic_vader_lexicon_count = len(sid.lexicon)
print("Number of keywords in basic VADER lexicon:", basic_vader_lexicon_count)

# Assuming 'rescaled_scores' contains your custom sentiment scores
# For example: rescaled_scores = {'aviation_term1': -2.0, 'aviation_term2': 1.5, ...}

# Update VADER lexicon with rescaled scores
sid.lexicon.update(rescaled_scores)

# Count the total number of entries in the updated lexicon
updated_lexicon_count = len(sid.lexicon)
print("Number of entries in the updated VADER lexicon:", updated_lexicon_count)

# Initialize SentimentIntensityAnalyzer with custom lexicon
custom_sia = SentimentIntensityAnalyzer()
custom_sia.lexicon.update(rescaled_scores)  # Assuming 'rescaled_scores' is your custom lexicon

from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Initialize VADER
sid = SentimentIntensityAnalyzer()
```

```
# Get the number of entries in the default VADER lexicon
basic_vader_lexicon_count = len(sid.lexicon)
print("Number of entries in the basic VADER lexicon:", basic_vader_lexicon_count)

"""### VADERS - GOOGLE new lexicon"""

import pandas as pd
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from tqdm import tqdm
import time

# Initialize VADER SentimentIntensityAnalyzer
sia = SentimentIntensityAnalyzer()

# Initialize results
custom_res = []
custom_source_indicator = []

# Function to truncate long text from the end
def truncate_text(text, max_length=1000):
    if pd.notnull(text) and len(text) > max_length:
        return text[:max_length]
    return text

# Start the timer
start_time = time.time()

# Process rows in google_data
for i, row in tqdm(google_data.iterrows(), total=google_data.shape[0]):
    # Extract and truncate text
    text = row['FullText'] if pd.notnull(row['FullText']) and row['FullText'] != "None" else row['Snippet']
    text = truncate_text(text) if pd.notnull(text) and text != "None" else ""

    # If valid text exists, calculate sentiment; otherwise, assign neutral scores
```

```
    if text:
        scores = sia.polarity_scores(text)
        source_used = 'FullText' if text == row['FullText'] else 'Snippet'
    else:
        scores = {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}  # Default neutral scores
        source_used = 'None'

    # Append results
    custom_res.append({
        'vaders_link': row['Link'],
        'vaders_new_neg': scores['neg'],
        'vaders_new_neu': scores['neu'],
        'vaders_new_pos': scores['pos'],
        'vaders_new_compound': scores['compound']
    })
    custom_source_indicator.append(source_used)

# End the timer
end_time = time.time()

# Calculate elapsed time
elapsed_time = end_time - start_time
print(f"Time spent on VADER sentiment analysis: {elapsed_time:.2f} seconds")

# Convert results to DataFrame
gvaders_new = pd.DataFrame(custom_res)

# Add SourceUsed column
gvaders_new['SourceUsed'] = custom_source_indicator

# Calculate composite score
gvaders_new['vaders_new_composite'] = gvaders_new['vaders_new_pos'] -
gvaders_new['vaders_new_neg']

# Ensure all rows are accounted for
```

```
print(f"Rows in google_data: {len(google_data)}")
print(f"Rows in gvaders_new: {len(gvaders_new)}")

# Display final results
print("Processed VADER Sentiment Analysis:")
display(gvaders_new)

"""### VADERS - BING new lexicon"""

import pandas as pd
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from tqdm import tqdm

# Initialize VADER SentimentIntensityAnalyzer
sia = SentimentIntensityAnalyzer()

# Initialize results for Bing data
custom_res_bing = []
custom_source_indicator_bing = []

# Function to truncate long text from the end
def truncate_text(text, max_length=1000):
    if pd.notnull(text) and len(text) > max_length:
        return text[:max_length]
    return text

# Process rows in bing_data
for i, row in tqdm(bing_data.iterrows(), total=bing_data.shape[0]):
    # Extract and truncate text
    text = row['FullText'] if pd.notnull(row['FullText']) and row['FullText'] != "None" else row['Snippet']
    text = truncate_text(text) if pd.notnull(text) and text != "None" else ""

    # If valid text exists, calculate sentiment; otherwise, assign neutral scores
    if text:
```

```
        scores = sia.polarity_scores(text)
        source_used = 'FullText' if text == row['FullText'] else 'Snippet'
    else:
        scores = {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}  # Default neutral scores
        source_used = 'None'

    # Append results
    custom_res_bing.append({
        'vaders_link': row['Link'],
        'vaders_new_neg': scores['neg'],
        'vaders_new_neu': scores['neu'],
        'vaders_new_pos': scores['pos'],
        'vaders_new_compound': scores['compound']
    })
    custom_source_indicator_bing.append(source_used)

# Convert results to DataFrame
bvaders_new = pd.DataFrame(custom_res_bing)

# Add SourceUsed column
bvaders_new['SourceUsed'] = custom_source_indicator_bing

# Calculate composite score
bvaders_new['vaders_new_composite'] = bvaders_new['vaders_new_pos'] - bvaders_new['vaders_new_neg']

# Ensure all rows are accounted for
print(f"Rows in bing_data: {len(bing_data)}")
print(f"Rows in bvaders_new: {len(bvaders_new)}")

# Display final results
print("Processed VADER Sentiment Analysis for Bing:")
display(bvaders_new)

import pandas as pd
```

```
# Ensure no duplicate rows in gvaders_new and bvaders_new
gvaders_new = gvaders_new.drop_duplicates(subset=['vaders_link'])
bvaders_new = bvaders_new.drop_duplicates(subset=['vaders_link'])

# Ensure the 'Link' column is consistent for merging
bvaders_new['vaders_link'] = bvaders_new['vaders_link'].str.strip()
gvaders_new['vaders_link'] = gvaders_new['vaders_link'].str.strip()
bing_data['Link'] = bing_data['Link'].str.strip()
google_data['Link'] = google_data['Link'].str.strip()

# Merge bvaders_new into bing_data
bing_data_vaders_new = pd.merge(
    bing_data,
    bvaders_new[['vaders_link', 'vaders_new_neg', 'vaders_new_neu', 'vaders_new_pos',
'vaders_new_compound', 'vaders_new_composite', 'SourceUsed']],
    left_on='Link',
    right_on='vaders_link',
    how='left'
)

# Merge gvaders_new into google_data
google_data_vaders_new = pd.merge(
    google_data,
    gvaders_new[['vaders_link', 'vaders_new_neg', 'vaders_new_neu', 'vaders_new_pos',
'vaders_new_compound', 'vaders_new_composite', 'SourceUsed']],
    left_on='Link',
    right_on='vaders_link',
    how='left'
)

# Drop duplicate 'vaders_link' column after merging
bing_data_vaders_new.drop(columns=['vaders_link'], inplace=True)
google_data_vaders_new.drop(columns=['vaders_link'], inplace=True)
```

```
# Fill NaN values for unmatched rows
for col in ['vaders_new_neg', 'vaders_new_neu', 'vaders_new_pos', 'vaders_new_compound', 'vaders_new_composite']:
    bing_data_vaders_new[col] = bing_data_vaders_new[col].fillna(0.0)
    google_data_vaders_new[col] = google_data_vaders_new[col].fillna(0.0)

bing_data_vaders_new['SourceUsed'] = bing_data_vaders_new['SourceUsed'].fillna('None')
google_data_vaders_new['SourceUsed'] = google_data_vaders_new['SourceUsed'].fillna('None')

# Validate row counts
print(f"Row count in google_data: {len(google_data)}")
print(f"Row count in google_data_vaders_new: {len(google_data_vaders_new)}")
print(f"Row count in bing_data: {len(bing_data)}")
print(f"Row count in bing_data_vaders_new: {len(bing_data_vaders_new)}")

import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Add Source column for new data
bing_data_vaders_new['Source'] = 'Bing'
google_data_vaders_new['Source'] = 'Google'

# Combine Bing and Google VADER data
combined_vaders_data = pd.concat([
    bing_data_vaders_new[['vaders_new_composite', 'Source']],
    google_data_vaders_new[['vaders_new_composite', 'Source']]
])

# Drop NaN values and ensure numeric type
combined_vaders_data = combined_vaders_data.dropna(subset=['vaders_new_composite'])
combined_vaders_data['vaders_new_composite'] = pd.to_numeric(
    combined_vaders_data['vaders_new_composite'], errors='coerce'
)
```

```
# Plot boxplot comparison
plt.figure(figsize=(10, 6))
sns.boxplot(x='Source', y='vaders_new_composite', data=combined_vaders_data, palette='coolwarm')

# Customize plot
plt.title('Distribution of VADER Composite Scores: Bing vs. Google')
plt.xlabel('Source')
plt.ylabel('VADER Composite Score')
plt.axhline(0, color='black', linestyle='--', linewidth=1)

plt.show()

import matplotlib.pyplot as plt
import pandas as pd

# Step 1: Aggregate sentiment counts
bing_sentiment_counts = {
    'Negative': (bing_data_vaders_new['vaders_new_neg'] > 0.5).sum(),
    'Neutral': (bing_data_vaders_new['vaders_new_neu'] > 0.5).sum(),
    'Positive': (bing_data_vaders_new['vaders_new_pos'] > 0.5).sum()
}

google_sentiment_counts = {
    'Negative': (google_data_vaders_new['vaders_new_neg'] > 0.5).sum(),
    'Neutral': (google_data_vaders_new['vaders_new_neu'] > 0.5).sum(),
    'Positive': (google_data_vaders_new['vaders_new_pos'] > 0.5).sum()
}

# Combine sentiment data into a DataFrame
sentiment_comparison = pd.DataFrame({
    'Bing': bing_sentiment_counts,
    'Google': google_sentiment_counts
})
```

```
# Step 2: Plot bar chart
ax = sentiment_comparison.plot(kind='bar', figsize=(10, 6), color=['blue', 'green'])

# Customize plot
plt.title('Sentiment Category Comparison: Bing vs. Google')
plt.xlabel('Sentiment Category')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.legend(title='Source')

# Annotate the exact values on the bars
for p in ax.patches:
    value = int(p.get_height())
    if value > 0:
        ax.text(p.get_x() + p.get_width() / 2, p.get_height() + 5,
                f'{value}', ha='center', va='bottom', fontsize=10, fontweight='bold')

plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Aggregate vaders_new_composite by legal_name
bing_grouped =
bing_data_vaders_new.groupby('legal_name')['vaders_new_composite'].mean().reset_index()
google_grouped =
google_data_vaders_new.groupby('legal_name')['vaders_new_composite'].mean().reset_index()

# Merge the two datasets
merged_data = pd.merge(bing_grouped, google_grouped, on='legal_name', how='outer',
suffixes=('_bing', '_google'))
merged_data.fillna(0, inplace=True)
```

```
# Calculate discrepancies
merged_data['discrepancy'] = abs(merged_data['vaders_new_composite_bing'] - merged_data['vaders_new_composite_google'])

# Top 20 discrepancies
top_discrepancies = merged_data.sort_values(by='discrepancy', ascending=False).head(20)

# Plot discrepancies
plt.figure(figsize=(12, 6))
pos = np.arange(len(top_discrepancies['legal_name']))
width = 0.35

plt.bar(pos - width/2, top_discrepancies['vaders_new_composite_bing'], width, color='blue', label='Bing')
plt.bar(pos + width/2, top_discrepancies['vaders_new_composite_google'], width, color='green', label='Google')

plt.xticks(pos, top_discrepancies['legal_name'], rotation=90)
plt.title('Top 20 Discrepancies in VADER Composite Scores: Bing vs. Google')
plt.ylabel('VADER Composite Score')
plt.legend()

plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt
import numpy as np

# Analyze matched links for companies in top discrepancies
discrepancy_analysis = []

for company in top_discrepancies['legal_name']:
    bing_links = bing_data_vaders_new[bing_data_vaders_new['legal_name'] == company]['Link'].unique()
```

```
    google_links = google_data_vaders_new[google_data_vaders_new['legal_name'] == company]['Link'].unique()
    matched_links = set(bing_links).intersection(set(google_links))

    discrepancy_analysis.append({
        'Company': company,
        'Bing_Links': len(bing_links),
        'Google_Links': len(google_links),
        'Matched_Links': len(matched_links)
    })

discrepancy_df = pd.DataFrame(discrepancy_analysis)

# Plot
plt.figure(figsize=(12, 6))
bar_width = 0.25
index = np.arange(len(discrepancy_df))

plt.bar(index, discrepancy_df['Bing_Links'], bar_width, label='Bing Links', color='blue')
plt.bar(index + bar_width, discrepancy_df['Google_Links'], bar_width, label='Google Links', color='green')
plt.bar(index + 2 * bar_width, discrepancy_df['Matched_Links'], bar_width, label='Matched Links', color='orange')

plt.xticks(index + bar_width, discrepancy_df['Company'], rotation=90)
plt.title('Link Comparison for Top Discrepancies')
plt.ylabel('Number of Links')
plt.legend()

plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

```
# Step 1: Relaxed condition for no discrepancies
threshold = 0.01
no_discrepancy_data = merged_data[
    abs(merged_data['vaders_new_composite_bing'] -
merged_data['vaders_new_composite_google']) < threshold
]

# Step 2: Initialize analysis list
no_discrepancy_analysis = []

# Step 3: Analyze links for companies with no discrepancies
for company in no_discrepancy_data['legal_name']:
    # Get Bing and Google links for the company
    bing_links = bing_data_vaders_new[bing_data_vaders_new['legal_name'] ==
company]['Link'].unique()
    google_links = google_data_vaders_new[google_data_vaders_new['legal_name'] ==
company]['Link'].unique()

    # Calculate matched links
    matched_links = set(bing_links).intersection(set(google_links))

    # Append results
    no_discrepancy_analysis.append({
        'Company': company,
        'Bing_Links': len(bing_links),
        'Google_Links': len(google_links),
        'Matched_Links': len(matched_links)
    })

# Step 4: Convert results to DataFrame
no_discrepancy_df = pd.DataFrame(no_discrepancy_analysis)

# Step 5: Plot if data is available
if not no_discrepancy_df.empty:
```

```
    plt.figure(figsize=(12, 6))
    bar_width = 0.3
    index = np.arange(len(no_discrepancy_df))

    # Plot Bing links
    plt.bar(index, no_discrepancy_df['Bing_Links'], bar_width, label='Bing Links', color='blue')

    # Plot Google links
    plt.bar(index + bar_width, no_discrepancy_df['Google_Links'], bar_width, label='Google Links', color='green')

    # Plot Matched links
    bars_matched = plt.bar(index + 2 * bar_width, no_discrepancy_df['Matched_Links'], bar_width, label='Matched Links', color='orange')

    # Annotate matched links
    for bar, value in zip(bars_matched, no_discrepancy_df['Matched_Links']):
        plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height() + 0.5,
                 f'{value}', ha='center', va='bottom', fontsize=10, fontweight='bold', color='red')

    # Step 6: Customize plot
    plt.xlabel('Company', fontsize=12)
    plt.ylabel('Number of Links', fontsize=12)
    plt.title('Link Comparison for Companies with No Discrepancies (VADER Composite Scores)')
    plt.xticks(index + bar_width, no_discrepancy_df['Company'], rotation=90)
    plt.legend()

    plt.tight_layout()
    plt.show()
else:
    print("No companies found with matching VADER composite scores.")

"""### ROBERTA"""

import pandas as pd
```

```
# Combine text from easa_df, faa_df, and icao_df
aviation_texts = pd.concat([easa_df['text'], faa_df['text'], icao_df['text']], ignore_index=True)

# Prepare text from sanctions dataframes
ofac_texts = ofac['lastName'].astype(str)
un_sanctions_texts = un_sanctions['FIRST_NAME'].str.cat(un_sanctions[['SECOND_NAME', 'THIRD_NAME']], sep=' ', na_rep='').astype(str)
eu_sanctions_texts = eu_sanctions['nameAliases'].astype(str)

# Combine all texts
all_texts = pd.concat([aviation_texts, ofac_texts, un_sanctions_texts, eu_sanctions_texts], ignore_index=True)

# Basic cleaning and preprocessing
all_texts = all_texts.str.lower().str.replace('[^\w\s]', '', regex=True)

from transformers import AutoTokenizer, AutoModelForSequenceClassification
from scipy.special import softmax
import torch

# Load RoBERTa model and tokenizer
MODEL = "cardiffnlp/twitter-roberta-base-sentiment"
tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModelForSequenceClassification.from_pretrained(MODEL)

def get_roberta_sentiment(text):
    # Tokenize and encode the text for RoBERTa
    encoded_input = tokenizer(text, return_tensors='pt', truncation=True, max_length=512, padding=True)

    # Get model predictions
    with torch.no_grad():
        outputs = model(**encoded_input)
```

```
    # Calculate softmax to get probabilities
    scores = softmax(outputs.logits.detach().numpy()[0])

    # We assume the order of sentiments is [negative, neutral, positive]
    sentiment = 'negative' if scores[0] > scores[2] else 'positive' if scores[2] > scores[0] else 'neutral'
    return sentiment

# Apply RoBERTa labeling to aviation_texts
aviation_sentiments = aviation_texts.apply(get_roberta_sentiment)

# Label sanctions texts with 'negative'
sanctions_sentiments = pd.Series(['negative'] * (len(ofac_texts) + len(un_sanctions_texts) + len(eu_sanctions_texts)))

# Combine the sentiments from both aviation and sanctions texts
combined_sentiments = pd.concat([aviation_sentiments, sanctions_sentiments], ignore_index=True)

# Create a DataFrame from the texts and their corresponding sentiment labels
training_data = pd.DataFrame({
    'text': all_texts,
    'label': combined_sentiments
})

# Encoding labels: negative -> 0, neutral -> 1, positive -> 2
label_mapping = {'negative': 0, 'neutral': 1, 'positive': 2}
training_data['label'] = training_data['label'].map(label_mapping)

# Shuffle the DataFrame
training_data = training_data.sample(frac=1).reset_index(drop=True)

from sklearn.model_selection import train_test_split

# Split the dataset into training and validation sets
train_df, val_df = train_test_split(training_data, test_size=0.2, random_state=42, stratify=training_data['label'])
```

```
from transformers import RobertaTokenizer
from torch.utils.data import Dataset
import torch

class SentimentDataset(Dataset):
    def __init__(self, dataframe, tokenizer):
        self.encodings = tokenizer(dataframe['text'].tolist(), truncation=True, padding=True, max_length=512)
        self.labels = dataframe['label'].tolist()

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

# Load the tokenizer
tokenizer = RobertaTokenizer.from_pretrained('roberta-base')

# Create dataset objects
train_dataset = SentimentDataset(train_df, tokenizer)
val_dataset = SentimentDataset(val_df, tokenizer)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
# When loading data, also move it to the device

!nvidia-smi

!pip install --upgrade transformers
!pip install --upgrade torch
!pip install --upgrade accelerate
```

```
!pip install --upgrade pip
!pip install transformers[torch] accelerate -U

import torch
print(f"CUDA Available: {torch.cuda.is_available()}")
print(f"GPU Memory Allocated: {torch.cuda.memory_allocated() / 1e6} MB")
print(f"GPU Memory Cached: {torch.cuda.memory_reserved() / 1e6} MB")

from transformers import RobertaForSequenceClassification, Trainer, TrainingArguments
import torch

# Load pre-trained RoBERTa model for sequence classification
model = RobertaForSequenceClassification.from_pretrained('roberta-base', num_labels=3)

# Move model to GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# Training arguments
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    evaluation_strategy="epoch"
)

# Initialize Trainer
trainer = Trainer(
    model=model,
    args=training_args,
```

```
    train_dataset=train_dataset,
    eval_dataset=val_dataset
)

# Train the model
trainer.train()

"""**Applying trained roBERTa model**"""

def get_sentiment_scores(row, model, tokenizer):
    if row['FullText'] == "None" or pd.isna(row['FullText']):
        text = row['Snippet']
        source = 'Snippet'
    else:
        text = row['FullText']
        source = 'FullText'

    inputs = tokenizer.encode_plus(
        text,
        add_special_tokens=True,
        max_length=512,
        truncation=True,
        return_tensors='pt'
    ).to(model.device)  # Ensure inputs are on the same device as the model

    outputs = model(**inputs)
    scores = softmax(outputs.logits.detach().cpu().numpy()[0])  # Move outputs to CPU for numpy operations

    return scores, source

# Create a copy of the dataframes with necessary columns
groberta_new = google_data.filter(['Link', 'Snippet', 'FullText']).copy()
broberta_new = bing_data.filter(['Link', 'Snippet', 'FullText']).copy()
```

```
# Apply the function to each dataframe
groberta_new[['SentimentScores', 'roberta_source']] = groberta_new.apply(lambda row: get_sentiment_scores(row, model, tokenizer), axis=1, result_type='expand')
broberta_new[['SentimentScores', 'roberta_source']] = broberta_new.apply(lambda row: get_sentiment_scores(row, model, tokenizer), axis=1, result_type='expand')

# Extracting individual sentiment scores
for df in [groberta_new, broberta_new]:
    df['roberta_neg'] = df['SentimentScores'].apply(lambda scores: scores[0])
    df['roberta_neu'] = df['SentimentScores'].apply(lambda scores: scores[1])
    df['roberta_pos'] = df['SentimentScores'].apply(lambda scores: scores[2])
    df['roberta_composite'] = df['roberta_pos'] - df['roberta_neg']
    df.rename(columns={'Link': 'roberta_link'}, inplace=True)

# Merging the new sentiment data with the original data
bing_data_roberta_new = pd.merge(bing_data, broberta_new, left_on='Link', right_on='roberta_link', how='left')
google_data_roberta_new = pd.merge(google_data, groberta_new, left_on='Link', right_on='roberta_link', how='left')

model_save_path = '/content/drive/My Drive/MTD/Colab saves/roberta_new'
model.save_pretrained(model_save_path)

groberta_new_save_path = '/content/drive/My Drive/MTD/Colab saves/groberta_new.csv'
broberta_new_save_path = '/content/drive/My Drive/MTD/Colab saves/broberta_new.csv'

groberta_new.to_csv(groberta_new_save_path, index=False)
broberta_new.to_csv(broberta_new_save_path, index=False)

from transformers import RobertaForSequenceClassification

model_load_path = '/content/drive/My Drive/MTD/Colab saves/roberta_new'
model = RobertaForSequenceClassification.from_pretrained(model_load_path)

import pandas as pd
```

```
groberta_new_load_path = '/content/drive/My Drive/MTD/Colab saves/groberta_new.csv'
broberta_new_load_path = '/content/drive/My Drive/MTD/Colab saves/broberta_new.csv'

groberta_new = pd.read_csv(groberta_new_load_path)
broberta_new = pd.read_csv(broberta_new_load_path)

import pandas as pd

# Merge groberta_new with google_data via 'Link'
groberta_merged = pd.merge(groberta_new, google_data[['Link', 'legal_name']],
                           left_on='roberta_link', right_on='Link', how='left')
groberta_merged.drop(columns=['Link'], inplace=True)  # Drop redundant Link column

# Merge broberta_new with bing_data via 'Link'
broberta_merged = pd.merge(broberta_new, bing_data[['Link', 'legal_name']],
                           left_on='roberta_link', right_on='Link', how='left')
broberta_merged.drop(columns=['Link'], inplace=True)  # Drop redundant Link column

# Verify the merges
print("Preview of groberta_merged:")
print(groberta_merged.head())

print("\nPreview of broberta_merged:")
print(broberta_merged.head())

# Check for missing legal_name values
missing_google = groberta_merged['legal_name'].isna().sum()
missing_bing = broberta_merged['legal_name'].isna().sum()

print(f"\nMissing 'legal_name' in groberta_merged: {missing_google}")
print(f"Missing 'legal_name' in broberta_merged: {missing_bing}")

import matplotlib.pyplot as plt
import seaborn as sns
```

```
import pandas as pd

# Step 1: Add Source column for merged Roberta data
broberta_merged['Source'] = 'Bing'
groberta_merged['Source'] = 'Google'

# Step 2: Combine Bing and Google Roberta data
combined_roberta_data = pd.concat([
    broberta_merged[['roberta_composite', 'Source']],
    groberta_merged[['roberta_composite', 'Source']]
])

# Step 3: Drop NaN values and ensure numeric type
combined_roberta_data = combined_roberta_data.dropna(subset=['roberta_composite'])
combined_roberta_data['roberta_composite'] = pd.to_numeric(
    combined_roberta_data['roberta_composite'], errors='coerce'
)

# Step 4: Plot boxplot comparison
plt.figure(figsize=(10, 6))
sns.boxplot(x='Source', y='roberta_composite', data=combined_roberta_data, palette='coolwarm')

# Customize the plot
plt.title('Distribution of Roberta Composite Scores: Bing vs. Google')
plt.xlabel('Source')
plt.ylabel('Roberta Composite Score')
plt.axhline(0, color='black', linestyle='--', linewidth=1)

# Show the plot
plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt
import pandas as pd
```

```
# Step 1: Aggregate sentiment counts for Bing and Google
bing_sentiment_counts = {
    'Negative': (broberta_merged['roberta_neg'] > 0.5).sum(),
    'Neutral': (broberta_merged['roberta_neu'] > 0.5).sum(),
    'Positive': (broberta_merged['roberta_pos'] > 0.5).sum()
}

google_sentiment_counts = {
    'Negative': (groberta_merged['roberta_neg'] > 0.5).sum(),
    'Neutral': (groberta_merged['roberta_neu'] > 0.5).sum(),
    'Positive': (groberta_merged['roberta_pos'] > 0.5).sum()
}

# Step 2: Combine sentiment data into a DataFrame
sentiment_comparison = pd.DataFrame({
    'Bing': bing_sentiment_counts,
    'Google': google_sentiment_counts
})

# Step 3: Plot bar chart
ax = sentiment_comparison.plot(kind='bar', figsize=(10, 6), color=['blue', 'green'])

# Customize the plot
plt.title('Sentiment Category Comparison: Bing vs. Google (Roberta)')
plt.xlabel('Sentiment Category')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.legend(title='Source')

# Annotate the exact values on the bars
for p in ax.patches:
    value = int(p.get_height())
    if value > 0:
        ax.text(p.get_x() + p.get_width() / 2, p.get_height() + 5,
                f'{value}', ha='center', va='bottom', fontsize=10, fontweight='bold')
```

```
# Step 4: Display the plot
plt.tight_layout()
plt.show()

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Step 1: Aggregate 'roberta_composite' scores by 'legal_name'
bing_grouped = broberta_merged.groupby('legal_name')['roberta_composite'].mean().reset_index()
google_grouped = groberta_merged.groupby('legal_name')['roberta_composite'].mean().reset_index()

# Step 2: Merge Bing and Google data
merged_data = pd.merge(bing_grouped, google_grouped, on='legal_name', how='outer', suffixes=('_bing', '_google'))
merged_data.fillna(0, inplace=True)

# Step 3: Calculate discrepancies
merged_data['discrepancy'] = abs(merged_data['roberta_composite_bing'] - merged_data['roberta_composite_google'])

# Step 4: Identify top 20 discrepancies
top_discrepancies = merged_data.sort_values(by='discrepancy', ascending=False).head(20)

# Step 5: Analyze matched links for companies in top discrepancies
discrepancy_analysis = []

for company in top_discrepancies['legal_name']:
    bing_links = broberta_merged[broberta_merged['legal_name'] == company]['roberta_link'].unique()
    google_links = groberta_merged[groberta_merged['legal_name'] == company]['roberta_link'].unique()
    matched_links = set(bing_links).intersection(set(google_links))
```

```
    discrepancy_analysis.append({
        'Company': company,
        'Bing_Links': len(bing_links),
        'Google_Links': len(google_links),
        'Matched_Links': len(matched_links)
    })

# Step 6: Convert results to DataFrame
discrepancy_df = pd.DataFrame(discrepancy_analysis)

# Step 7: Plot the results
plt.figure(figsize=(12, 6))
bar_width = 0.25
index = np.arange(len(discrepancy_df))

plt.bar(index, discrepancy_df['Bing_Links'], bar_width, label='Bing Links', color='blue')
plt.bar(index + bar_width, discrepancy_df['Google_Links'], bar_width, label='Google Links', color='green')
plt.bar(index + 2 * bar_width, discrepancy_df['Matched_Links'], bar_width, label='Matched Links', color='orange')

plt.xticks(index + bar_width, discrepancy_df['Company'], rotation=90)
plt.title('Link Comparison for Top Discrepancies')
plt.ylabel('Number of Links')
plt.legend()

plt.tight_layout()
plt.show()

"""### **BERT re-training**"""

import pandas as pd
from transformers import BertTokenizer, BertForSequenceClassification
from torch.nn.functional import softmax
```

```
import torch

# Assume your dataframes are loaded into the variables:
# un_sanctions, eu_sanctions, ofac, easa_df, faa_df, icao_df

# Merge and process the text columns as described
un_sanctions['text'] = un_sanctions[['FIRST_NAME', 'SECOND_NAME', 'THIRD_NAME']].agg('
'.join, axis=1)
eu_sanctions['text'] = eu_sanctions['nameAliases']
ofac['text'] = ofac['lastName'] + ' ' + ofac['firstName']

# Function to perform sentiment analysis using BERT
def bert_sentiment_analysis(texts, model, tokenizer):
    inputs = tokenizer(texts, padding=True, truncation=True, max_length=512, return_tensors="pt")
    outputs = model(**inputs)
    probs = softmax(outputs.logits, dim=1)
    return ['positive' if prob[1] > prob[0] else 'negative' for prob in probs]

# Load a pre-trained BERT model for sentiment analysis
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert-base-uncased')

# Apply BERT analysis to easa_df, faa_df, and icao_df
easa_df['label'] = bert_sentiment_analysis(easa_df['text'].tolist(), model, tokenizer)
faa_df['label'] = bert_sentiment_analysis(faa_df['text'].tolist(), model, tokenizer)
icao_df['label'] = bert_sentiment_analysis(icao_df['text'].tolist(), model, tokenizer)

# Assign negative labels to sanctions lists
un_sanctions['label'] = 'negative'
eu_sanctions['label'] = 'negative'
ofac['label'] = 'negative'

# Combine all dataframes
combined_df = pd.concat([un_sanctions, eu_sanctions, ofac, easa_df, faa_df, icao_df])
```

```
# Select only the relevant columns
combined_df = combined_df[['text', 'label']]

distinct_label_values = combined_df['label'].value_counts()
print(distinct_label_values)

from sklearn.model_selection import train_test_split

# Shuffle the DataFrame
combined_df_shuffled = combined_df.sample(frac=1).reset_index(drop=True)

# Split into training and validation sets
train_df, validation_df = train_test_split(combined_df_shuffled, test_size=0.2)  # 80% training, 20% validation

# Now train_df and validation_df are your training and validation sets

import torch

# Check if CUDA is available
if torch.cuda.is_available():
    print("CUDA is available. GPU: ", torch.cuda.get_device_name(0))
else:
    print("CUDA is not available.")

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

import os
os.environ['CUDA_LAUNCH_BLOCKING'] = "1"

x = torch.rand(5, 3).to('cuda')
print(x)

from transformers import BertTokenizer
```

```
# Load the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

def encode_data(tokenizer, texts, labels, max_length):
    input_ids = []
    attention_masks = []

    for text in texts:
        encoded_dict = tokenizer.encode_plus(
                        text,
                        add_special_tokens = True,
                        max_length = max_length,
                        pad_to_max_length = True,
                        return_attention_mask = True,
                        return_tensors = 'pt',
                    )

        input_ids.append(encoded_dict['input_ids'])
        attention_masks.append(encoded_dict['attention_mask'])

    input_ids = torch.cat(input_ids, dim=0)
    attention_masks = torch.cat(attention_masks, dim=0)
    labels = torch.tensor(labels)

    return input_ids, attention_masks, labels

# Prepare the data
train_texts = train_df['text'].tolist()
train_labels = train_df['label'].apply(lambda x: 1 if x == 'positive' else 0).tolist()

validation_texts = validation_df['text'].tolist()
validation_labels = validation_df['label'].apply(lambda x: 1 if x == 'positive' else 0).tolist()

# Encode the data
```

```
train_inputs, train_masks, train_labels = encode_data(tokenizer, train_texts, train_labels, max_length=256)
validation_inputs, validation_masks, validation_labels = encode_data(tokenizer, validation_texts, validation_labels, max_length=256)

from torch.utils.data import DataLoader, RandomSampler, SequentialSampler, TensorDataset

# Create the DataLoader for our training set
train_data = TensorDataset(train_inputs, train_masks, train_labels)
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler, batch_size=32)

# Create the DataLoader for our validation set
validation_data = TensorDataset(validation_inputs, validation_masks, validation_labels)
validation_sampler = SequentialSampler(validation_data)
validation_dataloader = DataLoader(validation_data, sampler=validation_sampler, batch_size=32)

from transformers import BertForSequenceClassification, AdamW

# Load BertForSequenceClassification, the pretrained BERT model with a single linear classification layer on top
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels = 2, # Binary classification (positive/negative)
    output_attentions = False,
    output_hidden_states = False,
)

# Move the model to the GPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

optimizer = AdamW(model.parameters(), lr=2e-5)

## TEST CPU
```

```
import torch
import time  # Import the time module for tracking

# Define the training loop with time estimation
def train_with_time_estimation(model, train_dataloader, validation_dataloader, optimizer, epochs=4):
    total_training_time = 0  # Initialize total training time

    for epoch_i in range(epochs):
        print('======== Epoch {:} / {:} ========'.format(epoch_i + 1, epochs))

        # Record the start time of the epoch
        epoch_start_time = time.time()

        # Training phase
        model.train()
        total_train_loss = 0

        for step, batch in enumerate(train_dataloader):
            b_input_ids = batch[0].to(device)
            b_input_mask = batch[1].to(device)
            b_labels = batch[2].to(device)

            model.zero_grad()

            outputs = model(b_input_ids, token_type_ids=None, attention_mask=b_input_mask, labels=b_labels)
            loss = outputs.loss
            total_train_loss += loss.item()
            loss.backward()
            optimizer.step()

        avg_train_loss = total_train_loss / len(train_dataloader)
        print(f"Average training loss: {avg_train_loss:.2f}")
```

```
    # Validation phase
    model.eval()
    total_eval_loss = 0

    for batch in validation_dataloader:
        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].to(device)

        with torch.no_grad():
            outputs = model(b_input_ids, token_type_ids=None, attention_mask=b_input_mask, labels=b_labels)

        loss = outputs.loss
        total_eval_loss += loss.item()

    avg_val_loss = total_eval_loss / len(validation_dataloader)
    print(f"Validation loss: {avg_val_loss:.2f}")

    # Calculate time taken for the epoch
    epoch_end_time = time.time()
    epoch_time = epoch_end_time - epoch_start_time
    total_training_time += epoch_time

    # Print epoch timing
    print(f"Time taken for epoch {epoch_i + 1}: {epoch_time:.2f} seconds")

    # Estimate remaining time
    remaining_epochs = epochs - (epoch_i + 1)
    estimated_total_time = total_training_time / (epoch_i + 1) * epochs
    estimated_remaining_time = estimated_total_time - total_training_time

    print(f"Estimated total time: {estimated_total_time:.2f} seconds")
    print(f"Estimated remaining time: {estimated_remaining_time:.2f} seconds")
```

```
    # Print total training time
    print(f"Total training time: {total_training_time:.2f} seconds")

# Run the training with time estimation
train_with_time_estimation(model, train_dataloader, validation_dataloader, optimizer, epochs=4)

import torch

# Define the training loop
def train(model, train_dataloader, validation_dataloader, optimizer, epochs=4):
    for epoch_i in range(epochs):
        print('======== Epoch {:} / {:} ========'.format(epoch_i + 1, epochs))

        # Training phase
        model.train()
        total_train_loss = 0

        for step, batch in enumerate(train_dataloader):
            b_input_ids = batch[0].to(device)
            b_input_mask = batch[1].to(device)
            b_labels = batch[2].to(device)

            model.zero_grad()

            outputs = model(b_input_ids, token_type_ids=None, attention_mask=b_input_mask, labels=b_labels)
            loss = outputs.loss
            total_train_loss += loss.item()
            loss.backward()
            optimizer.step()

        avg_train_loss = total_train_loss / len(train_dataloader)
        print(f"Average training loss: {avg_train_loss:.2f}")

        # Validation phase
```

```
        model.eval()
        total_eval_loss = 0

        for batch in validation_dataloader:
            b_input_ids = batch[0].to(device)
            b_input_mask = batch[1].to(device)
            b_labels = batch[2].to(device)

            with torch.no_grad():
                outputs = model(b_input_ids, token_type_ids=None, attention_mask=b_input_mask, labels=b_labels)

            loss = outputs.loss
            total_eval_loss += loss.item()

        avg_val_loss = total_eval_loss / len(validation_dataloader)
        print(f"Validation loss: {avg_val_loss:.2f}")

# Run the training
train(model, train_dataloader, validation_dataloader, optimizer, epochs=4)

# Define the save path
save_directory = '/content/drive/My Drive/MTD/Colab saves/bert_model_new'

# Create the directory if it doesn't exist
import os
if not os.path.exists(save_directory):
    os.makedirs(save_directory)

# Save the model
model.save_pretrained(save_directory)

import pandas as pd
from transformers import BertTokenizer, BertForSequenceClassification
import torch
```

```
from tqdm import tqdm

# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Load the standard pre-trained tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Load your fine-tuned model from the drive
model_path = '/content/drive/My Drive/MTD/Colab saves/bert_model_new'
model = BertForSequenceClassification.from_pretrained(model_path)

# Define the sentiment analysis function using your model
def bert_sentiment_analysis_new(text):
    inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True, max_length=512)
    with torch.no_grad():
        outputs = model(**inputs)
        predictions = torch.nn.functional.softmax(outputs.logits, dim=-1)
    return predictions.numpy()[0].tolist()

# Analyze sentiment on google_data
bert_results_new = []
source_indicator_new = []
links_new = []

for _, row in tqdm(google_data.iterrows(), total=google_data.shape[0]):
    links_new.append(row['Link'])
    text = row['FullText'] if pd.notnull(row['FullText']) and row['FullText'] != "None" else row['Snippet']
    text = text if pd.notnull(text) and text != "None" else ""
    bert_result_new = bert_sentiment_analysis_new(text)
    bert_results_new.append(bert_result_new)
    source_indicator_new.append('FullText' if pd.notnull(row['FullText']) and row['FullText'] != "None" else 'Snippet')
```

```
gbert_new = pd.DataFrame(bert_results_new, columns=['BERT_Neg', 'BERT_Pos'])
gbert_new['SourceUsed'] = source_indicator_new
gbert_new['Link'] = links_new

# Analyze sentiment on bing_data
bert_results_bing_new = []
source_indicator_bing_new = []
links_bing_new = []

for _, row in tqdm(bing_data.iterrows(), total=bing_data.shape[0]):
    links_bing_new.append(row['Link'])
    text = row['FullText'] if pd.notnull(row['FullText']) and row['FullText'] != "None" else row['Snippet']
    text = text if pd.notnull(text) and text != "None" else ""
    bert_result_bing_new = bert_sentiment_analysis_new(text)
    bert_results_bing_new.append(bert_result_bing_new)
    source_indicator_bing_new.append('FullText' if pd.notnull(row['FullText']) and row['FullText'] != "None" else 'Snippet')

bbert_new = pd.DataFrame(bert_results_bing_new, columns=['BERT_Neg', 'BERT_Pos'])
bbert_new['SourceUsed'] = source_indicator_bing_new
bbert_new['Link'] = links_bing_new

# Step 1: Aggregate BERT results to ensure uniqueness by 'Link'
gbert_new_agg = gbert_new.groupby('Link', as_index=False).agg({
    'BERT_Neg': 'mean',
    'BERT_Pos': 'mean',
    'SourceUsed': 'first'  # Retain the first source indicator
})

bbert_new_agg = bbert_new.groupby('Link', as_index=False).agg({
    'BERT_Neg': 'mean',
    'BERT_Pos': 'mean',
    'SourceUsed': 'first'
```

```
})

# Step 2: Merge gbert_new with google_data
google_data_bert_new = pd.merge(google_data, gbert_new_agg, on='Link', how='left')

# Step 3: Calculate the composite score for google_data
google_data_bert_new['BERT_Composite'] = google_data_bert_new['BERT_Pos'] -
google_data_bert_new['BERT_Neg']

# Step 4: Merge bbert_new with bing_data
bing_data_bert_new = pd.merge(bing_data, bbert_new_agg, on='Link', how='left')

# Step 5: Calculate the composite score for bing_data
bing_data_bert_new['BERT_Composite'] = bing_data_bert_new['BERT_Pos'] -
bing_data_bert_new['BERT_Neg']

# Step 6: Verify the row counts
print("Row count in google_data:", len(google_data))
print("Row count in google_data_bert_new:", len(google_data_bert_new))
print("Row count in bing_data:", len(bing_data))
print("Row count in bing_data_bert_new:", len(bing_data_bert_new))

# Display a sample of the merged results
print("\nSample of Google Data with BERT Sentiment:")
print(google_data_bert_new.head())

print("\nSample of Bing Data with BERT Sentiment:")
print(bing_data_bert_new.head())

# Optional: Save the results to CSV
output_path_google = "/content/drive/My Drive/MTD/KYC
data/Output/google_data_bert_new.csv"
output_path_bing = "/content/drive/My Drive/MTD/KYC data/Output/bing_data_bert_new.csv"

google_data_bert_new.to_csv(output_path_google, index=False)
```

```
bing_data_bert_new.to_csv(output_path_bing, index=False)

print(f"\nGoogle data with BERT sentiment saved to: {output_path_google}")
print(f"Bing data with BERT sentiment saved to: {output_path_bing}")

import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Add Source column to identify Bing and Google data
bbert_new['Source'] = 'Bing'
gbert_new['Source'] = 'Google'

# Calculate Composite Score
bbert_new['BERT_Composite'] = bbert_new['BERT_Pos'] - bbert_new['BERT_Neg']
gbert_new['BERT_Composite'] = gbert_new['BERT_Pos'] - gbert_new['BERT_Neg']

# Combine Bing and Google data
combined_bert_data_new = pd.concat([
    bbert_new[['BERT_Composite', 'Source']],
    gbert_new[['BERT_Composite', 'Source']]
])

# Plot Boxplot
plt.figure(figsize=(10, 6))
sns.boxplot(x='Source', y='BERT_Composite', data=combined_bert_data_new, palette='coolwarm')

# Customize Plot
plt.title('Distribution of BERT Composite Scores: Bing vs. Google')
plt.xlabel('Source')
plt.ylabel('BERT Composite Score')
plt.axhline(0, color='black', linestyle='--', linewidth=1)
plt.show()

import matplotlib.pyplot as plt
```

```
# Step 1: Aggregate sentiment counts
bing_sentiment_counts_new = {
    'Negative': (bbert_new['BERT_Neg'] > 0.5).sum(),
    'Positive': (bbert_new['BERT_Pos'] > 0.5).sum()
}

google_sentiment_counts_new = {
    'Negative': (gbert_new['BERT_Neg'] > 0.5).sum(),
    'Positive': (gbert_new['BERT_Pos'] > 0.5).sum()
}

# Step 2: Combine data into a DataFrame
sentiment_comparison_new = pd.DataFrame({
    'Bing': bing_sentiment_counts_new,
    'Google': google_sentiment_counts_new
})

# Step 3: Plot Bar Chart
ax = sentiment_comparison_new.plot(kind='bar', figsize=(10, 6), color=['blue', 'green'])

# Customize Plot
plt.title('Sentiment Category Comparison: Bing vs. Google (BERT)')
plt.xlabel('Sentiment Category')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.legend(title='Source')

# Annotate Values
for p in ax.patches:
    value = int(p.get_height())
    if value > 0:
        ax.text(p.get_x() + p.get_width() / 2, p.get_height() + 5,
                f'{value}', ha='center', va='bottom', fontsize=10, fontweight='bold')
```

```
plt.tight_layout()
plt.show()

"""# Chat GPT Open AI new prompt

"""

pip install --upgrade openai

import openai
import pandas as pd
from tqdm import tqdm
import re

# Set your OpenAI API Key
openai.api_key = "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"

# Modified function for sentiment analysis in aviation context
def gpt_aviation_sentiment_analysis(text):
    try:
        response = openai.ChatCompletion.create(
            model="gpt-3.5-turbo",
            messages=[
                {"role": "system", "content": "You are an AI that performs sentiment analysis specifically in the aviation context."},
                {"role": "user", "content": f"Please analyze the sentiment of this text in the aviation context. "
                                             f"Provide scores strictly in this format: [neg:<negative_score>, neu:<neutral_score>, pos:<positive_score>] "
                                             f"along with a brief judgment (e.g., 'Positive sentiment related to safety').\nText: {text}"}
            ],
            temperature=0.2,
            max_tokens=150
        )
```

```
        # Extract and return the content of the response
        result = response['choices'][0]['message']['content'].strip()
        return result
    except Exception as e:
        print(f"Error processing text: {e}")
        return None

# Function to process data for sentiment analysis with aviation context
def process_aviation_sentiment(data, data_name):
    gpt_results = []
    for _, row in tqdm(data.iterrows(), total=data.shape[0], desc=f"Processing {data_name}"):
        text = row['FullText'] if pd.notnull(row['FullText']) and row['FullText'] != "None" else row['Snippet']
        if pd.notnull(text) and text != "None":  # Ensure valid text
            sentiment_result = gpt_aviation_sentiment_analysis(text)
            gpt_results.append((row['Link'], sentiment_result))
        else:
            gpt_results.append((row['Link'], None))

    # Convert results to DataFrame
    gpt_sentiment_df = pd.DataFrame(gpt_results, columns=['Link', 'GPT_Sentiment'])
    return gpt_sentiment_df

# Split the GPT sentiment into components
def split_gpt_aviation_sentiment(sentiment):
    if isinstance(sentiment, str):
        matches = re.findall(r'neg:(\d*\.?\d*)|neu:(\d*\.?\d*)|pos:(\d*\.?\d*)', sentiment)
        neg = float(matches[0][0] or 0) if matches and len(matches) > 0 else 0.0
        neu = float(matches[1][1] or 0) if matches and len(matches) > 1 else 0.0
        pos = float(matches[2][2] or 0) if matches and len(matches) > 2 else 0.0
        return neg, neu, pos
    return 0.0, 0.0, 0.0

# Process the new GPT sentiment data
def process_gpt_aviation_sentiment(df, source_name):
```

```
    df_copy = df.copy()
    df_copy[['gpt_neg', 'gpt_neu', 'gpt_pos']] = pd.DataFrame(
        df_copy['GPT_Sentiment'].apply(split_gpt_aviation_sentiment).tolist(), index=df_copy.index
    )
    df_copy['gpt_composite'] = df_copy['gpt_pos'] - df_copy['gpt_neg']
    df_copy = df_copy.drop_duplicates(subset=['Link'], keep='first')
    df_copy['Source'] = source_name
    return df_copy

# Assuming google_data and bing_data are loaded DataFrames
google_data = pd.read_csv('/content/drive/My Drive/MTD/KYC data/Output/google_data.csv')
bing_data = pd.read_csv('/content/drive/My Drive/MTD/KYC data/Output/bing_data.csv')

# Process Bing and Google data
gpt_google_new = process_aviation_sentiment(google_data, "Google Data (Aviation Context)")
gpt_bing_new = process_aviation_sentiment(bing_data, "Bing Data (Aviation Context)")

# Save results
gpt_google_new.to_csv("/content/drive/My Drive/MTD/KYC data/Output/gpt_google_new.csv",
index=False)
gpt_bing_new.to_csv("/content/drive/My Drive/MTD/KYC data/Output/gpt_bing_new.csv",
index=False)

# Process the sentiment data
gpt_google_new_processed = process_gpt_aviation_sentiment(gpt_google_new, 'Google')
gpt_bing_new_processed = process_gpt_aviation_sentiment(gpt_bing_new, 'Bing')

# Merge new GPT sentiment back to Google and Bing data
columns_to_merge = ['Link', 'gpt_neg', 'gpt_neu', 'gpt_pos', 'gpt_composite', 'Source']

google_data_gpt_new = pd.merge(
    google_data,
    gpt_google_new_processed[columns_to_merge],
    on='Link',
    how='left'
```

```
)

bing_data_gpt_new = pd.merge(
    bing_data,
    gpt_bing_new_processed[columns_to_merge],
    on='Link',
    how='left'
)

# Fill missing values for rows without GPT sentiment
for col in ['gpt_neg', 'gpt_neu', 'gpt_pos', 'gpt_composite']:
    google_data_gpt_new[col] = google_data_gpt_new[col].fillna(0.0)
    bing_data_gpt_new[col] = bing_data_gpt_new[col].fillna(0.0)

# Save the final processed data
google_data_gpt_new.to_csv("/content/drive/My                    Drive/MTD/KYC
data/Output/google_data_gpt_new.csv", index=False)
bing_data_gpt_new.to_csv("/content/drive/My                    Drive/MTD/KYC
data/Output/bing_data_gpt_new.csv", index=False)

print("Saved processed GPT aviation sentiment analysis for Google and Bing Data.")

"""# Data load

"""

from google.colab import drive
drive.mount('/content/drive')

import os

# Check contents of the directories
new_dir = '/content/drive/My Drive/MTD/Final/New/'
primary_dir = '/content/drive/My Drive/MTD/Final/Primary/'
```

```
print("Files in New Directory:")
print(os.listdir(new_dir))

print("\nFiles in Primary Directory:")
print(os.listdir(primary_dir))

import pandas as pd
import os

# Define directories
new_dir = '/content/drive/My Drive/MTD/Final/New/'
primary_dir = '/content/drive/My Drive/MTD/Final/Primary/'

# Function to clean filenames and load files into individual DataFrames
def load_and_create_dataframes(directory, file_mapping):
    globals_dict = globals()  # Reference the global namespace for dynamic variable creation
    print(f"\nChecking directory: {directory}\n")
    for file in os.listdir(directory):
        if file.endswith('.csv') or 'kopija' in file:  # Check for CSV or 'kopija'
            file_path = os.path.join(directory, file)

            # Clean up the filename: remove special characters and spaces
            cleaned_name = (file.replace(',,', '')
                            .replace('“', '')
                            .replace(' kopija', '')
                            .replace('.csv', '')
                            .replace(' ', '_')
                            .lower())

            print(f"Original file: '{file}' -> Cleaned name: '{cleaned_name}'")  # Debugging

            if cleaned_name in file_mapping:
                dataframe_name = file_mapping[cleaned_name]
                globals_dict[dataframe_name] = pd.read_csv(file_path)  # Create a variable dynamically
                print(f"Loaded '{file}' into DataFrame '{dataframe_name}'")
```

```
            else:
                print(f"Warning: '{file}' does not match any mapping and will be skipped.")

# Map filenames to desired DataFrame names
new_file_mapping = {
    "google_data_vaders_new": "gvaders_new",
    "bing_data_vaders_new": "bvaders_new",
    "groberta_new": "groberta_new",
    "broberta_new": "broberta_new",
    "google_data_gpt_new": "ggpt_new",
    "bing_data_gpt_new": "bgpt_new",
    "google_data_bert_new": "gbert_new",
    "bing_data_bert_new": "bbert_new"
}

primary_file_mapping = {
    "google_data_vaders": "gvaders",
    "bing_data_vaders": "bvaders",
    "google_data_roberta": "groberta",
    "bing_data_roberta": "broberta",
    "google_data_bert": "gbert",
    "bing_data_bert": "bbert",
    "google_data_gpt": "ggpt",
    "bing_data_gpt": "bgpt"
}

# Load data from both directories and create individual DataFrames
load_and_create_dataframes(new_dir, new_file_mapping)
load_and_create_dataframes(primary_dir, primary_file_mapping)

# Display summary of all loaded DataFrames
print("\nSummary of Loaded DataFrames:")
for df_name in new_file_mapping.values():
    if df_name in globals():
        print(f"{df_name}: {eval(df_name).shape[0]} rows, {eval(df_name).shape[1]} columns")
```

```
for df_name in primary_file_mapping.values():
    if df_name in globals():
        print(f"{df_name}: {eval(df_name).shape[0]} rows, {eval(df_name).shape[1]} columns")

import pandas as pd
import os

# Define the directory where the files are located
primary_dir = '/content/drive/My Drive/MTD/Final/'

# Define the mapping of filenames to DataFrame names
file_mapping = {
    "MTD_Companies.csv": "mtd_companies",
    "google_data.csv": "google_data",
    "bing_data.csv": "bing_data"
}

# Load the CSV files into DataFrames
for file, dataframe_name in file_mapping.items():
    file_path = os.path.join(primary_dir, file)
    if os.path.exists(file_path):
        globals()[dataframe_name] = pd.read_csv(file_path)
        print(f"Loaded '{file}' into DataFrame '{dataframe_name}'")
    else:
        print(f"Warning: '{file}' not found in '{primary_dir}'")

# Display summary of loaded DataFrames
print("\nSummary of Loaded DataFrames:")
for dataframe_name in file_mapping.values():
    df = globals().get(dataframe_name)
    if df is not None:
        print(f"{dataframe_name}: {df.shape[0]} rows, {df.shape[1]} columns")

"""# Clustering risk levels
```

```
Vaders
"""

import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Function to group sentiment scores by 'legal_name' and apply K-Means clustering
def kmeans_clustering_risk_per_legal_name(data, name):
    # Step 1: Aggregate sentiment scores by 'legal_name'
    grouped_data = data.groupby('legal_name', as_index=False).agg({
        'vaders_neg': 'mean',
        'vaders_pos': 'mean',
        'vaders_neu': 'mean',
        'vaders_composite': 'mean'
    })

    # Step 2: Normalize the data
    scaler = StandardScaler()
    sentiment_data_scaled = scaler.fit_transform(grouped_data[['vaders_neg', 'vaders_pos', 'vaders_composite']])

    # Step 3: Apply K-Means Clustering with K=3
    kmeans = KMeans(n_clusters=3, random_state=42)
    clusters = kmeans.fit_predict(sentiment_data_scaled)

    # Step 4: Analyze cluster centers and assign risk levels
    cluster_centers = pd.DataFrame(kmeans.cluster_centers_, columns=['vaders_neg', 'vaders_pos', 'vaders_composite'])
    cluster_centers['Risk_Level'] = ['Low', 'Medium', 'High']  # Default assignment

    # Sort clusters based on vaders_composite to ensure proper order of risk levels
```

```
    cluster_centers = cluster_centers.sort_values(by='vaders_composite', ascending=True).reset_index(drop=True)
    risk_mapping = {i: risk for i, risk in enumerate(['Low', 'Medium', 'High'])}

    # Step 5: Add Risk Levels back to grouped data
    grouped_data['Cluster'] = clusters
    grouped_data['Risk_Level'] = grouped_data['Cluster'].map(risk_mapping)

    # Step 6: Display Results
    print(f"\n{name} Risk Levels:")
    print(grouped_data[['legal_name', 'vaders_neg', 'vaders_pos', 'vaders_composite', 'Risk_Level']].head())

    # Step 7: Visualize the Clusters
    plt.figure(figsize=(8, 6))
    plt.scatter(grouped_data['vaders_composite'], grouped_data['vaders_neg'], c=grouped_data['Cluster'], cmap='viridis', s=50)
    plt.title(f"{name} - Sentiment Clustering (Per Legal Name)")
    plt.xlabel('Vaders Composite Score')
    plt.ylabel('Vaders Negative Score')
    plt.colorbar(label='Cluster')
    plt.show()

    return grouped_data

# Apply the function to gvaders and bvaders using 'legal_name'
gvaders_risk = kmeans_clustering_risk_per_legal_name(gvaders, "Google Vaders")
bvaders_risk = kmeans_clustering_risk_per_legal_name(bvaders, "Bing Vaders")

"""Vaders new"""

import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

```
# Function to group sentiment scores by 'legal_name' and apply K-Means clustering
def kmeans_clustering_risk_per_legal_name(data, name):
    # Step 1: Detect and rename columns dynamically
    column_mapping = {
        'vaders_new_neg': 'vaders_neg',
        'vaders_new_pos': 'vaders_pos',
        'vaders_new_neu': 'vaders_neu',
        'vaders_new_composite': 'vaders_composite',
        'vaders_neg': 'vaders_neg',
        'vaders_pos': 'vaders_pos',
        'vaders_neu': 'vaders_neu',
        'vaders_composite': 'vaders_composite'
    }

    # Rename columns if necessary
    data = data.rename(columns={col: column_mapping[col] for col in data.columns if col in column_mapping})

    # Step 2: Aggregate sentiment scores by 'legal_name'
    grouped_data = data.groupby('legal_name', as_index=False).agg({
        'vaders_neg': 'mean',
        'vaders_pos': 'mean',
        'vaders_neu': 'mean',
        'vaders_composite': 'mean'
    })

    # Step 3: Normalize the data
    scaler = StandardScaler()
    sentiment_data_scaled = scaler.fit_transform(grouped_data[['vaders_neg', 'vaders_pos', 'vaders_composite']])

    # Step 4: Apply K-Means Clustering with K=3
    kmeans = KMeans(n_clusters=3, random_state=42)
    clusters = kmeans.fit_predict(sentiment_data_scaled)
```

```
    # Step 5: Analyze cluster centers and assign risk levels
    cluster_centers = pd.DataFrame(kmeans.cluster_centers_, columns=['vaders_neg', 'vaders_pos', 'vaders_composite'])
    cluster_centers['Risk_Level'] = ['Low', 'Medium', 'High']  # Default assignment

    # Sort clusters based on vaders_composite to ensure proper order of risk levels
    cluster_centers = cluster_centers.sort_values(by='vaders_composite', ascending=True).reset_index(drop=True)
    risk_mapping = {i: risk for i, risk in enumerate(['Low', 'Medium', 'High'])}

    # Step 6: Add Risk Levels back to grouped data
    grouped_data['Cluster'] = clusters
    grouped_data['Risk_Level'] = grouped_data['Cluster'].map(risk_mapping)

    # Step 7: Display Results
    print(f"\n{name} Risk Levels:")
    print(grouped_data[['legal_name', 'vaders_neg', 'vaders_pos', 'vaders_composite', 'Risk_Level']].head())

    # Step 8: Visualize the Clusters
    plt.figure(figsize=(8, 6))
    plt.scatter(grouped_data['vaders_composite'], grouped_data['vaders_neg'], c=grouped_data['Cluster'], cmap='viridis', s=50)
    plt.title(f"{name} - Sentiment Clustering (Per Legal Name)")
    plt.xlabel('Vaders Composite Score')
    plt.ylabel('Vaders Negative Score')
    plt.colorbar(label='Cluster')
    plt.show()

    return grouped_data

# Apply the function to gvaders_new and bvaders_new using 'legal_name'
gvaders_risk_new = kmeans_clustering_risk_per_legal_name(gvaders_new, "Google Vaders New")
bvaders_risk_new = kmeans_clustering_risk_per_legal_name(bvaders_new, "Bing Vaders New")
```

```
"""RoBERTa"""

import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Function to perform K-means clustering for risk levels
def kmeans_clustering_risk_per_legal_name(data, name):
    # Step 1: Aggregate sentiment scores by 'legal_name'
    grouped_data = data.groupby('legal_name', as_index=False).agg({
        'roberta_neg': 'mean',
        'roberta_pos': 'mean',
        'roberta_composite': 'mean'
    })

    # Step 2: Normalize the data
    scaler = StandardScaler()
    sentiment_data_scaled = scaler.fit_transform(grouped_data[['roberta_neg', 'roberta_pos', 'roberta_composite']])

    # Step 3: Apply K-Means Clustering with K=3
    kmeans = KMeans(n_clusters=3, random_state=42)
    clusters = kmeans.fit_predict(sentiment_data_scaled)

    # Step 4: Analyze cluster centers and assign risk levels
    cluster_centers = pd.DataFrame(kmeans.cluster_centers_, columns=['roberta_neg', 'roberta_pos', 'roberta_composite'])
    cluster_centers['Risk_Level'] = ['Low', 'Medium', 'High']

    # Sort clusters based on roberta_composite
    cluster_centers = cluster_centers.sort_values(by='roberta_composite', ascending=True).reset_index(drop=True)
    risk_mapping = {i: risk for i, risk in enumerate(['Low', 'Medium', 'High'])}
```

```
    # Step 5: Map risk levels back to grouped data
    grouped_data['Cluster'] = clusters
    grouped_data['Risk_Level'] = grouped_data['Cluster'].map(risk_mapping)

    # Step 6: Display Results
    print(f"\n{name} Risk Levels:")
    print(grouped_data[['legal_name', 'roberta_neg', 'roberta_pos', 'roberta_composite', 'Risk_Level']].head())

    # Step 7: Visualize the Clusters
    plt.figure(figsize=(8, 6))
    plt.scatter(grouped_data['roberta_composite'], grouped_data['roberta_neg'], c=grouped_data['Cluster'], cmap='viridis', s=50)
    plt.title(f"{name} - Sentiment Clustering (Per Legal Name)")
    plt.xlabel('Roberta Composite Score')
    plt.ylabel('Roberta Negative Score')
    plt.colorbar(label='Cluster')
    plt.show()

    return grouped_data

# Apply the function to groberta and broberta
groberta_risk = kmeans_clustering_risk_per_legal_name(groberta, "Google Roberta")
broberta_risk = kmeans_clustering_risk_per_legal_name(broberta, "Bing Roberta")

"""RoBERTa new"""

# Step 1: Link legal_name to groberta_new and broberta_new
groberta_new = groberta_new.merge(google_data[['Link', 'legal_name']], left_on='roberta_link', right_on='Link', how='left')
broberta_new = broberta_new.merge(bing_data[['Link', 'legal_name']], left_on='roberta_link', right_on='Link', how='left')

# Drop redundant 'Link' column after merging
```

```
groberta_new.drop(columns=['Link'], inplace=True)
broberta_new.drop(columns=['Link'], inplace=True)

# Step 2: Function to perform K-means clustering (same as above)
def kmeans_clustering_risk_per_legal_name_new(data, name):
    # Filter out rows without legal_name
    data = data.dropna(subset=['legal_name'])

    # Aggregate sentiment scores by 'legal_name'
    grouped_data = data.groupby('legal_name', as_index=False).agg({
        'roberta_neg': 'mean',
        'roberta_pos': 'mean',
        'roberta_composite': 'mean'
    })

    # Normalize the data
    scaler = StandardScaler()
    sentiment_data_scaled = scaler.fit_transform(grouped_data[['roberta_neg', 'roberta_pos', 'roberta_composite']])

    # Apply K-Means Clustering with K=3
    kmeans = KMeans(n_clusters=3, random_state=42)
    clusters = kmeans.fit_predict(sentiment_data_scaled)

    # Analyze cluster centers and assign risk levels
    cluster_centers = pd.DataFrame(kmeans.cluster_centers_, columns=['roberta_neg', 'roberta_pos', 'roberta_composite'])
    cluster_centers['Risk_Level'] = ['Low', 'Medium', 'High']

    # Sort clusters based on roberta_composite
    cluster_centers = cluster_centers.sort_values(by='roberta_composite', ascending=True).reset_index(drop=True)
    risk_mapping = {i: risk for i, risk in enumerate(['Low', 'Medium', 'High'])}

    # Map risk levels back to grouped data
```

```
    grouped_data['Cluster'] = clusters
    grouped_data['Risk_Level'] = grouped_data['Cluster'].map(risk_mapping)

    # Display Results
    print(f"\n{name} Risk Levels:")
    print(grouped_data[['legal_name', 'roberta_neg', 'roberta_pos', 'roberta_composite', 'Risk_Level']].head())

    # Visualize the Clusters
    plt.figure(figsize=(8, 6))
    plt.scatter(grouped_data['roberta_composite'], grouped_data['roberta_neg'], c=grouped_data['Cluster'], cmap='viridis', s=50)
    plt.title(f"{name} - Sentiment Clustering (Per Legal Name)")
    plt.xlabel('Roberta Composite Score')
    plt.ylabel('Roberta Negative Score')
    plt.colorbar(label='Cluster')
    plt.show()

    return grouped_data

# Apply the function to groberta_new and broberta_new
groberta_risk_new = kmeans_clustering_risk_per_legal_name_new(groberta_new, "Google Roberta New")
broberta_risk_new = kmeans_clustering_risk_per_legal_name_new(broberta_new, "Bing Roberta New")

"""BERT"""

import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Function to perform K-means clustering for risk levels
def kmeans_clustering_risk_per_legal_name(data, name):
```

```
    # Step 1: Aggregate sentiment scores by 'legal_name'
    grouped_data = data.groupby('legal_name', as_index=False).agg({
        'BERT_Neg': 'mean',
        'BERT_Pos': 'mean',
        'BERT_Composite': 'mean'
    })

    # Step 2: Normalize the data
    scaler = StandardScaler()
    sentiment_data_scaled = scaler.fit_transform(grouped_data[['BERT_Neg', 'BERT_Pos', 'BERT_Composite']])

    # Step 3: Apply K-Means Clustering with K=3
    kmeans = KMeans(n_clusters=3, random_state=42)
    clusters = kmeans.fit_predict(sentiment_data_scaled)

    # Step 4: Analyze cluster centers and assign risk levels
    cluster_centers = pd.DataFrame(kmeans.cluster_centers_, columns=['BERT_Neg', 'BERT_Pos', 'BERT_Composite'])
    cluster_centers['Risk_Level'] = ['Low', 'Medium', 'High']  # Default risk level

    # Sort clusters based on BERT_Composite score
    cluster_centers = cluster_centers.sort_values(by='BERT_Composite', ascending=True).reset_index(drop=True)
    risk_mapping = {i: risk for i, risk in enumerate(['Low', 'Medium', 'High'])}

    # Step 5: Map risk levels back to grouped data
    grouped_data['Cluster'] = clusters
    grouped_data['Risk_Level'] = grouped_data['Cluster'].map(risk_mapping)

    # Step 6: Display Results
    print(f"\n{name} Risk Levels:")
    print(grouped_data[['legal_name', 'BERT_Neg', 'BERT_Pos', 'BERT_Composite', 'Risk_Level']].head())
```

```
    # Step 7: Visualize the Clusters
    plt.figure(figsize=(8, 6))
    plt.scatter(grouped_data['BERT_Composite'], grouped_data['BERT_Neg'],
c=grouped_data['Cluster'], cmap='viridis', s=50)
    plt.title(f"{name} - Sentiment Clustering (Per Legal Name)")
    plt.xlabel('BERT Composite Score')
    plt.ylabel('BERT Negative Score')
    plt.colorbar(label='Cluster')
    plt.show()

    return grouped_data

# Apply the function to bbert and gbert
bbert_risk = kmeans_clustering_risk_per_legal_name(bbert, "Bing BERT")
gbert_risk = kmeans_clustering_risk_per_legal_name(gbert, "Google BERT")

"""BERT new"""

# Function for gbert_new and bbert_new
def kmeans_clustering_risk_per_legal_name_new(data, name):
    # Step 1: Aggregate sentiment scores by 'legal_name'
    grouped_data = data.groupby('legal_name', as_index=False).agg({
        'BERT_Neg': 'mean',
        'BERT_Pos': 'mean',
        'BERT_Composite': 'mean'
    })

    # Step 2: Normalize the data
    scaler = StandardScaler()
    sentiment_data_scaled = scaler.fit_transform(grouped_data[['BERT_Neg', 'BERT_Pos',
'BERT_Composite']])

    # Step 3: Apply K-Means Clustering with K=3
    kmeans = KMeans(n_clusters=3, random_state=42)
    clusters = kmeans.fit_predict(sentiment_data_scaled)
```

```
    # Step 4: Analyze cluster centers and assign risk levels
    cluster_centers = pd.DataFrame(kmeans.cluster_centers_, columns=['BERT_Neg', 'BERT_Pos', 'BERT_Composite'])
    cluster_centers['Risk_Level'] = ['Low', 'Medium', 'High']

    # Sort clusters based on BERT_Composite
    cluster_centers = cluster_centers.sort_values(by='BERT_Composite', ascending=True).reset_index(drop=True)
    risk_mapping = {i: risk for i, risk in enumerate(['Low', 'Medium', 'High'])}

    # Step 5: Map risk levels back to grouped data
    grouped_data['Cluster'] = clusters
    grouped_data['Risk_Level'] = grouped_data['Cluster'].map(risk_mapping)

    # Step 6: Display Results
    print(f"\n{name} Risk Levels:")
    print(grouped_data[['legal_name', 'BERT_Neg', 'BERT_Pos', 'BERT_Composite', 'Risk_Level']].head())

    # Step 7: Visualize the Clusters
    plt.figure(figsize=(8, 6))
    plt.scatter(grouped_data['BERT_Composite'], grouped_data['BERT_Neg'], c=grouped_data['Cluster'], cmap='viridis', s=50)
    plt.title(f"{name} - Sentiment Clustering (Per Legal Name)")
    plt.xlabel('BERT Composite Score')
    plt.ylabel('BERT Negative Score')
    plt.colorbar(label='Cluster')
    plt.show()

    return grouped_data

# Apply the function to gbert_new and bbert_new
gbert_new_risk = kmeans_clustering_risk_per_legal_name_new(gbert_new, "Google BERT New")
bbert_new_risk = kmeans_clustering_risk_per_legal_name_new(bbert_new, "Bing BERT New")
```

```
"""GPT"""

import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Function to perform K-means clustering for risk levels
def kmeans_clustering_risk_per_legal_name(data, name):
    # Step 1: Aggregate sentiment scores by 'legal_name'
    grouped_data = data.groupby('legal_name', as_index=False).agg({
        'gpt_neg': 'mean',
        'gpt_pos': 'mean',
        'gpt_composite': 'mean'
    })

    # Step 2: Normalize the data
    scaler = StandardScaler()
    sentiment_data_scaled = scaler.fit_transform(grouped_data[['gpt_neg', 'gpt_pos', 'gpt_composite']])

    # Step 3: Apply K-Means Clustering with K=3
    kmeans = KMeans(n_clusters=3, random_state=42)
    clusters = kmeans.fit_predict(sentiment_data_scaled)

    # Step 4: Analyze cluster centers and assign risk levels
    cluster_centers = pd.DataFrame(kmeans.cluster_centers_, columns=['gpt_neg', 'gpt_pos', 'gpt_composite'])
    cluster_centers['Risk_Level'] = ['Low', 'Medium', 'High']  # Default assignment

    # Sort clusters based on gpt_composite
    cluster_centers = cluster_centers.sort_values(by='gpt_composite', ascending=True).reset_index(drop=True)
    risk_mapping = {i: risk for i, risk in enumerate(['Low', 'Medium', 'High'])}
```

```
    # Step 5: Map risk levels back to grouped data
    grouped_data['Cluster'] = clusters
    grouped_data['Risk_Level'] = grouped_data['Cluster'].map(risk_mapping)

    # Step 6: Display Results
    print(f"\n{name} Risk Levels:")
    print(grouped_data[['legal_name', 'gpt_neg', 'gpt_pos', 'gpt_composite', 'Risk_Level']].head())

    # Step 7: Visualize the Clusters
    plt.figure(figsize=(8, 6))
    plt.scatter(grouped_data['gpt_composite'], grouped_data['gpt_neg'], c=grouped_data['Cluster'],
cmap='viridis', s=50)
    plt.title(f"{name} - Sentiment Clustering (Per Legal Name)")
    plt.xlabel('GPT Composite Score')
    plt.ylabel('GPT Negative Score')
    plt.colorbar(label='Cluster')
    plt.show()

    return grouped_data

# Apply the function to ggpt and bgpt
ggpt_risk = kmeans_clustering_risk_per_legal_name(ggpt, "Google GPT")
bgpt_risk = kmeans_clustering_risk_per_legal_name(bgpt, "Bing GPT")

"""GPT new"""

import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Function to perform K-means clustering for risk levels
def kmeans_clustering_risk_per_legal_name(data, name):
    # Step 1: Detect and rename columns dynamically
```

```
    column_mapping = {
        'gpt_neg': 'sent_neg',
        'gpt_pos': 'sent_pos',
        'gpt_neu': 'sent_neu',
        'gpt_composite': 'sent_composite'
    }

    # Rename columns if necessary
    data = data.rename(columns={col: column_mapping[col] for col in data.columns if col in column_mapping})

    # Step 2: Aggregate sentiment scores by 'legal_name'
    grouped_data = data.groupby('legal_name', as_index=False).agg({
        'sent_neg': 'mean',
        'sent_pos': 'mean',
        'sent_neu': 'mean',
        'sent_composite': 'mean'
    })

    # Step 3: Normalize the data
    scaler = StandardScaler()
    sentiment_data_scaled = scaler.fit_transform(grouped_data[['sent_neg', 'sent_pos', 'sent_composite']])

    # Step 4: Apply K-Means Clustering with K=3
    kmeans = KMeans(n_clusters=3, random_state=42)
    clusters = kmeans.fit_predict(sentiment_data_scaled)

    # Step 5: Analyze cluster centers and assign risk levels
    cluster_centers = pd.DataFrame(kmeans.cluster_centers_, columns=['sent_neg', 'sent_pos', 'sent_composite'])
    cluster_centers['Risk_Level'] = ['Low', 'Medium', 'High']

    # Sort clusters based on sent_composite to ensure proper order of risk levels
```

```
    cluster_centers = cluster_centers.sort_values(by='sent_composite', ascending=True).reset_index(drop=True)
    risk_mapping = {i: risk for i, risk in enumerate(['Low', 'Medium', 'High'])}

    # Step 6: Map risk levels back to grouped data
    grouped_data['Cluster'] = clusters
    grouped_data['Risk_Level'] = grouped_data['Cluster'].map(risk_mapping)

    # Step 7: Display Results
    print(f"\n{name} Risk Levels:")
    print(grouped_data[['legal_name', 'sent_neg', 'sent_pos', 'sent_composite', 'Risk_Level']].head())

    # Step 8: Visualize the Clusters
    plt.figure(figsize=(8, 6))
    plt.scatter(grouped_data['sent_composite'], grouped_data['sent_neg'], c=grouped_data['Cluster'], cmap='viridis', s=50)
    plt.title(f"{name} - Sentiment Clustering (Per Legal Name)")
    plt.xlabel('GPT Composite Score')
    plt.ylabel('GPT Negative Score')
    plt.colorbar(label='Cluster')
    plt.show()

    return grouped_data

# Apply the function to ggpt_new and bgpt_new
ggpt_risk_new = kmeans_clustering_risk_per_legal_name(ggpt_new, "Google GPT New")
bgpt_risk_new = kmeans_clustering_risk_per_legal_name(bgpt_new, "Bing GPT New")

"""Grouping model scores (PRIMARY)"""

from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Step 1: Combine all primary datasets
```

```
primary_dataframes = [gvaders, bvaders, groberta, broberta, gbert, bbert, ggpt, bgpt]
for df in primary_dataframes:
    df.rename(columns={'vaders_neg': 'neg', 'vaders_pos': 'pos',
                       'vaders_composite': 'composite', 'roberta_neg': 'neg',
                       'roberta_pos': 'pos', 'roberta_composite': 'composite',
                       'gpt_neg': 'neg', 'gpt_pos': 'pos', 'gpt_composite': 'composite',
                       'BERT_Neg': 'neg', 'BERT_Pos': 'pos', 'BERT_Composite': 'composite'},
              inplace=True)

combined_primary = pd.concat(primary_dataframes, ignore_index=True)

# Step 2: Clean data
combined_primary.drop_duplicates(inplace=True)
combined_primary.dropna(subset=['legal_name'], inplace=True)

# Step 3: Scale numeric features
scaler = StandardScaler()
features = ['neg', 'pos', 'composite']
combined_primary_scaled = scaler.fit_transform(combined_primary[features])

# Step 4: Apply K-Means Clustering
kmeans = KMeans(n_clusters=3, random_state=42)
combined_primary['risk_cluster'] = kmeans.fit_predict(combined_primary_scaled)

# Step 5: Calculate Risk Level per legal_name
final_risk = combined_primary.groupby('legal_name')['risk_cluster'].agg(
    lambda x: x.mode()[0])  # Most frequent cluster

# Step 6: Map risk levels to descriptive labels
risk_labels = {0: 'Low', 1: 'Medium', 2: 'High'}
final_risk = final_risk.map(risk_labels)

# Display the risk levels
print(final_risk.reset_index().rename(columns={'risk_cluster': 'Risk Level'}))
```

```
"""Grouping model scores (NEW)"""

# Prepare google_data and bing_data for clean matching
google_data_clean = google_data[['Link', 'legal_name']].drop_duplicates()
bing_data_clean = bing_data[['Link', 'legal_name']].drop_duplicates()

# Standardize Link columns for matching
google_data_clean['Link'] = google_data_clean['Link'].str.strip().str.rstrip('/')
bing_data_clean['Link'] = bing_data_clean['Link'].str.strip().str.rstrip('/')

groberta_new['roberta_link'] = groberta_new['roberta_link'].str.strip().str.rstrip('/')
broberta_new['roberta_link'] = broberta_new['roberta_link'].str.strip().str.rstrip('/')

# Left join to keep all rows in groberta_new and broberta_new
groberta_new = groberta_new.merge(
    google_data_clean, left_on='roberta_link', right_on='Link', how='left'
).drop('Link', axis=1)

broberta_new = broberta_new.merge(
    bing_data_clean, left_on='roberta_link', right_on='Link', how='left'
).drop('Link', axis=1)

# Verify row counts remain unchanged
print("groberta_new row count:", len(groberta_new))
print("broberta_new row count:", len(broberta_new))

# Check if the two columns are identical
groberta_new['legal_name_match'] = groberta_new['legal_name_x'] ==
groberta_new['legal_name_y']

# Count how many rows have matching and non-matching values
match_summary = groberta_new['legal_name_match'].value_counts()

# Display the rows where the values don't match
non_matching_rows = groberta_new[~groberta_new['legal_name_match']]
```

```
print("Match Summary:")
print(match_summary)

print("\nRows with Non-Matching 'legal_name':")
print(non_matching_rows)

# Rename the column for groberta_new
groberta_new.rename(columns={'legal_name_y': 'legal_name'}, inplace=True)

# Rename the column for broberta_new
broberta_new.rename(columns={'legal_name_y': 'legal_name'}, inplace=True)

# Confirm the changes
print(groberta_new.columns)
print(broberta_new.columns)

import pandas as pd

# Standardize columns for each dataframe
gvaders_new_std = gvaders_new[['legal_name', 'vaders_new_neg', 'vaders_new_pos', 'vaders_new_composite']].rename(
    columns={'vaders_new_neg': 'neg', 'vaders_new_pos': 'pos', 'vaders_new_composite': 'composite'})

bvaders_new_std = bvaders_new[['legal_name', 'vaders_new_neg', 'vaders_new_pos', 'vaders_new_composite']].rename(
    columns={'vaders_new_neg': 'neg', 'vaders_new_pos': 'pos', 'vaders_new_composite': 'composite'})

groberta_new_std = groberta_new[['legal_name', 'roberta_neg', 'roberta_pos', 'roberta_composite']].rename(
    columns={'roberta_neg': 'neg', 'roberta_pos': 'pos', 'roberta_composite': 'composite'})
```

```
broberta_new_std = broberta_new[['legal_name', 'roberta_neg', 'roberta_pos', 'roberta_composite']].rename(
    columns={'roberta_neg': 'neg', 'roberta_pos': 'pos', 'roberta_composite': 'composite'})

ggpt_new_std = ggpt_new[['legal_name', 'gpt_neg', 'gpt_pos', 'gpt_composite']].rename(
    columns={'gpt_neg': 'neg', 'gpt_pos': 'pos', 'gpt_composite': 'composite'})

bgpt_new_std = bgpt_new[['legal_name', 'gpt_neg', 'gpt_pos', 'gpt_composite']].rename(
    columns={'gpt_neg': 'neg', 'gpt_pos': 'pos', 'gpt_composite': 'composite'})

gbert_new_std = gbert_new[['legal_name', 'BERT_Neg', 'BERT_Pos', 'BERT_Composite']].rename(
    columns={'BERT_Neg': 'neg', 'BERT_Pos': 'pos', 'BERT_Composite': 'composite'})

bbert_new_std = bbert_new[['legal_name', 'BERT_Neg', 'BERT_Pos', 'BERT_Composite']].rename(
    columns={'BERT_Neg': 'neg', 'BERT_Pos': 'pos', 'BERT_Composite': 'composite'})

# Combine all standardized dataframes into one
combined_new = pd.concat([
    gvaders_new_std, bvaders_new_std,
    groberta_new_std, broberta_new_std,
    ggpt_new_std, bgpt_new_std,
    gbert_new_std, bbert_new_std
], axis=0, ignore_index=True)

# Verify the combined data
print("Combined Data Sample:")
print(combined_new.head())

# Save to a new dataframe for clarity
final_combined_new = combined_new

# Verify the row count
print(f"Total rows in combined dataframe: {len(final_combined_new)}")

from sklearn.cluster import KMeans
```

```
from sklearn.preprocessing import StandardScaler
import pandas as pd
import matplotlib.pyplot as plt

# Function to perform K-Means clustering and assign risk levels
def kmeans_clustering_risk(data, name='Combined Sentiment Model'):
    # Scale the data
    scaler = StandardScaler()
    sentiment_data_scaled = scaler.fit_transform(data[['neg', 'pos', 'composite']])

    # Apply K-Means Clustering
    kmeans = KMeans(n_clusters=3, random_state=42)
    clusters = kmeans.fit_predict(sentiment_data_scaled)

    # Analyze cluster centers and assign risk levels
    cluster_centers = pd.DataFrame(kmeans.cluster_centers_, columns=['neg', 'pos', 'composite'])
    cluster_centers = cluster_centers.sort_values(by='composite', ascending=True).reset_index(drop=True)
    risk_mapping = {i: risk for i, risk in enumerate(['High', 'Medium', 'Low'])}

    # Map risk levels back to the data
    data['Cluster'] = clusters
    data['Risk_Level'] = data['Cluster'].map(risk_mapping)

    # Aggregate risk level per legal_name
    risk_per_legal_name = data.groupby('legal_name')['Risk_Level'].agg(lambda x: x.mode()[0]).reset_index()
    print("Risk Levels per Legal Name:")
    print(risk_per_legal_name.head())

    # Visualization
    plt.figure(figsize=(8, 6))
    plt.scatter(data['composite'], data['neg'], c=data['Cluster'], cmap='viridis', s=50)
    plt.title(f"{name} - Sentiment Clustering")
    plt.xlabel('Composite Sentiment Score')
```

```
    plt.ylabel('Negative Sentiment Score')
    plt.colorbar(label='Cluster')
    plt.show()

    return risk_per_legal_name

# Apply clustering to the combined dataframe
risk_levels = kmeans_clustering_risk(final_combined_new, name="Unified Sentiment Model")

# Save or display the final risk levels
print("Final Risk Levels Summary:")
print(risk_levels)

"""# Validation"""

import pandas as pd

# Function to calculate accuracy
def calculate_accuracy(predicted_df, actual_df, model_name):
    # Merge on 'legal_name' to compare risk levels
    merged_df = pd.merge(predicted_df[['legal_name', 'Risk_Level']],
                         actual_df[['legal_name', 'risk_level']],
                         left_on='legal_name', right_on='legal_name',
                         how='inner')

    # Compare predicted and actual risk levels
    merged_df['is_correct'] = merged_df['Risk_Level'] == merged_df['risk_level']

    # Calculate percentage of correct matches
    accuracy = (merged_df['is_correct'].sum() / len(merged_df)) * 100

    print(f"Accuracy for {model_name}: {accuracy:.2f}%")
    return accuracy

# List of predicted dataframes and their names
```

```
predicted_dfs = [
    (ggpt_risk_new, "Google GPT New"),
    (bgpt_risk_new, "Bing GPT New"),
    (ggpt_risk, "Google GPT"),
    (bgpt_risk, "Bing GPT"),
    (gbert_new_risk, "Google BERT New"),
    (bbert_new_risk, "Bing BERT New"),
    (gbert_risk, "Google BERT"),
    (bbert_risk, "Bing BERT"),
    (groberta_risk, "Google RoBERTa"),
    (broberta_risk, "Bing RoBERTa"),
    (groberta_risk_new, "Google RoBERTa New"),
    (broberta_risk_new, "Bing RoBERTa New"),
    (gvaders_risk, "Google VADERs"),
    (bvaders_risk, "Bing VADERs"),
    (gvaders_risk_new, "Google VADERs New"),
    (bvaders_risk_new, "Bing VADERs New")
]

# Actual dataframe with true risk levels
actual_df = mtd_companies

# Calculate accuracy for each model
results = {}
for df, name in predicted_dfs:
    accuracy = calculate_accuracy(df, actual_df, name)
    results[name] = accuracy

# Print final results
print("\nSummary of Accuracy:")
for model, acc in results.items():
    print(f"{model}: {acc:.2f}%")

import pandas as pd
```

```
# Function to calculate accuracy
def calculate_accuracy(predicted_df, actual_df):
    merged_df = pd.merge(predicted_df[['legal_name', 'Risk_Level']],
                         actual_df[['legal_name', 'risk_level']],
                         on='legal_name',
                         how='inner')
    merged_df['is_correct'] = merged_df['Risk_Level'] == merged_df['risk_level']
    accuracy = (merged_df['is_correct'].sum() / len(merged_df)) * 100
    return accuracy

# List of models and their respective dataframes
model_pairs = [
    (ggpt_risk, ggpt_risk_new, "Google GPT"),
    (bgpt_risk, bgpt_risk_new, "Bing GPT"),
    (gbert_risk, gbert_new_risk, "Google BERT"),
    (bbert_risk, bbert_new_risk, "Bing BERT"),
    (groberta_risk, groberta_risk_new, "Google RoBERTa"),
    (broberta_risk, broberta_risk_new, "Bing RoBERTa"),
    (gvaders_risk, gvaders_risk_new, "Google VADERs"),
    (bvaders_risk, bvaders_risk_new, "Bing VADERs")
]

# Actual dataframe with true risk levels
actual_df = mtd_companies

# Calculate accuracy differences
print("Performance Comparison Between Original and _New DataFrames:\n")
comparison_results = []

for original_df, new_df, model_name in model_pairs:
    original_acc = calculate_accuracy(original_df, actual_df)
    new_acc = calculate_accuracy(new_df, actual_df)
    improvement = new_acc - original_acc

    comparison_results.append((model_name, original_acc, new_acc, improvement))
```

```
    print(f"{model_name} - Original Accuracy: {original_acc:.2f}%, New Accuracy: {new_acc:.2f}%, Improvement: {improvement:.2f}%")

# Summarize results in a DataFrame
results_df = pd.DataFrame(comparison_results,
                          columns=['Model', 'Original_Accuracy', 'New_Accuracy', 'Improvement'])

# Display Summary
print("\nSummary of Improvements:")
print(results_df.sort_values(by='Improvement', ascending=False))

import pandas as pd

# Function to calculate accuracy per risk level
def calculate_accuracy_per_risk_level(predicted_df, actual_df, model_name):
    # Merge predicted and actual risk levels
    merged_df = pd.merge(predicted_df[['legal_name', 'Risk_Level']],
                         actual_df[['legal_name', 'risk_level']],
                         on='legal_name',
                         how='inner')

    # Add a column to indicate whether the prediction is correct
    merged_df['is_correct'] = merged_df['Risk_Level'] == merged_df['risk_level']

    # Calculate accuracy for each risk level
    results = {}
    for level in ['Low', 'Medium', 'High']:
        subset = merged_df[merged_df['risk_level'] == level]
        if len(subset) > 0:
            accuracy = (subset['is_correct'].sum() / len(subset)) * 100
        else:
            accuracy = 0.0  # No data for this risk level
        results[level] = accuracy

    print(f"Accuracy for {model_name} (by Risk Level): {results}")
```

```
    return results

# List of models and their respective dataframes
model_pairs = [
    (ggpt_risk, ggpt_risk_new, "Google GPT"),
    (bgpt_risk, bgpt_risk_new, "Bing GPT"),
    (gbert_risk, gbert_new_risk, "Google BERT"),
    (bbert_risk, bbert_new_risk, "Bing BERT"),
    (groberta_risk, groberta_risk_new, "Google RoBERTa"),
    (broberta_risk, broberta_risk_new, "Bing RoBERTa"),
    (gvaders_risk, gvaders_risk_new, "Google VADERs"),
    (bvaders_risk, bvaders_risk_new, "Bing VADERs")
]

# Actual dataframe with true risk levels
actual_df = mtd_companies

# Initialize a DataFrame to store the results
accuracy_results = []

# Calculate accuracy for each risk level
for original_df, new_df, model_name in model_pairs:
    original_accuracy = calculate_accuracy_per_risk_level(original_df, actual_df, model_name + " Original")
    new_accuracy = calculate_accuracy_per_risk_level(new_df, actual_df, model_name + " New")

    # Append the results
    accuracy_results.append({
        'Model': model_name,
        'Low_Original': original_accuracy['Low'],
        'Medium_Original': original_accuracy['Medium'],
        'High_Original': original_accuracy['High'],
        'Low_New': new_accuracy['Low'],
        'Medium_New': new_accuracy['Medium'],
        'High_New': new_accuracy['High']
```

```
    })

# Convert results to DataFrame
accuracy_df = pd.DataFrame(accuracy_results)

# Display results
print("\nSummary of Accuracy by Risk Level:")
print(accuracy_df)

# Save the results to a file if needed
accuracy_df.to_csv("risk_level_accuracy_comparison.csv", index=False)

# Plotting
plt.figure(figsize=(10, 6))
bar_width = 0.4

# Bar plot
bars_original = plt.bar(x, original, width=bar_width, label='Original Models', alpha=0.7)
bars_new = plt.bar([i + bar_width for i in x], new, width=bar_width, label='New Models', alpha=0.7)

# Adding text labels (percentages) on top of each bar
for bar in bars_original:
    plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height(),
             f'{bar.get_height():.1f}%', ha='center', va='bottom')

for bar in bars_new:
    plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height(),
             f'{bar.get_height():.1f}%', ha='center', va='bottom')

# Adding details to the chart
plt.xlabel('Risk Levels')
plt.ylabel('Accuracy (%)')
plt.title('Comparison of Original and New NLP Models by Risk Levels')
plt.xticks([i + bar_width / 2 for i in x], labels)
plt.legend()
```

```
# Display the plot
plt.tight_layout()
plt.show()

import pandas as pd

# Function to calculate accuracy
def calculate_accuracy(predicted_df, actual_df):
    merged_df = pd.merge(predicted_df[['legal_name', 'Risk_Level']],
                         actual_df[['legal_name', 'risk_level']],
                         on='legal_name',
                         how='inner')
    merged_df['is_correct'] = merged_df['Risk_Level'] == merged_df['risk_level']
    accuracy = (merged_df['is_correct'].sum() / len(merged_df)) * 100
    return accuracy

# Define all Google and Bing model pairs
google_models = [
    ("Google GPT", ggpt_risk, ggpt_risk_new),
    ("Google BERT", gbert_risk, gbert_new_risk),
    ("Google RoBERTa", groberta_risk, groberta_risk_new),
    ("Google VADERs", gvaders_risk, gvaders_risk_new)
]

bing_models = [
    ("Bing GPT", bgpt_risk, bgpt_risk_new),
    ("Bing BERT", bbert_risk, bbert_new_risk),
    ("Bing RoBERTa", broberta_risk, broberta_risk_new),
    ("Bing VADERs", bvaders_risk, bvaders_risk_new)
]

# Actual risk levels
actual_df = mtd_companies
```

```
# Function to compute average accuracy for a list of models
def average_accuracy(model_list):
    total_accuracy = 0
    count = 0
    for name, original_df, new_df in model_list:
        original_acc = calculate_accuracy(original_df, actual_df)
        new_acc = calculate_accuracy(new_df, actual_df)
        total_accuracy += (original_acc + new_acc) / 2  # Average of original and new
        count += 1
    return total_accuracy / count

# Calculate average accuracies
google_avg_accuracy = average_accuracy(google_models)
bing_avg_accuracy = average_accuracy(bing_models)

# Compare Results
print("Performance Comparison Between Google and Bing DataFrames:")
print(f"Google Average Accuracy: {google_avg_accuracy:.2f}%")
print(f"Bing Average Accuracy: {bing_avg_accuracy:.2f}%")

# Determine which performed better
if google_avg_accuracy > bing_avg_accuracy:
    print("\nGoogle-based models performed better overall.")
elif bing_avg_accuracy > google_avg_accuracy:
    print("\nBing-based models performed better overall.")
else:
    print("\nGoogle and Bing models performed equally well.")

"""Combined models accuracy"""

# Convert final_risk to a DataFrame if it's not already
if isinstance(final_risk, pd.Series):
    final_risk = final_risk.reset_index()  # Converts Series to DataFrame with indices as columns
    final_risk.rename(columns={0: 'risk_cluster'}, inplace=True)  # Rename the value column appropriately
```

```
# Step 1: Verify that final_risk and mtd_companies are DataFrames
print(f"Type of final_risk: {type(final_risk)}")
print(f"Type of mtd_companies: {type(mtd_companies)}")

# Convert final_risk to DataFrame if it's a Series
if isinstance(final_risk, pd.Series):
    final_risk = final_risk.reset_index()  # Converts Series to DataFrame
    final_risk.rename(columns={0: 'risk_cluster'}, inplace=True)  # Adjust column names

# Validate final_risk and mtd_companies
if isinstance(final_risk, pd.DataFrame):
    final_risk.rename(columns={"risk_cluster": "risk_level"}, inplace=True)
    final_risk['legal_name'] = final_risk['legal_name'].str.lower().str.strip()
else:
    raise TypeError("final_risk is not a DataFrame. Check your input.")

if isinstance(mtd_companies, pd.DataFrame):
    mtd_companies.rename(columns={"risk_level": "risk_level_mtd"}, inplace=True)
    mtd_companies['legal_name'] = mtd_companies['legal_name'].str.lower().str.strip()
else:
    raise TypeError("mtd_companies is not a DataFrame. Check your input.")

# Step 2: Identify unmatched legal_names for debugging
unmatched_final = set(final_risk['legal_name']) - set(mtd_companies['legal_name'])
unmatched_mtd = set(mtd_companies['legal_name']) - set(final_risk['legal_name'])

print("Unmatched in final_risk:", unmatched_final)
print("Unmatched in mtd_companies:", unmatched_mtd)

# Step 3: Remove duplicates in legal_name if any
final_risk = final_risk.drop_duplicates(subset=['legal_name'])
mtd_companies = mtd_companies.drop_duplicates(subset=['legal_name'])

# Step 4: Merge the two datasets on legal_name
```

```
comparison_df = final_risk.merge(
    mtd_companies[['legal_name', 'risk_level_mtd']],
    on='legal_name',
    how='inner'
)

# Step 5: Compare risk levels
comparison_df['match'] = comparison_df['risk_level'] == comparison_df['risk_level_mtd']

# Step 6: Calculate accuracy
total_legal_names = len(comparison_df)
matching_count = comparison_df['match'].sum()
accuracy_percentage = (matching_count / total_legal_names) * 100

# Step 7: Display results
print(f"Total legal names compared: {total_legal_names}")
print(f"Number of matches: {matching_count}")
print(f"Accuracy of the combined model: {accuracy_percentage:.2f}%")

# Display sample of comparison results
print(comparison_df.head())

print(set(risk_levels['legal_name']).difference(set(mtd_companies['legal_name'])))

"""_new combined models accuracy"""

# Compare predicted risk levels with actual ones
def compare_risk_levels(predicted_risk, actual_risk):
    # Merge predicted risk with actual risk on 'legal_name'
    comparison = predicted_risk.merge(
        actual_risk[['legal_name', 'risk_level']],
        on='legal_name',
        how='inner'
    )
```

```
    # Compare risk levels
    comparison['Match'] = comparison['Risk_Level'] == comparison['risk_level']

    # Calculate accuracy
    accuracy = comparison['Match'].mean() * 100

    # Display results
    print(f"Accuracy of Unified Model: {accuracy:.2f}%")
    print("Sample of Comparison Results:")
    print(comparison.head())

    return accuracy, comparison

# Run the comparison
accuracy, comparison_results = compare_risk_levels(risk_levels, mtd_companies)

# Print final accuracy
print(f"Final Accuracy: {accuracy:.2f}%")
```