_turnovery _turnovery _turnovery

VILNIUS UNIVERSITY

FACULTY OF MATHEMATICS AND INFORMATICS

DATA SCIENCE STUDY PROGRAMME

Master’s Thesis

Machine Learning Methods for Automated Data Editing of the Turnover of Service Enterprises

Mašininio mokymosi metodai automatizuotam paslaugų įmonių apyvartos redagavimui

Klaudija Švambarytė

Supervisor : Assoc. Prof. Dr. Rūta Levulienė (VU)

Scientific advisor : Ieva Burakauskaitė (SDA, VU)

Vilnius
2025

Acknowledgments

I would like to express my deepest gratitude to all those who have supported me throughout the completion of this master's thesis.

First and foremost, I am grateful to my supervisor, Prof. Dr. Rūta Levulienė, and scientific advisor, Ieva Burakauskaitė, for their guidance, and feedback. I also extend my appreciation to the faculty of Mathematics and Informatics at Vilnius University. It has been an honor to be taught by experts in their respective fields. The knowledge and skills I have gained here have been fundamental to the development of this thesis.

Lastly, I would like to thank my family and loved ones for their continuous encouragement and support throughout this journey. Their belief in me has been a source of motivation during the writing process and beyond.

Summary

National Statistical Institutes (NSIs) face increasing pressure to streamline their data editing processes, as detecting and correcting erroneous entries requires substantial time and resources. This thesis investigates the application of a Random Forest based framework to automate two critical tasks in the data editing workflow: identifying whether the reported value is erroneous, and then imputation of these values.

The classification task demonstrated significant potential for accurately detecting erroneous records, enabling NSIs to focus their human and financial resources on the most critical cases. However, the imputation step faced challenges when predicting small or near-zero errors, particularly in cases that were wrongly classified as erroneous. Although several alternative modeling strategies were tested, none fully resolved these issues, aligning with findings from previous research.

Overall, the study highlights that complete automation of data editing using Random Forest did not achieve desired results.

Keywords: Data Editing; Random Forest; Classification; Imputation; National Statistical Institutes, Error Detection, Machine Learning

Santrauka

Nacionalinės statistikos institucijos (NSI) susiduria su vis didėjančiu spaudimu efektyvinti duomenų redagavimo procesus, nes galimai klaidingų įrašų nustatymas ir taisymas reikalauja didelių išteklių bei papildomo ryšio su respondentais. Šiame darbe nagrinėjamas atsitiktinių miškų metodas, siekiant automatizuoti du esminius duomenų redagavimo uždavinius: nustatyti, ar pateikta apyvartos reikšmė yra klaidinga, ir įrašyti reikšmes klaidingiems stebėjimams.

Pirmasis uždavinys (klasifikavimas) atskleidė reikšmingą potencialą tiksliau nustatyti klaidingas reikšmes, kas itin aktualu NSI praktikoje. Geresnė klaidų identifikacija leidžia sutelkti žmogiškuosius ir finansinius išteklius ten, kur jų tikrai reikia. Tačiau sprendžiant antrą uždavinį (reikšmių įrašymą) susidurta su sunkumais vertinant nedideles klaidas, ypač neteisingai klasifikuoiems duomenims.

Apibendrinant galima teigti, kad visiškas duomenų redagavimo automatizavimas naudojant atsitiktinių miškų metodą neužtikrina reikiamo tikslumo.

Raktiniai žodžiai: Duomenų redagavimas; Atsitiktiniai miškai; Klasifikavimas; Praleistų reikšmių įrašymas; Nacionalinės statistikos institucijos; Klaidų aptikimas; Mašininis mokymasis

List of Figures

List of Tables

Contents

List of abbreviations

E&I	Editing and Imputation
INE	National Statistics Institute of Spain
MAE	Mean Absolute Error
ML	Machine Learning
NRMSE	Normalized Root Mean Square Error
NSI	National Statistical Institute
SMOTE	Synthetic Minority Over-sampling Technique
VAT	Value Added Tax

1 Introduction

Data editing is essential for maintaining the accuracy and reliability of statistical data, which in turn affects the credibility of data-driven analyses and the decisions that follow. Both national and international statistical agencies have acknowledged that incorrect data points can significantly distort final parameters. To address this, there has been an increasing shift towards automated data editing techniques. These methods offer a more targeted and efficient solution compared to traditional manual approaches. They not only reduce the burden on respondents by minimizing the need for follow-up contacts but also help prevent significant errors in published statistics, thereby making the data processing workflow more cost-effective and accurate as data volumes continue to grow [11].

Despite advancements in automated data editing techniques, many National Statistical Institutes (NSIs) still rely heavily on labor-intensive manual error correction processes. As such, there is a need for scalable and efficient solutions to maintain and improve data quality. Endorsed by the *United Nations Economic Commission for Europe* (UNECE) in 2019, the *Generic Statistical Business Process Model* (GSBPM) outlines statistical processes into eight sequential phases: specification of needs, design, build, collection, processing, analysis, dissemination, and evaluation [25]. The processing phase includes the revision of data, new variables creation, weights calibration, parameter estimation, error detection, and error imputation. The latter two are the focus of this thesis.

Existing research has explored various automated data editing techniques. However, limited research focuses on the application of machine learning models for error localization and imputation for NSIs' data editing workflows. This thesis aims to investigate the effectiveness of a Random Forest model in automating two of the following tasks in the data editing workflow: identifying erroneous reported values and imputation of those values. The data utilized consists of encrypted quarterly statistical survey data on service enterprise activities provided by the State Data Agency, Statistics Lithuania (SDA), spanning from 2017 to 2023. The scope is limited to the specific dataset and the application of the Random Forest model; other machine-learning techniques are not explored in this thesis.

This thesis contributes to the literature of statistical data editing by developing and implementing a Random Forest model specifically designed for error detection and imputation in statistical quarterly survey data. Additionally, it presents empirical results derived from real-world data obtained from Statistics Lithuania, thereby demonstrating the practical applicability of machine learning techniques within statistical methodologies. This thesis addresses the following primary research questions:

1. How effective is the Random Forest in identifying erroneous reported values? (Evaluated by sensitivity and balanced accuracy)

2. What is the effectiveness of Random Forest in imputing erroneous data? (Evaluated by Normalized Root Mean Square Error (RMSE) and R^2)

The initial progress of this thesis was presented at the 2023 conference of the Lithuanian Mathematical Society (LMD). The theoretical part of this thesis was conducted using the R programming language. In addition, other software tools were employed; a comprehensive list of the software used can be found in Appendix A.

The remainder of this thesis is organized as follows:

1. **Literature Review**: Reviews data editing practices at NSIs, emphasizing automation.
2. **Nature of data at National Statistical Institutes**: outlines types of errors and SDA's current practices for detecting erroneous values.
3. **Theoretical Part: Random Forest**: outlines the theoretical part of Ranfom Forest.
4. **Dataset Description**: Provides a detailed overview of the dataset used in this research.
5. **Results & Discussion**: Includes exploratory data analysis, findings from the proposed framework, along with a discussion of their implications, and limitations.
6. **Conclusion**: Summarizes the research outcomes and offers recommendations grounded in the results.

2 Literature Review

Accurate statistical data underpins reliable published results, informing decision-making, policy development, and academic research. Because of that importance, NSIs focus on minimizing inconsistencies and errors at every stage of the data editing cycle. Historically, manual editing methods have been used to spot and correct survey data errors, often involving direct contact with respondents to address discrepancies. Although effective, these methods require significant time and resources, with estimates showing 20–40% of NSI resources devoted to data editing [11]. In response, various strategies have emerged to automate parts of this workflow, reducing the burden of manual editing while enhancing error detection and correction. This review examines statistical data editing approaches, including soft- and hard-edit rules (topic of interest at initial stage of the thesis), as well as methods that use machine learning.

2.1 Edit Rules

Edit rules are commonly used to detect errors by checking whether a given value is consistent or not with certain conditions. An edit e can be expressed as:

$$e : x \in S_x,$$

where S_x represents the set of permissible values for x. The variable x may represent a single or multiple values. If e evaluates to false, the edit is violated; if true, the edit is satisfied.

Hard edit rules impose strict constraints that define valid ranges or specific acceptable values for data [23]. For instance, a hard edit might enforce that profit equals total turnover minus total costs, flagging any failure to match this condition as an error. When all edits are satisfied, the record is viewed as consistent and requires no modification. Hard edits function binary: data either pass or fail them.

Soft edit rules, by contrast, are more adaptable, focusing on logical relationships among variables [23]. Rather than using absolute limits, soft edits gauge the likelihood that a value is incorrect by verifying plausible associations. One example is checking if profit is at most half of total turnover, highlighting anomalies for further review rather than immediate fail.

If a record violates one or more edits, then following task is the identification of variables that caused those failures (error localization problem). The generalized Fellegi and Holt (1976) method, which uses confidence weights, has been broadly implemented at NSIs for error localization [23]. Such approaches are integrated into tools like SLICE and the `editrules` package in R [10].

However, such an approach treats all edits (even if they are soft edits) as hard edits [23]. Any edit violation is automatically attributed as an inconsistent record. Recognizing the limitations of this framework, Scholtus explored incorporating soft edits for automatic error localization [20, 21, 22, 23]. The proposed solution includes a cost function that looks at either which soft edits failed or a cost function that would also look at the amount of soft edits that failed. More significant divergence from these criteria increases the suspicion of the record to be erroneous.

In a 2013 study, Scholtus [21] tested the proposed solution on a dataset derived from Dutch Business Statistics 2007 (covering medium-sized wholesale firms of 10–100 employees). The dataset was manually edited and treated as error-free before introducing artificial errors. Four methods were assessed: (1) using only hard edits, (2) treating soft and hard edits alike, (3) assigning the same failure weights to hard and soft edits, and (4) assigning different fixed failure weights to them. Strategy (4) offered the best performance. Nevertheless, a 2015 replication with statistical data without synthetic errors did not replicate these findings, leading Scholtus [22] to describe the outcomes as "*disappointing*." Since then, this line of research has not seen any advancements, and further investigation into balancing soft and hard edits in practical applications remains limited. As such, while this topic was of interest at the initial stages of the thesis, it was not progressed.

2.2 Data Editing with Machine Learning

In recent years, machine learning (ML) has become a key resource for improving statistical data editing in NSIs. By uncovering complex patterns in large datasets, ML algorithms have enabled more accurate and efficient detection of data inconsistencies, complementing traditional data editing techniques.

A prime example is the use of supervised learning models to classify entries as valid or erroneous based on labeled training data. For instance, the National Statistics Institute of Spain (INE) has successfully integrated Random Forest algorithms into their data editing framework by leveraging auxiliary information. Bohnensteffen in her master thesis [2] utilized a Random Forest model within a selective editing framework, focusing on data subsets most likely to affect overall data quality. Selective editing recognizes that not all data points influence dataset integrity to the same degree, allowing NSIs to concentrate resources on the most critical or error-prone segments.

For the INE's implementation, two separate models were created to improve data editing process [2]. The first model classified individual entries as erroneous or correct. Given a roughly 5% rate of erroneous entries, Synthetic Minority Over-sampling Technique (SMOTE) and undersampling were tested to correct the class imbalance. The undersampling approach (splitting the dataset evenly) outperformed the SMOTE-augmented models, achieving a balanced accuracy of 0.771 and a sensitivity of 0.824. Although the SMOTE-augmented models had comparable balanced accuracy, but they exhibited lower sensitivity (0.637–0.767). The second model estimated error magnitude but required additional historical data to increase the number of erroneous instances for training. After expanding the dataset, this model reached R^2 of 0.57, indicating a moderate level of predictive accuracy. This two-tiered approach was deemed successful, leading to its adoption in the INE's ongoing data editing procedures.

Beyond classification and error detection, ML-based methods have also proven effective for imputing missing data. Uogele [26] examined ML imputation methods for monthly statistical survey of trade and catering enterprises at the SDA. Both MissForest and MissRanger demonstrated favorable Normalized Root Mean Square Error (NRMSE) and Mean Absolute Error (MAE) under various missingness levels, highlighting ML's potential to boost data completeness and reliability.

Integrating machine learning into statistical data editing marks a substantial leap forward, allowing NSIs to maintain data quality with fewer manual edits. As datasets grow in size and complexity, manual intervention becomes less practical because of rising resource demands. ML offers a scalable, flexible solution, capable of handling complex, diverse data with minimal oversight.

Moreover, research on ML-driven data editing is advancing rapidly, targeting algorithmic accuracy, interpretability, and efficiency. Future progress is expected to widen the applicability of ML methods to various data types and error patterns, making them even more valuable for NSIs.

3 Nature of data at National Statistical Institutes

To understand the data editing process used by NSIs, this section first outlines the types of errors encountered in statistical data. Following this, it introduces the current data editing methodologies utilized by the SDA in its data editing process.

3.1 Error Types and Missing Values

Understanding the kind of errors helps to better prevent them and build appropriate data editing process. Data errors are discrepancies where reported values diverge from their true values. These errors can be broadly categorized by their type into systematic errors and random errors [11], each originating from distinct sources and necessitating specific strategies for detection and correction.

Systematic Errors are consistent and repeatable inaccuracies that stem from inherent flaws in the data collection process or survey design. Common sources include poorly designed survey questions that may be ambiguous or prone to misinterpretation by respondents. Additionally, inconsistencies in terminology or definitions across various departments or sections of a survey can cause discrepancies in the collected data. Unit measurement errors, such as reporting amounts in incorrect units (e.g., euros instead of dollars), further contribute to systematic errors. Addressing these requires revising survey instruments for clarity, standardizing definitions and units across all sections, and conducting thorough pilot testing to identify and rectify potential biases before full-scale data collection.

Random Errors are unpredictable and occur sporadically without a consistent pattern. They arise from unforeseen factors that affect individual data points, such as typographical mistakes, misreporting, or accidental omissions. For example, a respondent might inadvertently enter an extra digit in a numerical response or skip a question altogether. Detecting random errors often requires robust statistical techniques, such as outlier detection methods, consistency checks, and validation against additional data sources. Once identified, these errors can be corrected through methods like data imputation or by cross-verification.

Influential Errors are a subset of data inaccuracies that exert a disproportionate impact on the final statistical outputs. Influential errors often result from erroneous data points that lie far from the central tendency of the dataset or from data points that disproportionately affect model parameters in statistical analyses. Identifying influential errors typically involves using selective editing methods that incorporate score functions, which quantify the influence and error of data points. In some cases, it may be appropriate to estimate the true value, while in others, if the error has a high score, re-collecting the data from the original respondent may be necessary to obtain accurate information.

Missing Values also constitute a significant challenge in survey data collected by NSIs. Missing data can occur when the true value is unknown, unavailable, or difficult for respondents to provide. Common causes include non-response to specific survey questions, data collection errors, or the inapplicability of certain questions to particular respondents. Missing data can be categorized based on the mechanism that leads to the absence of values, as defined by Rubin [19]. The categories include Missing Completely at Random (MCAR), Missing at Random (MAR), and Missing Not at Random (MNAR). Addressing missing data requires appropriate imputation techniques, ranging from simple methods like mean imputation to advanced methods such as Multiple Imputation by Chained Equations (MICE) and machine learning-based approaches like Random Forest Imputation.

3.2 Current Methods for Error Detection

The Editing and Imputation (E&I) process is designed to improve data quality by identifying and correcting possibly erroneous or missing values [7]. The process begins with initial editing, which involves checking for errors related to specific domains and systematic mistakes in raw data. After this, selective editing is applied to identify outliers in the data. The next phase, interactive editing, involves experts analyzing these outliers and making corrections either by re-contacting data sources or using more automated methods. In the final stage, macro-editing, the focus shifts to examining population-level aggregates and estimates for errors, utilizing additional information to ensure accuracy. If the aggregated data do not meet the required standards after this stage, the E&I process is repeated. Visualization of the process can be seen in Figure 1.

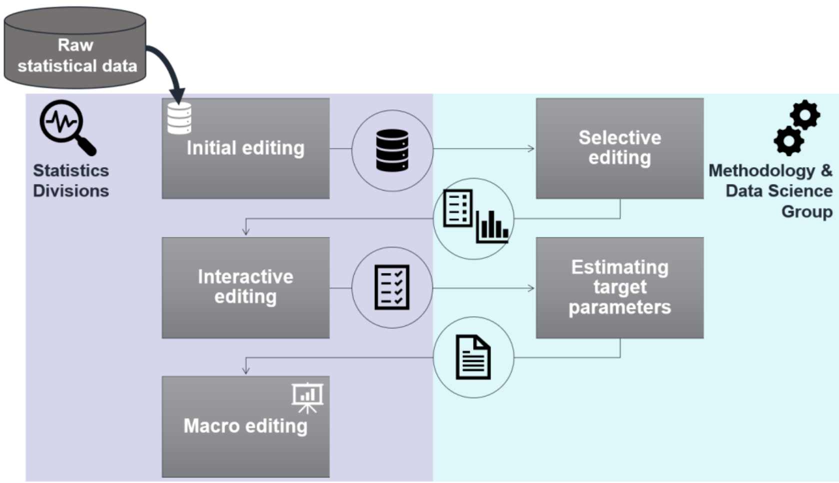


***Figure 1 .** Editing and Imputation Process Employed at SDA [7]*

Currently, VDA employs logical, arithmetic and mathematical rules to detect errors [7]. Besides those quartile, Hidiroglou-Berthelot, and selective editing methods are used, which are described in more detailed below.

Quartile Method detects outliers based on the distribution of a given variable. This technique utilizes quartiles to partition the data into four equal parts, emphasizing the first quartile (Q_1) and the third quartile (Q_3), which represent the 25th and 75th percentiles, respectively. Outliers are identified by calculating the interquartile range (IQR), defined as the difference between Q_3 and Q_1:

$$\text{IQR} = Q_3 - Q_1.$$

An observation x is considered an outlier if it falls outside the range defined by:

$$x < Q_1 - k \times \text{IQR} \quad \text{or} \quad x > Q_3 + k \times \text{IQR},$$

where k is a multiplier commonly set to 1.5 for mild outliers and 3 for extreme outliers. For instance, in this study, a company's quarterly total turnover that significantly deviates beyond $Q_3 + 3 \times \text{IQR}$ or below $Q_1 - 3 \times \text{IQR}$ would be flagged as an outlier, necessitating further review.

Hidiroglou-Berthelot compares current observations with historical data, using previous measurements as benchmarks to identify significant discrepancies that may indicate errors. Specifically, the method calculates the ratio R_k for each unit k as:

$$R_k = \frac{y_k}{x_k},$$

where y_k is the current value of the variable of interest for unit k, and x_k is the corresponding historical value, serving as an auxiliary variable. In this research, the total turnover from the corresponding quarter of the previous year ($x_k - 4$) as well as last quarter ($x_k - 1$). Outliers are detected by identifying ratios that significantly deviate from expected norms.

Selective editing is an error detection strategy that focuses on prioritizing data units with a high probability of containing significant errors, thereby optimizing resource allocation for manual verification. This approach targets subsets of data that are most influential on the final estimates or that exhibit suspicious discrepancies when compared to reliable external sources.

In this study, Value Added Tax (VAT) data are used as auxiliary information within the selective editing framework. VAT data are typically subject to stringent verification processes due to tax compliance requirements, rendering them highly reliable for cross-referencing financial information.

The selective editing process involves comparing the reported values in the dataset with corresponding VAT records. An example of such an approach defines the score s_k as:

$$s_k = \mathbb{E}\left[d_k \cdot |Y_k^{\text{raw}} - Y_k^{\text{true}}| \,\middle|\, \mathbf{x}_k \right], \tag{1}$$

where:

- $\mathbb{E}[\cdot \mid \mathbf{x}_k]$ denotes the expected value conditioned on the auxiliary information $\mathbf{x}_k$.
- Y_k^{raw} is the random variable representing the raw collected data for unit k.
- Y_k^{true} is the random variable representing the true value for unit k.
- $\mathbf{x}_k$ represents the auxiliary information available for unit k in this VAT variable.

Method	TP	FP	TN	FN	Precision	Sensitivity	Accuracy	Bal. Acc.	Specificity
Quartile Method	21	795	6,302	121	0.026	0.148	0.873	0.518	0.888
H-B with $x_k - 1$	213	3,734	4,206	162	0.054	0.568	0.531	0.549	0.530
H-B with with $x_k - 1$	267	6,011	5,670	248	0.042	0.518	0.487	0.502	0.485
Selective Editing with VAT	320	3,600	6,040	175	0.082	**0.646**	0.627	**0.636**	0.626

Table 1 . *Performance Metrics for Methods Currently Used by SDA to Detect Errors*

Between 2017 and 2023, all three methods were applied to detect erroneous values as their outputs were provided by SDA. The comparative analysis aimed to determine each method's effectiveness in flagging incorrect entries and examining how many of those flagged items were ultimately corrected. Performance metrics were then computed to evaluate how well each technique identified errors.

The Quartile Method achieved an accuracy of 0.873, largely because it accurately classified a high number of true negatives. However, its sensitivity of 0.148 reveals major limitations in detecting erroneous values, and its precision of 0.026 indicates a substantial number of false positives. As a result, despite effectively confirming correct values, it remains suboptimal for uncovering errors.

The Hidiroglou-Berthelot method employing last year's same quarter turnover produced a precision of 0.054 and a sensitivity of 0.568, culminating in an accuracy of 0.531. These figures signal an improved balance over the Quartile Method in locating incorrect entries, though a sizable count of false positives persists. While its enhanced sensitivity reflects fewer overlooked errors, it still misses a considerable portion of them.

When combined with last quarter's turnover the Hidiroglou-Berthelot method attained a precision of 0.042 and a sensitivity of 0.518, generating an accuracy of 0.487. This variant catches more total outliers but also yields a high volume of false positives. Although it aims to reconcile thorough detection with minimizing false alarms, numerous inaccuracies remain undiscovered.

Among all approaches, Selective Editing with VAT demonstrated the strongest results, marked by a precision of 0.082 and a sensitivity of 0.646. With an accuracy of 0.627 and a balanced accuracy of 0.636, this method excels at recognizing actual errors while keeping false positives in check. Although some errors still evade detection, it outperforms the other techniques in this comparison.

Based on these metrics, Selective Editing with VAT presents the most favorable compromise between identifying true errors and avoiding unnecessary edits. Its higher precision curtails unwarranted corrections, whereas its increased sensitivity means fewer undetected issues. Despite the remaining gaps in overall recall, this method stands as the most balanced option.

Performance of these methods will be further considered in subsequent discussions, particularly regarding strategies to refine error detection.

4 Random Forest: Theoretical Foundations

Why Random Forest for NSIs? Random Forest effectively handles large and diverse datasets. Random Forest's ensemble design and resilience to noise help detect genuine patterns amidst data variability. Its feature importance measures can guide NSIs in identifying which variables most strongly signal errors. Hyperparameter tuning, such as adjusting the number of trees or maximum depth, further enhances the model's ability to capture underlying error patterns. This blend of robustness, interpretability, and flexibility makes Random Forest a valuable tool for improving data quality and reliability in official statistics. This section goes more in-depth of the theoretical underpinnings of Random Forest.

4.1 Random Forest Algorithm

Random Forest is a non-parametric, supervised machine learning method recognized for its strong predictive performance and resilience to overfitting. Originally introduced by Breiman [6], it combines multiple decision trees through the bagging (Bootstrap Aggregation) technique to reduce the variance of individual trees without substantially increasing their bias. Over the past two decades, Random Forest has gained significant traction in both academia and industry owing to its ease of implementation, intrinsic feature importance metrics, and robustness in high-dimensional settings.

Decision Trees as Building Blocks.

At the heart of Random Forest are decision trees. Each tree partitions the feature space into regions intended to be as homogeneous as possible with respect to the target variable. For classification, commonly used splitting criteria include Gini impurity and entropy, whereas regression trees frequently adopt Mean Squared Error (MSE) as an impurity measure [4]:

- Gini Impurity (Classification):

  $$I_{\text{Gini}}(m) = 1 - \sum_{k=1}^{K} p_{mk}^2,$$

  where p_{mk} denotes the proportion of samples belonging to class k within node m, and K is the total number of classes. A lower impurity indicates a more "pure" node.

- MSE (Regression):

  $$I_{\text{MSE}}(m) = \frac{1}{|m|} \sum_{(x_i,y_i)\in m} (y_i - \bar{y}_m)^2,$$

  where $\bar{y}_m$ is the mean response in node m. Minimizing MSE at each split yields partitions in which data points exhibit relatively similar response values.

Although individual trees can be highly interpretable, they often overfit the training data by capturing noise or anomalous patterns. This overfitting results in high variance, where the model performs exceptionally well on training data but poorly on new, unseen datasets. Random Forest addresses this limitation by creating an ensemble of diverse trees, thereby averaging out individual tree errors and achieving a more stable and accurate prediction.

Bagging for Variance Reduction. Bagging, short for Bootstrap Aggregating,, serves as the foundational ensemble technique in Random Forest to reduce the variance component of the model error [13]. The primary idea behind bagging is to train multiple models on different subsets of the training data and then aggregate their predictions to form a final output. This process enhances the overall stability and accuracy of the model. Given a dataset

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N},$$

multiple bootstrap samples $\mathcal{D}_b$, each of size N, are drawn *with replacement*. Consequently, each $\mathcal{D}_b$ includes a subset of unique observations (some repeated) from the original dataset. Each tree T_b is trained on a distinct bootstrap sample, yielding a "forest" of trees that are diverse in their structure and learned rules. By averaging (for regression) or voting (for classification) across these trees, the ensemble demonstrates markedly lower variance than any single tree [13].

Random Subset of Features. In addition to bootstrapping the data, Random Forest introduces another layer of randomness by selecting a random subset of features at each split in a decision tree. This parameter, denoted as m_{try}, controls the number of features considered when determining the best split at each node [6]. The selection of m_{try} is critical for enhancing the diversity of the trees and improving the overall performance of the ensemble.

By limiting the number of features evaluated at each split, m_{try} decreases the likelihood that multiple trees will select the same dominant features. This reduction in correlation among trees enhances the diversity of the ensemble, thereby amplifying the benefits of averaging or voting.

Evaluating a smaller subset of features at each split reduces the computational burden, particularly in high-dimensional datasets where the number of features p is large. This efficiency gain allows for faster tree construction and enables the handling of more complex datasets without prohibitive computational costs.

Prediction Aggregation and OOB Error. Once all trees are built, Random Forest aggregates their individual predictions to obtain a final output. In classification, the ensemble prediction is determined via majority vote:

$$\hat{y} = \text{arg}\max_{k} \sum_{b=1}^{B} \mathbb{I}\big(T_b(\mathbf{x}) = k\big),$$

while in regression, the average of all tree outputs is taken:

$$\hat{y} = \frac{1}{B} \sum_{b=1}^{B} T_b(\mathbf{x}).$$

This aggregation harnesses the "wisdom of the crowd," typically resulting in higher accuracy and stability compared to a single decision tree. Moreover, because each tree is trained on a distinct bootstrap sample, the $\sim 1/3$ of observations omitted from a particular tree's training set (known as *out-of-bag* samples) can be used to estimate performance without an external validation set [6]. This out-of-bag (OOB) error assessment provides a convenient and efficient measure of generalization error.

Variable Importance and Feature Selection. Beyond making predictions, Random Forest offers valuable insights into the relative importance of input features, enhancing model interpretability and aiding in feature selection. Two primary metrics for assessing feature importance:

- Mean Decrease in Impurity (MDI): This metric quantifies the total reduction in impurity (e.g., Gini impurity or MSE) contributed by each feature across all splits in all trees.
- Mean Decrease in Accuracy (MDA): Assesses how random permutation of a feature in the OOB set degrades prediction accuracy. A larger drop indicates a more critical feature [6, 18].

These importance scores not only facilitate model interpretation but also help practitioners identify the subset of features that most influence predictions.

4.2 Hyperparameter Tuning

Hyperparameter tuning in Random Forest centers on balancing predictive performance, bias-variance trade-offs, and computational cost:

- *Number of Trees*: Increasing number of trees typically leads to a reduction in variance, as the ensemble's averaging effect becomes more pronounced. This means that the predictions become more stable and less sensitive to the fluctuations in the training data. However, the relationship between number of trees and performance exhibits diminishing returns; beyond a certain point, adding more trees yields minimal improvements in accuracy while significantly increasing computational costs [9]. Practically, number of trees is often set to values ranging from 100 to 1000, depending on the size and complexity of the dataset. Cross-validation techniques can help identify an optimal number of trees by evaluating performance gains against the associated computational overhead.
- *Number of Features per Split (m_{try})*: The parameter m_{try} determines the number of features randomly selected at each split in a decision tree. For classification tasks, a common default is $m_{\text{try}} = \sqrt{p}$, and for regression tasks, $m_{\text{try}} = p/3$, where p represents the total number of features [6]. variance:
  - Reduces Correlation: By limiting the number of features considered at each split, m_{try} decreases the correlation between individual trees within the forest. Lower correlation enhances the diversity of the ensemble, thereby improving its overall predictive power and reducing variance.
  - Increases Bias: Conversely, if m_{try} is set too low, the model may become too simplistic, leading to higher bias as the trees may not capture all relevant patterns in the data.

  Empirical tuning of m_{try} can yield significant performance improvements, especially in high-dimensional datasets where feature selection plays a pivotal role in model accuracy and efficiency.

- *Minimum Samples per Leaf (n_{min})*: The hyperparameter n_{min} specifies the minimum number of samples required to form a leaf node in each decision tree. Setting a higher n_{min} imposes a constraint that prevents the tree from creating overly specific partitions that capture noise in the training data [16]. This reduces the model's propensity to overfit, as larger leaf sizes promote more generalized rules. However, excessively large values of n_{min} can lead to underfitting, where the model fails to capture important nuances in the data. Conversely, smaller values of n_{min} allow for more detailed splits, potentially increasing accuracy on training data but risking overfitting. Therefore, n_{min} must be tuned to strike an optimal balance between capturing meaningful patterns and maintaining generalization capability.

- *Maximum Tree Depth (d_{max})*: The maximum depth of the trees, denoted by d_{max}, controls the extent to which a tree can grow before ceasing to split further. Limiting d_{max} constrains the complexity of each tree, thereby mitigating the risk of overfitting, especially in datasets with significant noise or high dimensionality [16]. A deeper tree can model more intricate relationships within the data, potentially increasing accuracy but also elevating the risk of capturing spurious patterns. On the other hand, a shallower tree may underfit by being too simplistic, failing to capture essential data structures. Selecting an appropriate d_{max} is crucial for ensuring that each tree contributes effectively to the ensemble without compromising its generalization performance.

- *Impurity Measure*: Measure influences how splits are evaluated within each decision tree. Gini, entropy, or MSE may be chosen depending on the task [4].

These methods are typically combined with cross-validation techniques, such as k-fold cross-validation, to ensure that the selected hyperparameters generalize well to unseen data. Common strategies to identify optimal hyperparameter configurations are grid search, random search, or Bayesian optimization [1].

When tuning hyperparameters, practitioners must consider the computational resources and time constraints, especially with large datasets. Increasing the number of trees (B) or the depth of trees (d_{max}) can exponentially increase training time and memory usage. Therefore, a pragmatic approach often involves starting with default hyperparameter values and iteratively adjusting them based on cross-validated performance metrics. Additionally, leveraging parallel computing capabilities can significantly expedite the tuning process.

4.3 Theoretical Guarantees and Applications

Random Forest boasts several theoretical advantages that contribute to its empirical success. Random Forest offers several theoretical advantages:

- *Consistency*: As $B \to \infty$, the predictions converge to the true function under certain conditions [24].

- *Bias-Variance Reduction*: By averaging many uncorrelated models, Random Forest reduces variance substantially while only minimally increasing bias [5].

- *Robustness to Overfitting*: The combination of bootstrap sampling, random feature selection, and averaging makes Random Forest much less prone to overfitting than a single, unconstrained decision tree. [9].

Numerous studies confirm these properties in practice, demonstrating Random Forest's effectiveness across diverse fields [18]. Extensions and variants, such as Extra Trees [15], Weighted Random Forests [8], and Survival Random Forests [17], have further broadened its applicability.

For national statistical institutes, Random Forest provides a robust framework for identifying and correcting potentially erroneous data points. By modeling complex interactions among features, outliers or inconsistencies can be flagged when they deviate from learned patterns. Variable-importance metrics help detect which features best indicate data quality, guiding the definition or refinement of hard and soft edit rules. Additionally, Random Forest is well-suited to high-dimensional, heterogeneous data frequently encountered in administrative and survey settings [9], making it a versatile option for modern statistical data editing workflows.

5 Dataset

The dataset used in this study comprises quarterly data from a statistical survey on service enterprise activities conducted by the SDA between 2017 and 2023. Data was provided as seven Excel files, each corresponding to a different year, with three sheets: raw data, edited data, and an "outliers" sheet indicating which values were flagged as outliers by currently used methods by thee SDA (see section 3.2). This data then were merged based on ID; any ID inconsistencies (for example, an ID present in the raw file but missing in the edited file) were removed. Variables not appearing in all years were also excluded.

The primary variable of interest is total turnover for each accounting quarter. However, to improve the robustness of error detection, several auxiliary variables have been included too, for those see Table Table 2 .. These auxiliary variables are employed both to predict potential errors in the target variable (i.e., to identify whether a reported turnover is erroneous) and to assist in later stage of imputation. For instance, *turnover_y* captures the turnover from the same quarter of the previous year, providing a historical performance benchmark, while *VAT* adds additional financial information. In addition to the original variables, a set of derived or composite predictors were generated to enrich the analytical space and enhance modeling performance. These newly created features include:

- *rel_change_turnover* – the relative change in turnover compared to a baseline (e.g., previous quarter),
- *val_VAT_interaction – val * VAT*,
- *turnover_pe* – turnover per employee,
- *change_turnover* – an absolute or percentage change in turnover from current period to period of last year's same quarter.

A detailed description of all variables is provided in Table Table 2 .. Additional exploratory data analysis is presented in the Results and Discussion section of this thesis.

Variable Name	Description
	Survey Variables
turnover	Total turnover of the current quarter.
turnoverR	Edited total turnover of the current quarter.
turnover_y	Total turnover for the same quarter of the previous year.
val	Total hours worked by the employee(s).
VAT1m	Value-added tax for the first month of the quarter.
VAT2m	Value-added tax for the second month of the quarter.
VAT3m	Value-added tax for the third month of the quarter.
VAT	Value-added tax for the quarter.
DSK_SOD	Number of employees employed by the company.
	Identification and Additional Variables
ID	Encrypted unique identifier assigned to each company.
quarter	Fiscal quarter of the year, indicated as integers from 1 to 4.
status	Indicator of the company's active status, where 1 signifies active and 0 signifies inactive.
comment	Indicator of whether there was a comment left by an expert regarding the enterprise.
EVRK2_perk	Encrypted 4-character code representing the company's economic activity sector, used for categorization.
inspected	Indicator of whether the company was selected for inspection, where 1 indicates selection and 0 indicates non-selection.
answered	Indicator of whether the company submitted a completed report, where 1 indicates submission and 0 indicates non-submission.
change_turnover	Absolute difference in total turnover between the current quarter and the same quarter in the previous year.
rel_change_turnover	Relative difference in total turnover between the current quarter and the same quarter in the previous year.
turnover_pe	Turnover per employee for each quarter.
val_VAT_interaction	Interaction between hours worked and VAT.

Table 2 . *Variables and Their Descriptions*

6 Results & Discussion

This section begins with exploratory data analysis of the dataset, including correlation and visualization analysis. Following this, the results of RF classification and RF regression models are discussed, highlighting their performance metrics and the most significant predictors. Finally, the implications of these findings are evaluated in the context of the research objectives, providing insights into the effectiveness of the models and potential areas for further investigation.

6.1 Exploratory Data Analysis

Before proceeding with model building, exploratory data analysis was performed, and missing values were addressed.

Despite the original size of the dataset, the initial version contained missing values. Approximately 7.5% of the key variable of interest (turnover) was missing. Previous studies and internal SDA practices indicate that advanced imputation techniques, such as *missForest* and *missRanger*, are effective for detecting and imputing missing data [26]. However, in this thesis, these incomplete turnover records were removed rather than imputed.

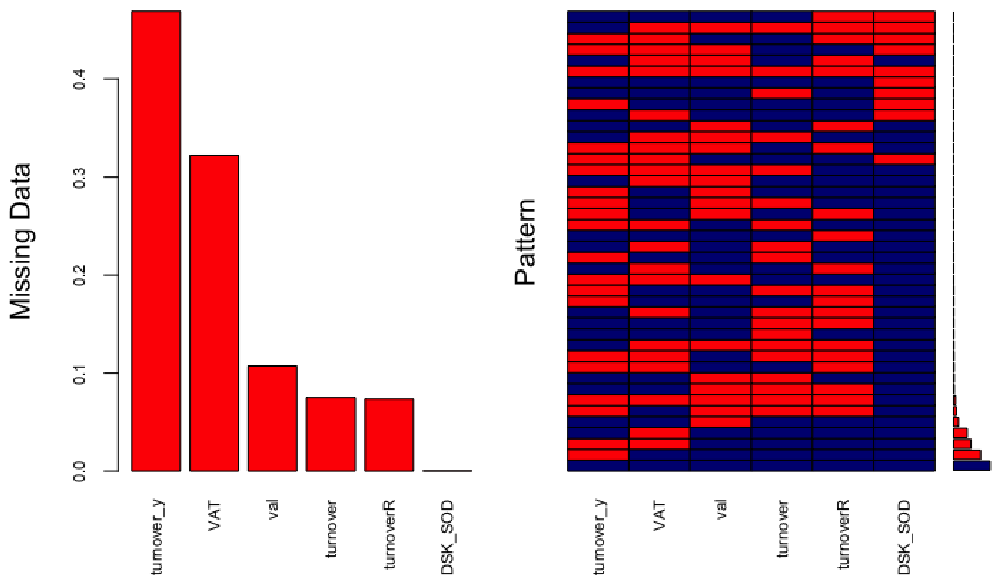


Figure 2 . *Missing Data and its Patterns in Key Variables*

Figure 2 illustrates the quantity and pattern of missing data. Notably, 47% of last year's turnover values were missing. If an enterprise was not selected for the previous year's subset, the corresponding turnover data point became unavailable. For this variable, missing values were imputed to match turnover, while all other missing values were removed. After removing these incomplete records, the dataset was reduced to approximately 80.6k observations, thereby retaining the most complete and reliable subset of the original data.

After addressing missing values, a correlation analysis was undertaken to identify redundancies. Although monthly VAT data was provided in addition to quarterly VAT, these monthly variables

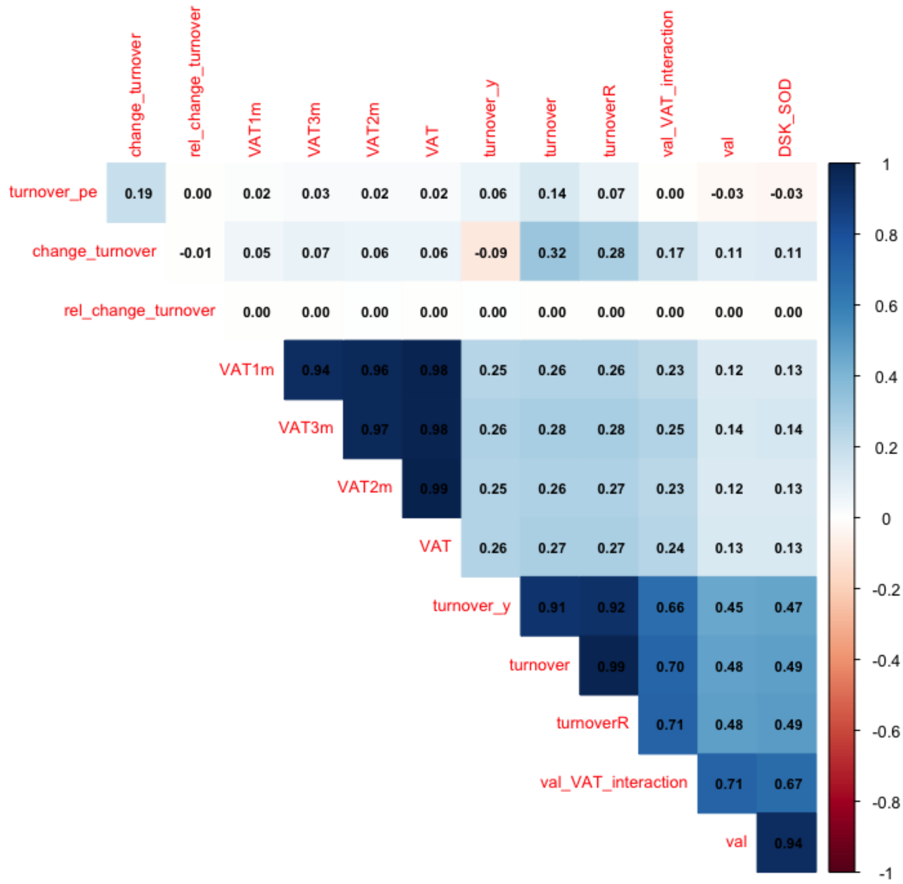


Figure 3 . *Variable Correlations*

exhibited a high correlation with the quarterly totals. Consequently, the monthly *VAT1m, VAT2m, VAT3m* variables were removed (see Figure 3 for correlations). By retaining only the quarterly totals, excessive duplication of information is avoided. A strong correlation can be seen between turnover and last year's turnover (*turnover_y*) as the missing values for the latter variable were imputed to equal (*turnover*).

The dataset also includes several categorical variables that offer insights into the classification and activity of enterprises. Notably, *EVRK2_perk* specifies enterprise sectors. Scatter plots were used for visual analysis to determine whether there are different relationships between turnover and last year's turnover, as well as between turnover and VAT variables, for the top four most frequent enterprise sectors. Figure 4 includes data across all years and shows a strong linear relationship between VAT and turnover. The "EBca" enterprise sector has one outlier that was not edited (it is not erroneous). Additionally, it can be observed that the turnover spread differs between groups; some sectors have higher turnover than others. Figure 5 shows a more moderate relationship between turnover and last year's turnover (values that were missing and imputed were not included in the visualization). Scatter plots suggest that VAT has a stronger relationship with the target variable.

Some of the scatter plots suggest the presence of outliers within certain economic sectors. It is important to note that outliers in this context may not necessarily be erroneous values; they may simply reflect atypical but valid enterprise performances. Since the primary aim of this thesis is to detect erroneous data rather than strictly eliminate all outliers, these data points were retained

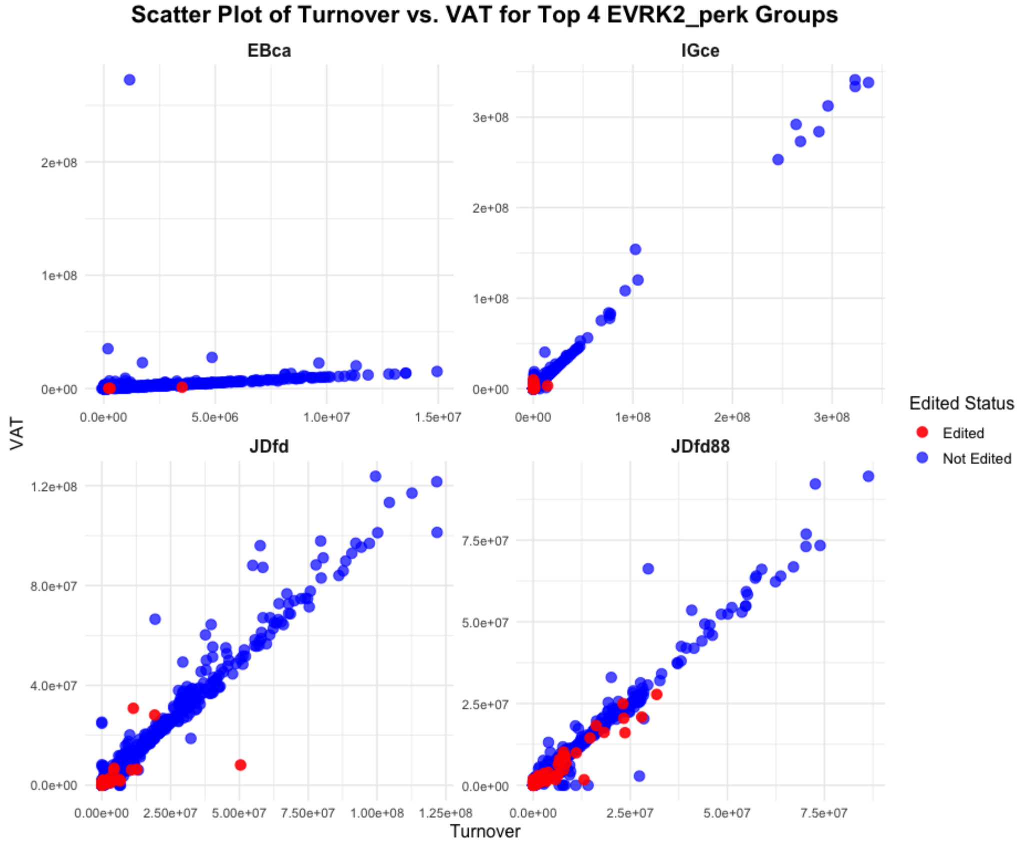


Figure 4 . *Scatter Plot of Turnover vs. VAT for the Top Four Enterprise Sectors*

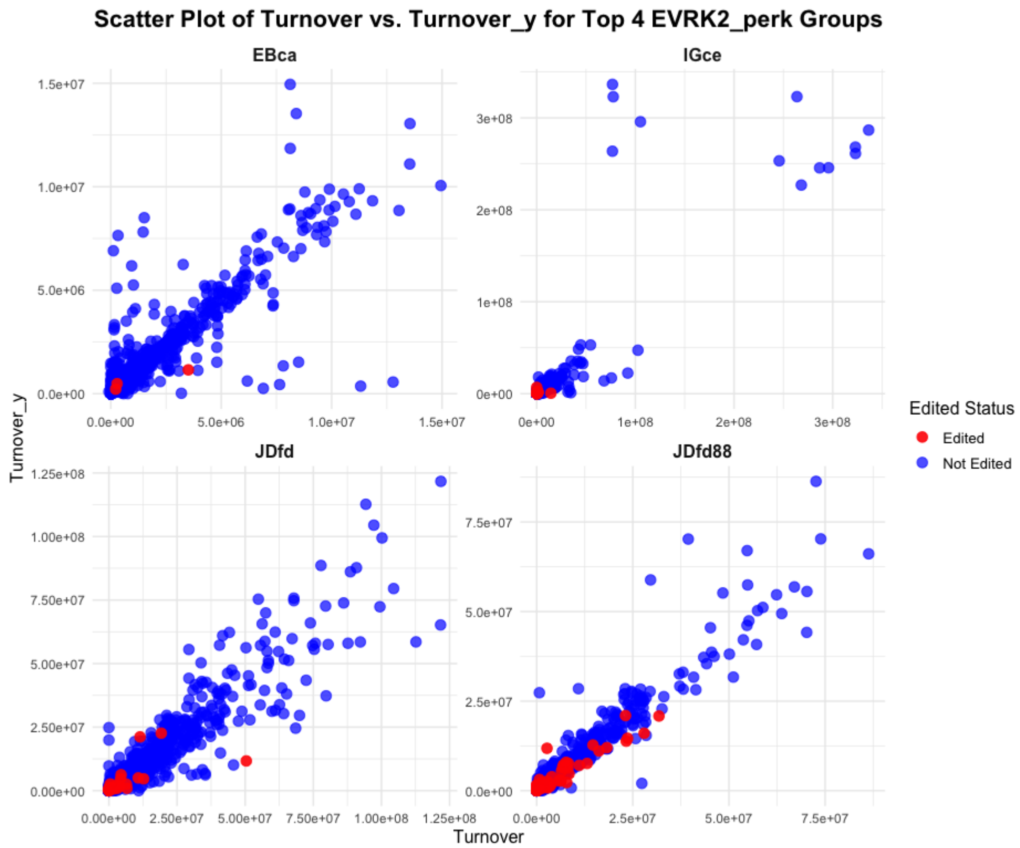


Figure 5 . *Scatter Plot of Turnover vs. Last Year's Turnover for the Top Four Enterprise Sectors*

to preserve the natural variability of the dataset. Furthermore, the RF algorithm selected for this study is relatively robust to outliers. In particular, RF aggregates predictions over multiple decision trees, thus mitigating the undue influence of a small number of extreme observations. As a result, potential outliers that remain in the dataset are less likely to skew the model predictions, allowing for a comprehensive detection of truly erroneous values without discarding data points that may represent legitimate but uncommon behaviors.

Upon further investigation, it was found that the dataset encompassed a total of 306 different enterprise sectors. However, certain sectors were very sparsely represented, with fewer than six observations recorded across all years. These minimally populated categories pose significant challenges for model training and validation, especially for methods like RF, which rely on partitioning data into multiple trees. Groups with extremely limited data points do not provide sufficient variety for robust parameter estimation, often leading to unstable or biased predictions.

To address these challenges and maintain a more balanced distribution of observations per category, enterprise sector groups with fewer than six observations were consolidated into a single overarching category labeled “Other.” This aggregation ensures that each remaining classifier has enough instances to support the modeling process while still retaining high-level distinctions among primary service enterprise types.

6.2 RF Model for Error Classification

A Random Forest classification model was developed to identify erroneous turnover values in the dataset. The target variable, *edited*, was converted into a categorical factor with two levels: *Class1* and *Class0*. Subsequent analysis of class distributions within each subset revealed a significant imbalance, with the majority class (*Class0*) dominating, and only 1.40% of the observations having been edited from the original submitted value.

To ensure effective model training and evaluation, the dataset was divided into training and testing subsets. The data was shuffled, and 85% was allocated to training. This stratified partitioning was performed using the `createDataPartition` function from the `caret` package, ensuring the distribution of the target variable (*edited*) remained consistent across both subsets. The training subset contained 1.39% of the minority class, while the test subset contained 1.47%.

Class imbalance poses a substantial challenge in classification tasks, often leading to models biased toward the majority class. Such disparities necessitate strategies to address class imbalance, thereby enhancing the model’s ability to accurately predict the minority class. To mitigate this issue, both oversampling and undersampling techniques were employed to create balanced training subsets with varying minority-class proportions.

The Synthetic Minority Over-sampling Technique (SMOTE) was utilized to generate synthetic samples of the minority class (*Class1*). This method creates plausible synthetic instances by interpolating between existing minority class observations, thereby enriching the feature space without merely duplicating existing samples. Three oversampled training subsets were generated, targeting minority class proportions of 40%, and 20%.

Conversely, undersampling was employed to reduce the number of majority class instances

Table 3 . *Class Distributions and Imbalance Ratios in Training Datasets*

Name	Class0	Class1	Imbalance Ratio
SMOTE_40	41021	27493	0.401
SMOTE_20	54772	13742	0.200
Under_60	633	950	0.600
Under_40	1425	950	0.400
Under_20	3800	950	0.200
Original	67564	950	0.014

(*Class0*) in the training data. This approach involved selectively sampling a subset of the majority class to achieve desired minority class proportions of 60%, 40%, and 20%. By limiting the dominance of the majority class, undersampling helps prevent the model from becoming biased toward predicting the majority class, thereby improving its sensitivity to the minority class. The class distributions for these datasets are shown in Table 3.

For each training dataset: original; oversampled (SMOTE_40, SMOTE_20); and undersampled (Under_60, Under_40, Under_20), an RF classification model was trained. The training process involved the following steps:

1. **Feature Selection**: Relevant predictor variables were selected based on their potential influence on the target *edited* variable. These included: *quarter*, *EVRK2_perk*, *status*, *turnover*, *turnover_y*, *val*, *VAT*, *DSK_SOD*, *change_turnover*, *rel_change_turnover*, *val_VAT_interaction*, and *turnover_pe*.

2. **Hyperparameter Tuning**: A grid search was conducted to optimize key RF hyperparameters, including:

   - the number of variables randomly sampled at each split (`mtry`), tested from 1 to 13,
   - the split rule (`splitrule`), set to `"gini"`,
   - the minimum node size (`min.node.size`), tested from 1 to 10.

   The objective was to find a configuration that maximizes the model's sensitivity, thereby enhancing its capacity to detect erroneous values.

3. **Cross-Validation**: Five-fold cross-validation was employed to assess model performance during training.

4. **Class Weights**: Although class imbalance was addressed through oversampling and undersampling, class weights were also computed (inversely proportional to class frequencies) for each dataset. These weights further balanced the influence of each class during model training.

6.2.1 Model Evaluation

Post-training, each RF classifier was evaluated on cross-validation and test dataset to assess its generalizability. The evaluation metrics included:

- **Confusion Matrix**: Provides a summary of prediction results, highlighting true positives, true negatives, false positives, and false negatives.
- **Sensitivity (Recall)**: Measures the proportion of actual erroneous values correctly identified by the model.
- **Specificity**: Assesses the proportion of non-erroneous values correctly identified.
- **Balanced Accuracy**: Represents the average of sensitivity and specificity, offering a more balanced measure of model performance, particularly in datasets with class imbalance.
- **Area Under the Curve (AUC)**: Quantifies the overall discriminative ability of the model.

	Original Data	Under_60	Under_40	Under_20	SMOTE_40	SMOTE_20
min.node.size	1	8	4	3	2	3
mtry	13	8	8	8	8	8
Accuracy	0.960	0.657	0.736	0.816	0.984	0.985
Sensitivity	0.348	**0.713**	**0.713**	0.567	0.011	0.011
Specificity	0.969	0.657	0.886	0.820	0.990	0.999
AUC	0.811	0.762	0.794	0.803	0.674	0.674
Bal. Accuracy	0.659	0.685	**0.725**	0.694	0.505	0.505

Table 4 . *Model Performance on Test Data for Different Training Datasets*

When interpreting the hyperparameter settings seen in Table 4, it is useful to consider how each parameter (*min.node.size* and *mtry*) influences the complexity and generalization of the Random Forest (RF) model. Specifically, *mtry* (the number of predictors randomly sampled at each split) determines how many features are considered at each node, while *min.node.size* (the minimum number of observations allowed in a leaf node) regulates the depth of each decision tree.

For the original dataset, the best-performing configuration used an *mtry* of 13 and a *min.node.size* of 1. This essentially gave the model full access to all features at each split, allowing very deep trees. Because the original dataset is the largest and has the highest number of examples, it can tolerate this greater complexity without immediately succumbing to overfitting. In contrast, the undersampled datasets, which reduce the size of the majority class, performed best with a consistent *mtry* of 8 but varied *min.node.size* values (3, 4, or 8). Larger *min.node.size* values generally make trees shallower, serving as a safeguard against overfitting to the smaller, undersampled training set.

For SMOTE-based datasets, an *mtry* of 8 also emerged as optimal, but with moderate *min.node.size* values (2 or 3). Because SMOTE introduces synthetic instances to expand the minority class, a balance is needed between deep trees (lower *min.node.size*) and avoiding overfitting to artificially generated samples. While SMOTE helps to address class imbalance, the model still benefits from not considering all features at once (i.e., *mtry* < total number of predictors), likely because it reduces the risk of spurious splits.

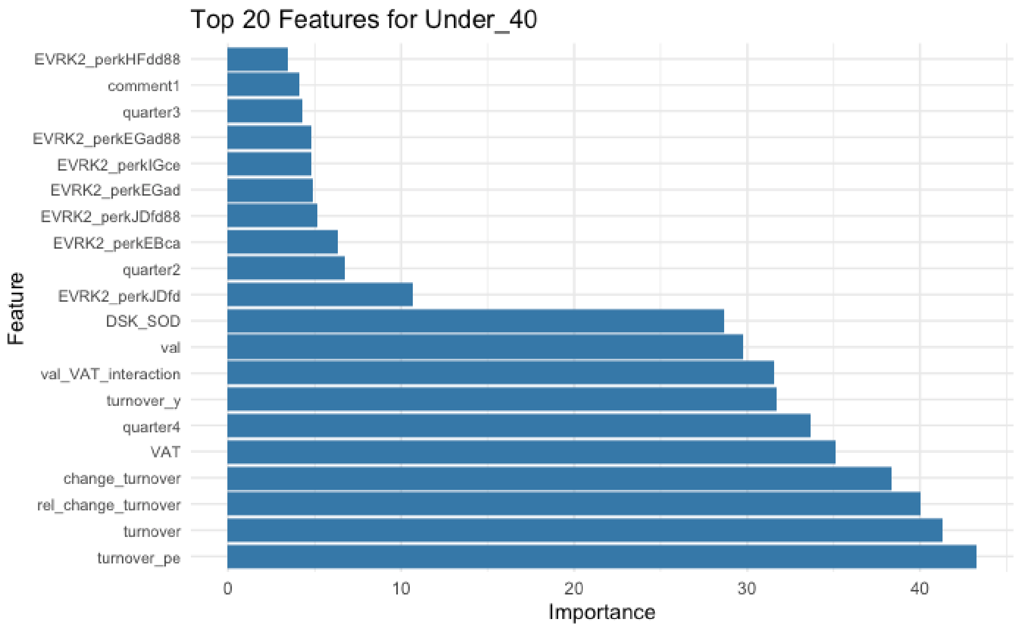


Figure 6 . *Top 20 Features for the Under_20 Model on the Test Dataset*

Image 5 illustrates the top 20 features contributing to the model trained on the *Under_40* dataset. These rankings provide insight into which variables most strongly influence the classifier's decisions once undersampling has balanced the classes. Turnover per employee, turnover and relative change of turnover rank as top three features. Enterprise sector classifiers (*EVRK_perk*) did not rank in top 10, which is expected as not all sectors groups had many observations.

Table 4 presents the performance metrics of models performance. The classifier trained on the original dataset achieved a high overall accuracy of 0.960 but exhibited a notably low sensitivity of 0.348, indicating poor performance in identifying the minority class (*Class1*, erroneous *turnover* values). In contrast, undersampling methods (*Under_60*, *Under_40*, *Under_20*) substantially improved sensitivity, with *Under_40* and *Under_20* both achieving the highest sensitivity score of 0.713. This suggests that reducing the majority class instances helped the model detect erroneous entries more effectively.

Meanwhile, oversampling approaches (*SMOTE_40*, *SMOTE_20*) did not yield similar improvements in sensitivity; oversampled models scored as low as 0.011. Although oversampling maintained high accuracy, its low sensitivity indicates that simply augmenting the minority class did not enhance error detection. Additionally, the *AUC* values for oversampled models were lower than those achieved by undersampling-based models, further highlighting the limited efficacy of oversampling in this setting (see Figure 6).

While oversampling techniques like SMOTE are recognized for their ability to address class imbalance by generating synthetic minority class instances [3], their application in this study yielded limited improvements in sensitivity and overall model performance. Several limitations inherent to SMOTE likely contributed to its underperformance in enhancing error detection.

SMOTE generates synthetic instances by interpolating between existing minority class samples. However, if the minority class is extremely sparse or lacks sufficient variability, the synthetic samples may not capture the true underlying distribution of the minority class [14, 27]. This can result in

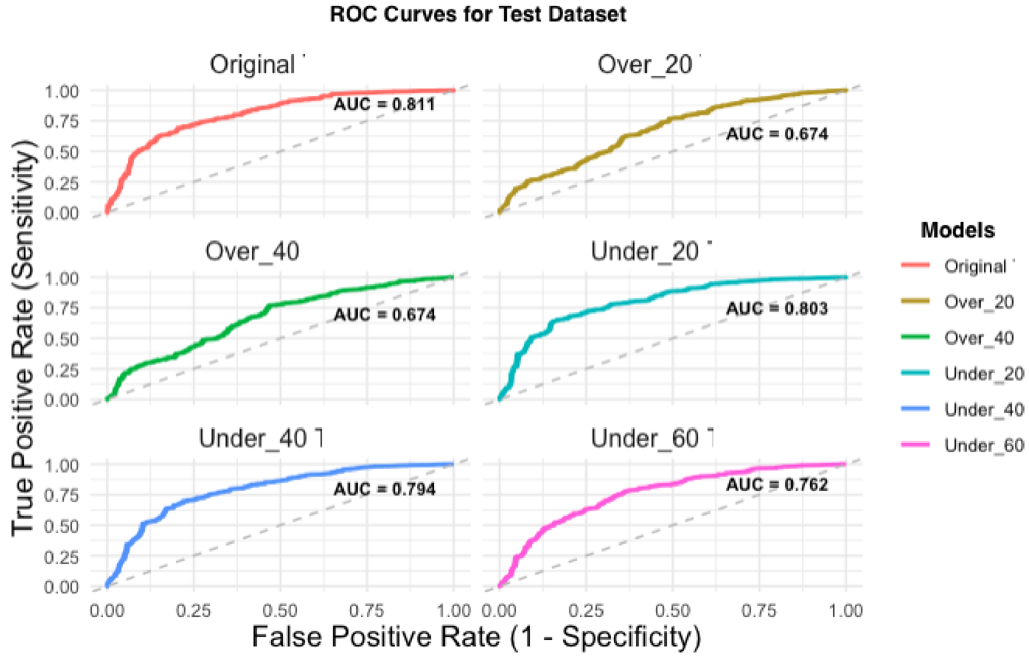


Figure 7 . *Models' ROC Cruves and AUC Values on Test Data*

synthetic instances that are not representative, thereby providing little to no benefit in improving model sensitivity. In scenarios where the feature space of the majority and minority classes overlaps significantly, SMOTE may inadvertently create synthetic samples that lie in regions where the classes are not well-separated. This can lead to confusion within the classifier, diminishing its ability to accurately distinguish between classes. Given that the original dataset exhibited strong class imbalance (1.40% minority class), SMOTE's effectiveness may be constrained. In cases of strong imbalance, oversampling alone may not suffice to significantly improve model sensitivity. In this study, the minimal gains in sensitivity and AUC observed in oversampled models indicate that SMOTE alone was insufficient to overcome the challenges posed by the class imbalance.

Models trained on undersampled datasets demonstrated superior performance, a finding that aligns with existing research. For instance, the best-performing RF classification model for statistical data performed by Bohnensteffen [2] also incorporated undersampling.

6.3 Regression Forest for Error Imputation

The second stage of this thesis involves imputing values that were classified as erroneous in the previous stage. To accomplish this, a RF regression model was developed using the subset of the training dataset flagged for erroneous turnover values (*Class1*). The primary objective was to predict corrected turnover values, represented by the *turnoverR* variable.

The training dataset consisted solely of observations edited in the SDA's data editing process, totaling 950 records. For testing, 3,264 instances labeled as erroneous by the best classification model from the previous stage (trained on the 20% undersampled dataset) were used.

Some predictors, notably *EVRK2_perk* and *status*, were excluded from the regression analysis. *EVRK2_perk* was removed due to its limited levels, which necessitated consolidating categories into an "Other" label. Similarly, *status* was excluded for the same reason, as it contributed little informative value to the regression process.

6.3.1 Model Evaluation

To validate the model's effectiveness, a custom summary function was developed to compute key performance metrics: Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R^2. Hyperparameter tuning was conducted via grid search, considering both *extratrees* and *variance* as potential split rules. A 10-fold cross-validation procedure was applied to refine and select the final model.

Initially, only erroneous turnover observations were used for training. However, the results revealed a systematic overestimation of turnover. To address this, non-erroneous values were introduced using different ratios, aiming to find the optimal balance of erroneous and correct entries. Three splits were tested:

- **0.5 split**: Half as many correct entries (*turnover_correct* $= 0.5 \times$ *turnover_error*).
- **1.0 split**: Equal numbers of correct and erroneous entries (*turnover_correct* $=$ *turnover_error*).
- **2.0 split**: Twice as many correct entries (*turnover_correct* $= 2 \times$ *turnover_error*).

As a baseline, a benchmark model used only the raw turnover value as a sole predictor, serving as a simple reference point. The performance metrics (RMSE, R^2, MAE) were then computed by comparing the predicted *turnoverR* with the true edited turnover. Among the evaluated configurations, the 2.0 ratio of correct to erroneous entries produced the most favorable results, improving the R^2 value from 0.674 (when only erroneous values were included) to 0.810 (see Table 5). Despite this substantial gain, the baseline model continued to deliver superior performance, implying that adjustments of training dataset were not enough an further enhancements are needed to improve the regression model's accuracy. For all models, the best performing split rule was `extratrees`.

Metric	Benchmark	Errors-Only Split	0.5 Split	1.0 Split	2.0 Split
mtry	NA	1	2	3	4
min.node.size	NA	1	1	1	1
RMSE	3,480,299	6,930,039	5,855,146	5,016,681	4,500,000
R^2	0.898	0.674	0.731	0.786	0.810
MAE	123,579	958,746	742,506	586,808	550,000

Table 5 . *Regression Forest Performance on Test Data for Different Training Splits*

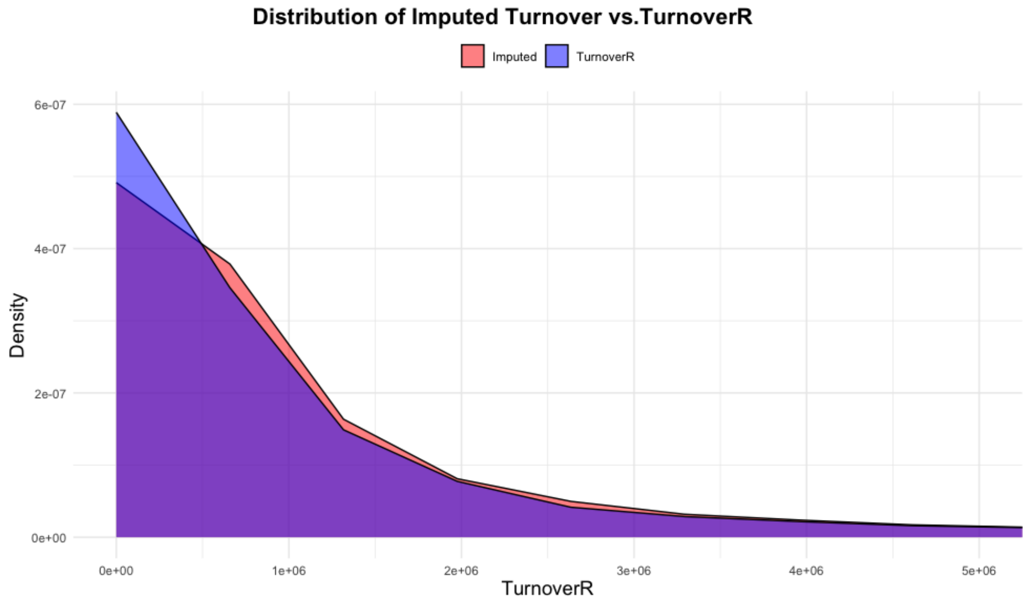
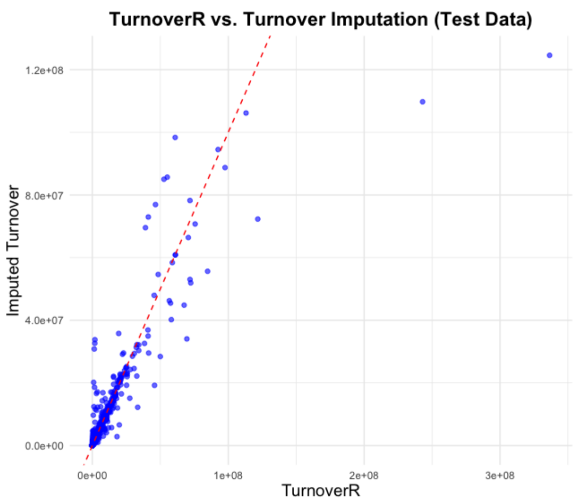
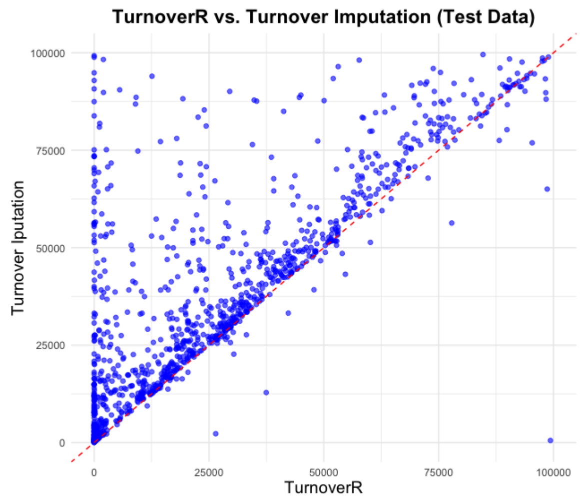


Figure 9 . *Density Plot of Imputed vs. Edited Turnover*



(a) *Scatter Plot of Imputed vs. Edited Turnover*



(b) *Scatter Plot of Imputed vs. Edited Turnover (Axes Limited to 100k)*

Figure 8 . *Comparison of Scatter Plots: Imputed vs. Edited Turnover*

Closer examination of the imputed turnover values compared to the target values reveals that the model unperformed to accurately predict cases where turnover was zero (see Figure 8(b)). One possible reason is that zero turnover events are inherently sparse and may differ substantially from non-zero data points in terms of underlying business or economic drivers. This scarcity of training samples with zero turnover can limit the model's ability to learn the distinct factors that characterize such instances.

In addition, extreme turnover values tended to be underestimated. Although the regression approach captured moderate turnover patterns, it appears less effective for outliers at the higher end of the distribution. Such understimations might stem from the relatively low frequency of extreme observations, which can lead the model to generalize poorly in regions where limited data exist. Figure 9 provides a more detailed view through a density plot, highlighting how the predicted turnover values deviate most noticeably at the zero.

Despite efforts to refine the training data composition by introducing non-erroneous observations at various ratios, the proposed Random Forest regression model consistently underperformed relative to the simple benchmark. While expanding the training set to include more correct entries significantly improved R^2 from 0.674 to 0.810, further enhancements remain necessary to close the gap with the benchmark.

Closer examination of prediction errors highlights two key challenges. First, the model struggled to handle zero-turnover cases, likely due to the scarcity of such observations, which impedes the learning of a distinct rule for zero. Second, extreme turnover values were underestimated. Addressing these challenges may require additional feature engineering.

7 Conclusion

Data editing is a fundamental process for NSIs, yet it demands substantial resources, particularly when additional follow-up with respondents is required. This thesis explored the application of RF techniques to automate two critical editing tasks: identifying erroneous turnover records and imputing accurate values for those flagged as errors.

From a classification perspective, the RF model demonstrated significant potential to improve or work as addition to currently used method at SDA, Lithuania. By incorporating undersampling techniques, the model achieved a balanced accuracy of 0.725, compared to the 0.636 balanced accuracy attained by the currently employed selective editing approach at the SDA. Additionally, the RF model exhibited higher sensitivity in detecting genuinely erroneous values (*true positives*) and reduced the incidence of *false positives*. This improvement is particularly valuable for NSIs, as more precise identification of erroneous records enables more efficient allocation of resources for manual error correction.

Conversely, the RF regression model designed to impute corrected turnover values encountered notable challenges, especially in cases where the true turnover was zero. Even after expanding the training dataset to include additional non-erroneous observations, the model continued to overestimate these zero turnover values. These findings align with those reported by Bohnensteffen [2], who also observed difficulties with RF regression in accurately predicting near-zero values. Furthermore, the limited auxiliary information—such as relying on only a single historical data point with substantial missingness—likely constrained the model's ability to generalize effectively. This suggests that integrating more comprehensive longitudinal and contextual data is necessary to enhance imputation accuracy.

Looking ahead, future research could build upon these insights by:

1. Incorporating a broader range of historical and contextual features to further improve both classification and regression performance.

2. Investigating advanced modeling techniques specifically designed to handle zero-inflated data, which could mitigate issues associated with predicting zero and near-zero turnover cases.

Overall, this thesis concludes that while Random Forests offer a promising solution for enhancing error detection in data editing processes, accurately imputing true values remains challenging without more extensive data and refined feature engineering. The complete automation of the statistical editing process continues to be a difficult goal, as also noted by De Waal, Quéré [12]. However, with the increasing focus on the application of machine learning within NSIs, ongoing improvements are being made. A hybrid or selective editing approach appears more feasible, wherein model-derived error estimates are used to prioritize observations for manual review. This ensures that human expertise is reserved for the most suspicious or influential cases, while relatively minor or unambiguous instances can be addressed algorithmically.

In summary, the integration of machine learning techniques like Random Forests into NSI data editing workflows has the potential to enhance both efficiency and accuracy. Continued research and the incorporation of richer datasets will be essential to overcoming the current limitations and achieving more robust automation in data editing processes.

8 References

References

[1] J. Bergstra, Y. Bengio. “Random Search for Hyper-Parameter Optimization.” In: *International Conference on Neural Information Processing Systems*. 2012, pages 281–289.

[2] S. Bohnensteffen. “Selective Data Editing of Continuous Variables with Random Forests in Official Statistics.” Supervisors: Dr. David Salgado Fernandez, Elena Rosa Perez. Master’s Thesis. Madrid, Spain: Complutense University Madrid, Faculty of Mathematical Science, 2020. URL: `https://www.ucm.es/`.

[3] K. W. Bowyer, N. V. Chawla, L. O. Hall, W. P. Kegelmeyer. “SMOTE: Synthetic Minority Oversampling Technique.” In: *CoRR* abs/1106.1813 (2011). URL: `http://arxiv.org/abs/1106.1813`.

[4] L. Breiman. *Classification and Regression Trees*. Chapman & Hall/CRC, 1984.

[5] L. Breiman. “Bagging Predictors.” In: *Machine Learning* 24.2 (1996), pages 123–140.

[6] L. Breiman. “Random Forests.” In: *Machine Learning* 45.1 (2001), pages 5–32.

[7] I. Burakauskaite. “Moving towards the standardized process of automatic statistical data editing using machine learning techniques.” In: *Expert Meeting on Statistical Data Editing, United Nations Economic Commission for Europe, Conference of European Statisticians*. UNECE. Vienna, Austria, 2024, pages 7–9.

[8] C.-J. Chen, A. Liaw, L. Breiman. “Using Random Forest to Learn Imbalanced Data.” In: *Proceedings of the Workshop on Learning from Imbalanced Datasets*. 2004.

[9] D. R. Cutler, T. C. Edwards, K. H. Beard, A. Cutler, K. T. Hess, J. Gibson, J. J. Lawler. “Random Forests for Classification in Ecology.” In: *Ecology* 88.11 (2007), pages 2783–2792.

[10] E. De Jonge, M. Van der Loo. *Manipulation of Linear Edits and Error Localization with the Editrules Package*. Discussion Paper 201120. The Hague: Statistics Netherlands, 2011.

[11] T. De Waal, J. Pannekoek, S. Scholtus. *Handbook of statistical data editing and imputation*. Volume 563. Hoboken, New Jersey: John Wiley & Sons, 2011.

[12] T. De Waal, R. Quéré. “A Fast and Simple Algorithm for Automatic Editing of Mixed Data.” In: *Journal of official statistics* 19 (2003), pages 383–402.

[13] B. Efron. “Bootstrap Methods: Another Look at the Jackknife.” In: *The Annals of Statistics* 7.1 (1979), pages 1–26.

[14] Y. Elor, H. Averbuch-Elor. “To SMOTE, or not to SMOTE?” In: *CoRR* abs/2201.08528 (2022). URL: `https://arxiv.org/abs/2201.08528`.

[15] P. Geurts, D. Ernst, L. Wehenkel. “Extremely Randomized Trees.” In: *Proceedings of the 22nd International Conference on Machine Learning*. ACM. 2006, pages 825–832.

[16] T. Hastie, R. Tibshirani, J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

[17] H. Ishwaran, U. B. Kogalur. “Random Survival Forests for R.” In: *R News* 8.3 (2008), pages 25–31.

[18] A. Liaw, M. Wiener. “Classification and Regression by randomForest.” In: *R News* 2.3 (2002), pages 18–22.

[19] D. B. Rubin. “Inference and Missing Data.” In: *Biometrika* 63 (1976), pages 581–590.

[20] S. Scholtus. *Automatic Editing with Soft Edits*. Discussion Paper 201130. The Hague: Statistics Netherlands, 2011.

[21] S. Scholtus. “Automatic Editing with Hard and Soft Edits.” In: *Survey Methodology* 39 (2013), pages 59–89.

[22] S. Scholtus. *New Results on Automatic Editing Using Hard and Soft Edit Rules*. Working Paper. Conference on Statistical Data Editing. 2015.

[23] S. Scholtus, S. Göksen. “Automatic Editing with Hard and Soft Edits–Some First Experiences.” In: *Work Session on Statistical Data Editing, Conference of European Statisticians, United Nations Economic Commission for Europe*. Oslo, Norway, 2012, pages 24–26.

[24] E. Scornet, G. Biau, J.-P. Vert. “Consistency of Random Forests.” In: *The Annals of Statistics* 43.2 (2015), pages 1028–1056.

[25] UNECE Statistics Division. *GSBPM v5.1: Generic Statistical Business Process Model (GSBPM)*. 5.1. Updated: November 17, 2024. UNECE Statistics Division. 2019. URL: `https://example.com/GSBPMv5_1.pdf`.

[26] J. Uogelė. “Missing Data Imputation Methods for Monthly Statistical Survey on Trade and Catering Enterprises.” Unpublished Master’s Thesis. Vilnius University, 2023.

[27] G. M. Weiss. “Foundations of Imbalanced Learning.” In: *Imbalanced Learning: Foundations, Algorithms, and Applications*. Edited by H. He, Y. Ma. Wiley, 2013.

9 Appendix

A: Software and Tools Utilized

In the development and execution of this thesis, several software applications and artificial intelligence tools were utilized.

The theoretical part of this thesis was conducted using R, a statistical programming language.

As English is not the author's native language, it was essential to ensure that the thesis met the required academic standards for language and presentation. To assist in refining the written material, ChatGPT-4, an artificial intelligence language model developed by OpenAI, was employed. ChatGPT-4 was used to ensure that the writing was suitable for academic purposes, as well as to troubleshoot and resolve issues encountered during the implementation of theoretical aspects in R and the creation of LaTeX tables.

The following prompt was used:

```
"You are a data scientist writing an article for an academic journal
on statistical data editing. Your article will be read by experts in
the field, and no mistakes or inconsistencies should be present in
your text. You write in North American English, clearly and with no
grammatical mistakes. Edit the following for clarity:.."
```

The author acknowledges that AI can produce errors and assumes full responsibility for all content presented in this thesis.

B: R Code for Classification and Regression Tasks

```
# ----------------------------
# Data Processing
# ----------------------------

data_2017 <- import_list("Duomenys/Duomenys2017.xlsx")
data_2018 <- import_list("Duomenys/Duomenys2018.xlsx")
data_2019 <- import_list("Duomenys/Duomenys2019.xlsx")
data_2020 <- import_list("Duomenys/Duomenys2020.xlsx")
data_2021 <- import_list("Duomenys/Duomenys2021.xlsx")
data_2022 <- import_list("Duomenys/Duomenys2022.xlsx")
data_2023 <- import_list("Duomenys/Duomenys2023.xlsx")


# Removing F1_600
process_data <- function(data_year) {
  data_year$pradiniai <- data_year$pradiniai %>%
    select(-starts_with("F1_600")) %>%
    arrange(ID)

  data_year$isskirtys <- data_year$isskirtys %>%
    select(-starts_with("F1_600")) %>%
    arrange(ID)

  data_year$redaguoti <- data_year$redaguoti %>%
    select(-starts_with("F1_600")) %>%
    arrange(ID)

  data_year$pradiniai <- data_year$pradiniai %>%
    group_by(ID) %>%
    summarise(across(everything(), ~ .x[1]), .groups = 'drop')

  return(data_year)
}

data_2017 <- process_data(data_2017)
data_2018 <- process_data(data_2018)
data_2019 <- process_data(data_2019)
data_2020 <- process_data(data_2020)
data_2021 <- process_data(data_2021)
data_2022 <- process_data(data_2022)
```

```r
data_2023 <- process_data(data_2023)


# Removing some other variables that are not consistent
data_2017$pradiniai <- data_2017$pradiniai %>%
  select(-c("EVRK2_perk_20172","EVRK2_perk_20173","EVRK2_perk_20174","PASTABOS2_20171","PASTABOS2_20172","PASTABOS2_20173","PASTABOS2_20174"))
data_2017$redaguoti <- data_2017$redaguoti %>%
  select(-c("EVRK2_perk_20172","EVRK2_perk_20173","EVRK2_perk_20174","PASTABOS2_20171","PASTABOS2_20172","PASTABOS2_20173","PASTABOS2_20174"))
data_2017$isskirtys <- data_2017$isskirtys %>%
  select(-c("sel_F01_20171","sel_F01_20172","sel_F01_20173","sel_F01_20174"))

data_2018$pradiniai  <- data_2018$pradiniai  %>%
  select(-c("EVRK2_perk_20182","EVRK2_perk_20183","EVRK2_perk_20184","PASTABOS2_20181","PASTABOS2_20182","PASTABOS2_20183","PASTABOS2_20184"))
data_2018$redaguoti <- data_2018$redaguoti %>%
  select(-c("EVRK2_perk_20182","EVRK2_perk_20183","EVRK2_perk_20184","PASTABOS2_20181","PASTABOS2_20182","PASTABOS2_20183","PASTABOS2_20184"))
data_2018$isskirtys <- data_2018$isskirtys %>%
  select(-c("sel_F01_20181","sel_F01_20182","sel_F01_20183","sel_F01_20184"))

data_2019$pradiniai <- data_2019$pradiniai %>%
  select(-c("EVRK2_perk_20192","EVRK2_perk_20193","EVRK2_perk_20194","PASTABOS2_20191","PASTABOS2_20192","PASTABOS2_20193","PASTABOS2_20194"))
data_2019$redaguoti <- data_2019$redaguoti %>%
  select(-c("EVRK2_perk_20192","EVRK2_perk_20193","EVRK2_perk_20194","PASTABOS2_20191","PASTABOS2_20192","PASTABOS2_20193","PASTABOS2_20194"))
data_2019$isskirtys <- data_2019$isskirtys %>%
  select(-c("sel_F01_20191","sel_F01_20192","sel_F01_20193","sel_F01_20194"))

data_2020$pradiniai <- data_2020$pradiniai %>%
```

```r
  select(-c("EVRK2_perk_20202","EVRK2_perk_20203","EVRK2_perk_20204","PASTABOS2_20201","PASTABOS2_20202","PASTABOS2_20203","PASTABOS2_20204"))
data_2020$redaguoti <- data_2020$redaguoti %>%
  select(-c("EVRK2_perk_20202","EVRK2_perk_20203","EVRK2_perk_20204","PASTABOS2_20201","PASTABOS2_20202","PASTABOS2_20203","PASTABOS2_20204"))
data_2020$isskirtys <- data_2020$isskirtys %>%
  select(-c("sel_F01_20201","sel_F01_20202","sel_F01_20203","sel_F01_20204"))

data_2021$pradiniai <- data_2021$pradiniai %>%
  select(-c("DPS_PASTABOS_20211","DPS_PASTABOS_20212","DPS_PASTABOS_20213","DPS_PASTABOS_20214"))
data_2021$redaguoti <- data_2021$redaguoti %>%
  select(-c("DPS_PASTABOS_20211","DPS_PASTABOS_20212","DPS_PASTABOS_20213","DPS_PASTABOS_20214"))

data_2022$pradiniai <- data_2022$pradiniai %>%
  select(-c("DPS_PASTABOS_20221","DPS_PASTABOS_20222","DPS_PASTABOS_20223","DPS_PASTABOS_20224"))
data_2022$redaguoti <- data_2022$redaguoti %>%
  select(-c("DPS_PASTABOS_20221","DPS_PASTABOS_20222","DPS_PASTABOS_20223","DPS_PASTABOS_20224"))

# removing unmatched IDs
ids_2018 <- data_2018$pradiniai[!data_2018$pradiniai$ID %in% data_2018$redaguoti$ID, ]
data_2018$pradiniai <- data_2018$pradiniai %>%
  filter(!ID %in% ids_2018$ID)
data_2018$isskirtys <- data_2018$isskirtys %>%
  filter(!ID %in% ids_2018$ID)

ids_2019 <- data_2019$pradiniai[!data_2019$pradiniai$ID %in% data_2019$redaguoti$ID, ]
data_2019$pradiniai <- data_2019$pradiniai %>%
  filter(!ID %in% ids_2019$ID)
data_2019$isskirtys <- data_2019$isskirtys %>%
  filter(!ID %in% ids_2019$ID)

# Define the standard column names
standard_colnames <- c("ID", "EVRK2_perk", "BUKLE_1", "PASTABA_1",
                       "viso_1", "viso_y1","val_1", "pvm_m1",
```

```
                       "pvm_m2", "pvm_m3", "pvm_1", "DSK_SOD_1",
                       "BUKLE_2", "PASTABA_2","viso_2", "viso_y2",
                       "val_2", "pvm_m4", "pvm_m5", "pvm_m6",
                       "pvm_2", "DSK_SOD_2", "BUKLE_3", "PASTABA_3",
                       "viso_3", "viso_y3", "val_3", "pvm_m7",
                       "pvm_m8", "pvm_m9", "pvm_3", "DSK_SOD_3",
                       "BUKLE_4", "PASTABA_4","viso_4", "viso_y4",
                       "val_4", "pvm_m10", "pvm_m11", "pvm_m12",
                       "pvm_4", "DSK_SOD_4", "imtyje_1", "atsake_1",
                       "svoriai_1", "svoriai_kalibr_1", "imtyje_2", "
                          atsake_2",
                       "svoriai_2", "svoriai_kalibr_2", "imtyje_3", "
                          atsake_3",
                       "svoriai_3","svoriai_kalibr_3", "imtyje_4", "
                          atsake_4",
                       "svoriai_4", "svoriai_kalibr_4", "tikrinti_1","
                          tikrinti_2",
                       "tikrinti_3", "tikrinti_4")

colnames(data_2017$pradiniai) <- standard_colnames
colnames(data_2018$pradiniai) <- standard_colnames
colnames(data_2019$pradiniai) <- standard_colnames
colnames(data_2020$pradiniai) <- standard_colnames
colnames(data_2021$pradiniai) <- standard_colnames
colnames(data_2022$pradiniai) <- standard_colnames

# tikrinti_1 is missing in 2023
standard_colnames <- c("ID", "EVRK2_perk", "BUKLE_1", "PASTABA_1",
                       "viso_1", "viso_y1","val_1", "pvm_m1",
                       "pvm_m2", "pvm_m3", "pvm_1", "DSK_SOD_1",
                       "BUKLE_2", "PASTABA_2","viso_2", "viso_y2",
                       "val_2", "pvm_m4", "pvm_m5", "pvm_m6",
                       "pvm_2", "DSK_SOD_2", "BUKLE_3", "PASTABA_3",
                       "viso_3", "viso_y3", "val_3", "pvm_m7",
                       "pvm_m8", "pvm_m9", "pvm_3", "DSK_SOD_3",
                       "BUKLE_4", "PASTABA_4","viso_4", "viso_y4",
                       "val_4", "pvm_m10", "pvm_m11", "pvm_m12",
                       "pvm_4", "DSK_SOD_4", "imtyje_1", "atsake_1",
                       "svoriai_1", "svoriai_kalibr_1", "imtyje_2", "
                          atsake_2",
                       "svoriai_2", "svoriai_kalibr_2", "imtyje_3", "
                          atsake_3",
```

```
                        "svoriai_3","svoriai_kalibr_3", "imtyje_4", "
                           atsake_4",
                        "svoriai_4", "svoriai_kalibr_4", "tikrinti_2",
                        "tikrinti_3", "tikrinti_4")


colnames(data_2023$pradiniai) <- standard_colnames

standard_colnames <- c("ID", "EVRK2_perk", "BUKLE_1", "PASTABA_1",
                        "viso_1", "viso_y1","val_1", "pvm_m1",
                        "pvm_m2", "pvm_m3", "pvm_1", "DSK_SOD_1",
                        "BUKLE_2", "PASTABA_2","viso_2", "viso_y2",
                        "val_2", "pvm_m4", "pvm_m5", "pvm_m6",
                        "pvm_2", "DSK_SOD_2", "BUKLE_3", "PASTABA_3",
                        "viso_3", "viso_y3", "val_3", "pvm_m7",
                        "pvm_m8", "pvm_m9", "pvm_3", "DSK_SOD_3",
                        "BUKLE_4", "PASTABA_4","viso_4", "viso_y4",
                        "val_4", "pvm_m10", "pvm_m11", "pvm_m12",
                        "pvm_4", "DSK_SOD_4", "imtyje_1", "atsake_1",
                        "svoriai_1", "svoriai_kalibr_1", "imtyje_2", "
                           atsake_2",
                        "svoriai_2", "svoriai_kalibr_2", "imtyje_3", "
                           atsake_3",
                        "svoriai_3","svoriai_kalibr_3", "imtyje_4", "
                           atsake_4",
                        "svoriai_4", "svoriai_kalibr_4")

colnames(data_2017$redaguoti) <- standard_colnames
colnames(data_2018$redaguoti) <- standard_colnames
colnames(data_2019$redaguoti) <- standard_colnames
colnames(data_2020$redaguoti) <- standard_colnames
colnames(data_2021$redaguoti) <- standard_colnames
colnames(data_2022$redaguoti) <- standard_colnames
colnames(data_2023$redaguoti) <- standard_colnames

# Define the standard column names
standard_colnames <- c(
  "ID", "sel_pvm_1",  "HB_prm_1", "HB_prk_1", "kvartiliai_1", "tikrinti_1
     ",
  "sel_pvm_2",  "HB_prm_2", "HB_prk_2", "kvartiliai_2", "tikrinti_2",
  "sel_pvm_3",  "HB_prm_3", "HB_prk_3", "kvartiliai_3", "tikrinti_3",
  "sel_pvm_4",  "HB_prm_4", "HB_prk_4", "kvartiliai_4", "tikrinti_4"
)
```

```r

colnames(data_2017$isskirtys) <- standard_colnames
colnames(data_2018$isskirtys) <- standard_colnames
colnames(data_2019$isskirtys) <- standard_colnames
colnames(data_2020$isskirtys) <- standard_colnames
colnames(data_2021$isskirtys) <- standard_colnames
colnames(data_2022$isskirtys) <- standard_colnames

# Q1 is missing
standard_colnames <- c(
  "ID",
  "sel_pvm_2",  "HB_prm_2", "HB_prk_2", "kvartiliai_2", "tikrinti_2",
  "sel_pvm_3",  "HB_prm_3", "HB_prk_3", "kvartiliai_3", "tikrinti_3",
  "sel_pvm_4",  "HB_prm_4", "HB_prk_4", "kvartiliai_4", "tikrinti_4"
)

colnames(data_2023$isskirtys) <- standard_colnames


# combining data
years <- 2018:2023

numeric_columns <- c(
  paste0("viso_", 1:4, "_pradiniai"),
  paste0("viso_", 1:4, "_redaguoti"),
  paste0("viso_y", 1:4, "_pradiniai"),
  paste0("viso_y", 1:4, "_redaguoti"),
  paste0("val_", 1:4, "_pradiniai"),
  paste0("val_", 1:4, "_redaguoti"),
  paste0("pvm_", 1:4, "_pradiniai"),
  paste0("pvm_", 1:4, "_redaguoti"),
  paste0("DSK_SOD_", 1:4, "_pradiniai"),
  paste0("DSK_SOD_", 1:4, "_redaguoti"),
  paste0("diff_", 1:4)
)

character_columns <- c(
  "ID",
  "EVRK2_perk_pradiniai",
  "EVRK2_perk_redaguoti",
  paste0("BUKLE_", 1:4, "_pradiniai"),
  paste0("BUKLE_", 1:4, "_redaguoti"),
  paste0("edited_", 1:4),
```

```r
  paste0("sel_pvm_", 1:4, "_isskirtys"),
  paste0("HB_prm_", 1:4, "_isskirtys"),
  paste0("HB_prk_", 1:4, "_isskirtys"),
  paste0("kvartiliai_", 1:4, "_isskirtys"),
  paste0("tikrinti_", 1:4, "_isskirtys")
)

# Data is cleaned now, merging section
# Loop over the years 2017 to 2023
for (year in 2017:2023) {

  data_year <- get(paste0("data_", year))

  pradiniai_data <- data_year$pradiniai
  redaguoti_data <- data_year$redaguoti
  isskirtys_data <- data_year$isskirtys

  pradiniai_data$ID <- as.character(pradiniai_data$ID)
  redaguoti_data$ID <- as.character(redaguoti_data$ID)
  isskirtys_data$ID <- as.character(isskirtys_data$ID)

  pradiniai_data <- pradiniai_data %>%
    rename_with(~ paste0(., "_pradiniai"), -ID)

  redaguoti_data <- redaguoti_data %>%
    rename_with(~ paste0(., "_redaguoti"), -ID)

  merged_data <- full_join(pradiniai_data, redaguoti_data, by = "ID")

  merged_data <- left_join(merged_data, isskirtys_data, by = "ID")

  assign(paste0("merged_data_", year), merged_data)
}


# Adressing NAs
cleaned_data_list <- list()

for (year in years) {
  df_name <- paste0("merged_data_", year)
  merged_data <- get(df_name)

  merged_data$year <- year
```

```r

  merged_data[merged_data == ""] <- NA
  merged_data[merged_data == "NA"] <- NA
  merged_data[merged_data == "NaN"] <- NA
  merged_data[merged_data == "<NA>"] <- NA

  for (col in numeric_columns) {
    if (col %in% names(merged_data)) {
      merged_data[[col]] <- gsub(",", "", merged_data[[col]])
      merged_data[[col]] <- gsub("[^0-9\\.\\-]", "", merged_data[[col]])
      merged_data[[col]] <- as.numeric(merged_data[[col]])
    }
  }

  for (col in character_columns) {
    if (col %in% names(merged_data)) {
      merged_data[[col]] <- as.character(merged_data[[col]])
    }
  }

  cleaned_data_list[[as.character(year)]] <- merged_data

  assign(df_name, merged_data)
}

cleaned_data_list <- convert_columns_to_logical(cleaned_data_list, columns_to_convert)
combined_data_3 <- bind_rows(cleaned_data_list)

# Selecting only columns that will be used for RF
selected_columns <- c("ID", "year", "EVRK2_perk_pradiniai",
                      "BUKLE_1_pradiniai", "PASTABA_1_pradiniai", "viso_1_pradiniai", "viso_y1_redaguoti", "val_1_redaguoti", "pvm_m1_pradiniai", "pvm_m2_pradiniai", "pvm_m3_pradiniai","pvm_1_redaguoti", "DSK_SOD_1_pradiniai",  "edited_1", "diff_y1",
                      "BUKLE_2_pradiniai", "PASTABA_2_pradiniai","viso_2_pradiniai", "viso_y2_redaguoti", "val_2_redaguoti", "pvm_2_redaguoti","pvm_m4_pradiniai", "pvm_m5_pradiniai","pvm_m6_pradiniai","DSK_SOD_2_pradiniai", "edited_2", "diff_y2",
                      "BUKLE_3_pradiniai", "PASTABA_3_pradiniai","viso_3_pradiniai", "viso_y3_redaguoti", "val_3_
```

```
                              redaguoti", "pvm_3_redaguoti", "pvm_m7_pradiniai
                              ","pvm_m8_pradiniai","pvm_m9_pradiniai","DSK_SOD
                              _3_pradiniai", "edited_3", "diff_y3",
                           "BUKLE_4_pradiniai", "PASTABA_4_pradiniai","viso_4_
                              pradiniai", "viso_y4_redaguoti", "val_4_
                              redaguoti", "pvm_4_redaguoti","pvm_m10_pradiniai
                              ", "pvm_m11_pradiniai","pvm_m12_pradiniai","DSK_
                              SOD_4_pradiniai", "edited_4","diff_y4", "atsake_
                              1_pradiniai", "atsake_2_pradiniai","atsake_3_
                              pradiniai","atsake_4_pradiniai", "viso_1_
                              redaguoti","viso_2_redaguoti","viso_3_redaguoti"
                              ,"viso_4_redaguoti", "diff_abs_1", "diff_abs_2",
                               "diff_abs_3", "diff_abs_4", "diff_rel_y1", "
                              diff_rel_y2", "diff_rel_y3", "diff_rel_y4", "
                              svoriai_1_pradiniai", "svoriai_kalibr_1_
                              pradiniai", "svoriai_2_pradiniai", "svoriai_
                              kalibr_2_pradiniai", "svoriai_3_pradiniai", "
                              svoriai_kalibr_3_pradiniai", "svoriai_4_
                              pradiniai", "svoriai_kalibr_4_pradiniai", "
                              tikrinti_1_pradiniai","tikrinti_2_pradiniai","
                              tikrinti_3_pradiniai","tikrinti_4_pradiniai", "
                              imtyje_1_pradiniai", "imtyje_2_pradiniai", "
                              imtyje_3_pradiniai", "imtyje_4_pradiniai")

pradiniai_df <- combined_data_3[, c(selected_columns)]

column_names <-  c("ID", "year", "EVRK2_perk_pradiniai",
                        "BUKLE_1_pradiniai", "PASTABA_1_pradiniai", "viso_1_
                           pradiniai", "viso_y1_pradiniai", "val_1_pradiniai",
                            "pvm1m_1_pradiniai", "pvm2m_1_pradiniai", "pvm3m_1
                           _pradiniai","pvm_1_pradiniai", "DSK_SOD_1_pradiniai
                           ",  "edited_1", "diff_y1",
                        "BUKLE_2_pradiniai", "PASTABA_2_pradiniai","viso_2_
                           pradiniai", "viso_y2_pradiniai", "val_2_pradiniai",
                            "pvm_2_pradiniai","pvm1m_2_pradiniai", "pvm2m_2_
                           pradiniai","pvm3m_2_pradiniai","DSK_SOD_2_pradiniai
                           ", "edited_2", "diff_y2",
                        "BUKLE_3_pradiniai", "PASTABA_3_pradiniai","viso_3_
                           pradiniai", "viso_y3_pradiniai", "val_3_pradiniai",
                            "pvm_3_pradiniai", "pvm1m_3_pradiniai","pvm2m_3_
                           pradiniai","pvm3m_3_pradiniai","DSK_SOD_3_pradiniai
                           ", "edited_3", "diff_y3",
```

```r
                     "BUKLE_4_pradiniai", "PASTABA_4_pradiniai","viso_4_pradiniai", "viso_y4_pradiniai", "val_4_pradiniai", "pvm_4_pradiniai","pvm1m_4_pradiniai", "pvm2m_4_pradiniai","pvm3m_4_pradiniai","DSK_SOD_4_pradiniai", "edited_4","diff_y4", "atsake_1_pradiniai", "atsake_2_pradiniai","atsake_3_pradiniai","atsake_4_pradiniai", "visoR_1_pradiniai","visoR_2_pradiniai","visoR_3_pradiniai","visoR_4_pradiniai", "diff_abs_1", "diff_abs_2", "diff_abs_3", "diff_abs_4","diff_rel_y1", "diff_rel_y2", "diff_rel_y3", "diff_rel_y4", "svoriai_1_pradiniai", "svoriai_kalibr_1_pradiniai", "svoriai_2_pradiniai", "svoriai_kalibr_2_pradiniai", "svoriai_3_pradiniai", "svoriai_kalibr_3_pradiniai", "svoriai_4_pradiniai", "svoriai_kalibr_4_pradiniai","tikrinti_1_pradiniai", "tikrinti_2_pradiniai","tikrinti_3_pradiniai","tikrinti_4_pradiniai", "imtyje_1_pradiniai", "imtyje_2_pradiniai", "imtyje_3_pradiniai", "imtyje_4_pradiniai")

names(pradiniai_df) <- column_names

pradiniai_df <- pradiniai_df %>%
  rename(EVRK2_perk_1_pradiniai = EVRK2_perk_pradiniai) %>%
  mutate(EVRK2_perk_2_pradiniai = EVRK2_perk_1_pradiniai,
         EVRK2_perk_3_pradiniai = EVRK2_perk_1_pradiniai,
         EVRK2_perk_4_pradiniai = EVRK2_perk_1_pradiniai)

pradiniai_df <- pradiniai_df %>%
  rename(year_1 = year) %>%
  mutate(year_2 = year_1,
         year_3 = year_1,
         year_4 = year_1)

colnames(pradiniai_df) <- gsub("_pradiniai$", "", colnames(pradiniai_df))

colnames(pradiniai_df) <- sub("^(.*[a-zA-Z])([1-4])$", "\\1_\\2", colnames(pradiniai_df))
pradiniai_df$ID.1 <- NULL

reshaped_pradiniai <- pradiniai_df %>%
  pivot_longer(
    cols = -c(ID),
```

```r
    names_to = c(".value", "quarter"),
    names_pattern = "(.*)_(\\d+)$"
  ) %>%
  mutate(quarter = as.integer(quarter))

# ----------------------------
# Classification Forrest
# ----------------------------

data <- reshaped_pradiniai %>%
  select(-ID, -year,  -svoriai_kalibr)
#data <- data_no_missing
data <- data %>%
  mutate(
    turnover_pe = ifelse(DSK_SOD == 0 | is.na(DSK_SOD), 0, viso / DSK_SOD
       ),
    rel_change_turnover = ifelse(viso == 0 | is.na(viso_y), 0, (viso_y -
       viso) / viso),
    val_VAT_interaction = val * pvm,
    change_turnover = viso - viso_y
  ) %>%
  rename(
    turnover = viso,
    turnover_y = viso_y,
    VAT1m = pvm1m,
    VAT2m = pvm2m,
    VAT3m = pvm3m,
    VAT = pvm,
    turnoverR = visoR
  )

  data <- data %>%
  mutate(
    BUKLE = as.factor(BUKLE),
    quarter = as.factor(quarter),
    edited = as.factor(edited),
    PASTABA = as.factor(PASTABA),
    atsake = as.factor(atsake)
  )

  EVRK2_perk_counts <- edited_final_data %>%
  group_by(EVRK2_perk) %>%
  summarise(count = n()) %>%
```

```
  arrange(desc(count))

# Replace rare EVRK2_perk levels with 'Other'
rare_threshold <- 6
evrk2_counts <- data %>%
  group_by(EVRK2_perk) %>%
  tally()
rare_levels <- evrk2_counts %>%
  filter(n <= rare_threshold) %>%
  pull(EVRK2_perk)
data <- data %>%
  mutate(
    EVRK2_perk = as.character(EVRK2_perk),
    EVRK2_perk = ifelse(EVRK2_perk %in% rare_levels, "Other", EVRK2_perk),
    EVRK2_perk = as.factor(EVRK2_perk)
  )
print(table(data$EVRK2_perk))

# Splitting data
set.seed(666)
split <- initial_split(data, prop = 0.85, strata = edited)
train_data <- training(split)
test_data  <- testing(split)
cat("Training Set Class Distribution:\n")
print(prop.table(table(train_data$edited)))
cat("\nTesting Set Class Distribution:\n")
print(prop.table(table(test_data$edited)))

# Ensuring levels in both train_data and test_data
train_levels <- levels(train_data2[[EVRK2_perk]])
test_levels  <- levels(test_final2[[EVRK2_perkr]])
test_not_in_train <- setdiff(test_levels, train_levels)
if(length(test_not_in_train) > 0){
  cat("\nLevels in test_data$EVRK2_perk not present in train_data$EVRK2_perk:\n")
  print(test_not_in_train)
} else {
  cat("\nAll EVRK2_perk levels in test_data are present in train_data.\n")


# CREATING SMOTE and UNDERSAMPLING TRAIN_DATA
```

```r
create_smote_data <- function(train_data, minority_perc) {
  N_total <- nrow(train_data)
  smote_data <- ROSE(edited ~ ., data = train_data, N = N_total, p =
     minority_perc, seed = 1)$data
  return(smote_data)
}
train_data_smote2 <- create_smote_data(train_data, 0.40) # 25% minority
train_data_smote3 <- create_smote_data(train_data, 0.20) # 15% minority

# Undersampling datasets
create_undersample_data <- function(train_data, minority_perc) {
  minority_class <- train_data[train_data$edited == "Class1", ]
  majority_class <- train_data[train_data$edited == "Class0", ]
  N1 <- nrow(minority_class)
  N0_desired <- round(N1 * (1 - minority_perc) / minority_perc)
  set.seed(1)
  majority_class_under <- majority_class[sample(nrow(majority_class), N0_
     desired), ]
  under_data <- rbind(minority_class, majority_class_under)
  return(under_data)
}
train_data_under1 <- create_undersample_data(train_data, 0.60) # 25%
   minority
train_data_under2 <- create_undersample_data(train_data, 0.40) # 15%
   minority
train_data_under3 <- create_undersample_data(train_data, 0.20) # 15%
   minority

datasets <- list(
  Original = train_data,
  Under_60 = train_data_under1,
  Under_40 = train_data_under2,
  Under_20 = train_data_under3,
  Over_40 = train_data_smote2,
  Over_20 = train_data_smote3
)
table(train_data_smote1$edited)
table(train_data_smote2$edited)
table(train_data_smote3$edited)
table(train_data_under1$edited)
table(train_data_under2$edited)
table(train_data$edited)
```

```
# -----------------------------
# MODEL TRAINING
# -----------------------------

datasets <- list(
  Original = train_data,
  Under_60 = train_data_under1,
  Under_40 = train_data_under2,
  Under_20 = train_data_under3,
  Over_40 = train_data_smote2,
  Over_20 = train_data_smote3
)

# Initialize lists to store results
models_list <- list()
best_tunes <- list()
confusion_matrices_test <- list()
roc_test_list <- list()
auc_test_list <- list()
test_predictions <- list()

# Loop over each dataset
for (dataset_name in names(datasets)) {

  cat("Processing dataset:", dataset_name, "\n")

  tryCatch({
      train_data <- datasets[[dataset_name]] %>%
      select(
        edited, quarter, EVRK2_perk, BUKLE, PASTABA,
             viso, viso_y, val,
             pvm, DSK_SOD, change_viso, rel_change_viso, val_pvm_interaction, viso_pe
      ) %>%
      na.omit()

    train_data$edited <- factor(train_data$edited, levels = c("Class1", "Class0"))
    test_data$edited <- factor(test_data$edited, levels = c("Class1", "Class0"))

    # Compute class weights
    class_counts <- table(train_data$edited)
```

```r
  class_weights <- 1 / class_counts
  obs_weights <- class_weights[as.character(train_data$edited)]

  if (any(is.na(obs_weights))) {
    stop(paste("Missing weights in dataset:", dataset_name))
  }

  custom_summary <- function(data, lev = NULL, model = NULL) {
    sensitivity_val <- caret::sensitivity(data$obs, data$pred, positive = "Class1")
    specificity_val <- caret::specificity(data$obs, data$pred, positive = "Class1")
    roc_obj <- pROC::roc(response = data$obs, predictor = data$Class1, levels = c("Class1", "Class0"))
    roc_auc <- pROC::auc(roc_obj)
    c(Sens = sensitivity_val, Spec = specificity_val, ROC = roc_auc)
  }


  formula <- edited ~  quarter + EVRK2_perk + BUKLE + PASTABA +
           viso + viso_y + val +
           pvm + DSK_SOD + change_viso + rel_change_viso + val_pvm_interaction + viso_pe

  predictor_names <- all.vars(formula)[-1]
  num_predictors <- length(predictor_names)
  mtry_max <- num_predictors
  grid <- expand.grid(
    mtry = 1:13,
    splitrule = "gini",
    min.node.size = seq(1, 10, by = 1)
  )

  control <- caret::trainControl(
    method = "cv",
    number = 5,
    verboseIter = TRUE,
    classProbs = TRUE,
    summaryFunction = custom_summary,
    savePredictions = TRUE,
    allowParallel = FALSE
  )

```

```r
    set.seed(123)
    model <- caret::train(
      formula,
      data = train_data,
      method = "ranger",
      tuneGrid = grid,
      trControl = control,
      metric = "Sens",
      importance = 'impurity',
      num.trees = 750,
      weights = obs_weights
    )

    models_list[[dataset_name]] <- model
    best_tunes[[dataset_name]] <- model$bestTune

    test_probs <- predict(model, newdata = test_data, type = "prob")
    test_preds <- predict(model, newdata = test_data)

    test_predictions[[dataset_name]] <- test_data %>%
      mutate(
        Predicted_Class = test_preds,
        Probability_Class1 = test_probs$Class1,
        visoR = test_data$visoR
      )

    roc_test <- pROC::roc(response = test_data$edited, predictor = test_probs$Class1, levels = c("Class1", "Class0"))
    roc_test_list[[dataset_name]] <- roc_test
    auc_test_list[[dataset_name]] <- pROC::auc(roc_test)

    confusion_matrices_test[[dataset_name]] <- caret::confusionMatrix(
      test_preds,
      test_data$edited,
      positive = "Class1"
    )

    cat("Completed dataset:", dataset_name, "\n\n")

  }, error = function(e) {
    cat("Error processing dataset:", dataset_name, "\n")
    cat("Error message:", e$message, "\n\n")

```

```r
    models_list[[dataset_name]] <- NULL
    best_tunes[[dataset_name]] <- NA
    confusion_matrices_test[[dataset_name]] <- NA
    roc_test_list[[dataset_name]] <- NA
    auc_test_list[[dataset_name]] <- NA
    test_predictions[[dataset_name]] <- NA
  })
}

# -----------------------------
# MODEL RESULTS
# -----------------------------
for (dataset_name in names(best_tunes)) {
  cat("Best hyperparameters for", dataset_name, ":\n")
  print(best_tunes[[dataset_name]])
  cat("\n")
}

# Test dataset
for (dataset_name in names(confusion_matrices_test)) {
  cat("Confusion Matrix for", dataset_name, "on Test Data:\n")
  print(confusion_matrices_test[[dataset_name]])
  cat("\n")
}

#Var Importance
var_imp <- varImp(model, scale = FALSE)
print(var_imp)
plot(var_imp, top = 10, main = "Top 10 Variable Importances")


# -----------------------------
# Regression forest
# -----------------------------
data$edited <- factor(data$edited, levels = c(1, 0), labels = c("Class1",
    "Class0"))
data$edited <- factor(data$edited, levels = c(1, 0), labels = c("Class1",
    "Class0"))

train_data2 <- train_data %>% filter(edited == 'Class1')
train_data2 <- train_data2 %>%
  rename(
    comment = PASTABA,
```

```
    status = BUKLE
  )
train_data2 <- train_data2 %>%
  mutate(
    diff_R = turnover - turnoverR,
    VAT_turnover = ifelse(turnover == 0 | is.na(VAT), 0, (VAT - turnover)
        / turnover),
    val_VAT_interaction = val * VAT,
    turnover_VAT_interaction = turnover * VAT,
    turnovery_VAT_interaction = turnover_y * VAT,
    turnover_sq = turnover^2,
    change_turnover_y = turnover - turnover_y,
    turnover_DSK_ratio = ifelse(DSK_SOD == 0, turnover, (turnover / DSK_
       SOD)),
    VAT_DSK_ratio = ifelse(DSK_SOD == 0, VAT, (VAT / DSK_SOD))
  ) %>%  select(everything())


train_data_Class0 <- train_data %>% filter(edited == 'Class0')
train_data_Class0 <- train_data_Class0 %>% mutate(
    diff_R = turnover - turnoverR,
    VAT_turnover = ifelse(turnover == 0 | is.na(VAT), 0, (VAT - turnover)
        / turnover),
    val_VAT_interaction = val * VAT,
    turnover_VAT_interaction = turnover * VAT,
    turnovery_VAT_interaction = turnover_y * VAT,
    turnover_sq = turnover^2,
    change_turnover_y = turnover - turnover_y,
    turnover_DSK_ratio = ifelse(DSK_SOD == 0, turnover, (turnover / DSK_
       SOD)),
    VAT_DSK_ratio = ifelse(DSK_SOD == 0, VAT, (VAT / DSK_SOD))
  ) %>%  select(everything())
train_data_Class0 <- train_data_Class0 %>%  rename(
    comment = PASTABA,
    status = BUKLE
  )

  test_final2 <- original_predictions %>% filter(Predicted_Class == '
     Class1')
test_final2 <- test_final2 %>% mutate(
    diff_R = turnover - turnoverR,
    VAT_turnover = ifelse(turnover == 0 | is.na(VAT), 0, (VAT - turnover)
        / turnover),
```

```
    val_VAT_interaction = val * VAT,
    turnover_VAT_interaction = turnover * VAT,
    turnovery_VAT_interaction = turnover_y * VAT,
    turnover_sq = turnover^2,
    change_turnover_y = turnover - turnover_y,
    turnover_DSK_ratio = ifelse(DSK_SOD == 0, turnover, (turnover / DSK_
       SOD)),
    VAT_DSK_ratio = ifelse(DSK_SOD == 0, VAT, (VAT / DSK_SOD)),
    quarter = as.factor(quarter)
  ) %>%  select(everything())

# Benchmark
rmse <- sqrt(mean((test_final2$turnover - test_final2$turnoverR)^2))
r2   <- cor(test_final2$turnover, test_final2$turnoverR)^2
mae  <- mean(abs(test_final2$turnover - test_final2$turnoverR))
# Print the results
cat("Model Performance Metrics:\n")
cat("--------------------------\n")
cat(sprintf("RMSE: %.4f\n", rmse))
cat(sprintf("R²  : %.4f\n", r2))
cat(sprintf("MAE : %.4f\n", mae))

# Additing Class0 to training
num_train <- nrow(train_data2)
num_to_sample <- floor(2 * num_train)

set.seed(123)
replace_flag <- ifelse(num_to_sample > nrow(train_data_Class0), TRUE,
   FALSE)

sampled_Class0 <- train_data_Class0[sample(
  nrow(train_data_Class0),
  size = num_to_sample,
  replace = replace_flag
), ]

# Code was done manually rather than function
train_data_0.5 <- rbind(train_data2, sampled_Class0)
train_data_1 <- rbind(train_data2, sampled_Class0)
train_data_1.5 <- rbind(train_data2, sampled_Class0)
train_data_2 <- rbind(train_data2, sampled_Class0)
train_data_3 <- rbind(train_data2, sampled_Class0)
```

```r
shuffled_train_data <- train_data_2[sample(nrow(train_data_2)), ]
rownames(shuffled_train_data) <- NULL
shuffled_train_data

# ----------------------------
# TRAINING
# ----------------------------

formula <- turnoverR ~  turnover +  turnover_y  + val + VAT + DSK_SOD +
    val_VAT_interaction + VAT_turnover


selected_columns <- c("turnoverR", "turnover", "turnover_y", "val", "VAT"
    , "DSK_SOD",  "val_VAT_interaction", "VAT_turnover")
shuffled_train_data_subset <- shuffled_train_data[, selected_columns]
shuffled_train_data_subset <- na.omit(shuffled_train_data_subset)


custom_summary_regression <- function(data, lev = NULL, model = NULL) {
  RMSE_val <- RMSE(data$pred, data$obs)
  R2_val <- R2(data$pred, data$obs)
  MAE_val <- MAE(data$pred, data$obs)

  out <- c(RMSE = RMSE_val, R2 = R2_val, MAE = MAE_val)
  return(out)
}

num_predictors <- length(all.vars(formula)) - 1

grid <- expand.grid(
  mtry = 1:7, # Adjust based on number of predictors
  splitrule = c("variance", "extratrees"),
  min.node.size = seq(1, 5, by = 1)
)

control <- trainControl(
  method = "cv",
  number = 10,
  verboseIter = TRUE,
  summaryFunction = custom_summary_regression,
  savePredictions = "final",
  allowParallel = FALSE
)
```

```
model <- tryCatch({
  train(
    formula,
    data = shuffled_train_data_subset,
    method = "ranger",
    tuneGrid = grid,
    preProcess = c("center", "scale"),
    trControl = control,
    metric = "RMSE",
    importance = 'impurity',
    num.trees = 1000,
    verbose = TRUE
  )
}, error = function(e) {
  cat("Error during model training:\n")
  print(e$message)
  NULL
})

print(model$resample)
model$results[model$results$mtry == model$bestTune$mtry &
+                     model$results$min.node.size == model$bestTune$min.
    node.size, ]

# TEST DATYA
  predictions_test_pred <- predict(model, newdata = test_final2)

  metrics_test_pred <- data.frame(
    RMSE = RMSE(predictions_test_pred, test_final2$turnoverR),
    R2 = R2(predictions_test_pred, test_final2$turnoverR),
    MAE = MAE(predictions_test_pred, test_final2$turnoverR)
  )
  cat("\nTest_Pred Data Metrics:\n")
  print(metrics_test_pred)
} else {
  cat("\nModel training was unsuccessful. Please check the errors above.\
     n")
}


# Var Impr.
var_imp <- varImp(model, scale = FALSE)
```

```
print(var_imp)
plot(var_imp, top = 10, main = "Top 10 Variable Importances")

# Plot min node. size
plot(model)

# Scatter Plots
plot_data_test <- data.frame(
  Actual = test_final2$turnoverR,
  Predicted = predictions_test_pred
)

ggplot(plot_data_test, aes(x = Actual, y = Predicted)) +
  geom_point(alpha = 0.6, color = "blue") +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed", color = "red
     ") +
  theme_minimal() +

  labs(
    title = "TurnoverR vs. Turnover Imputation (Test Data)",
    x = "TurnoverR",
    y = "Imputed Turnover"
  ) +
  theme(
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    axis.title = element_text(size = 14)
  )
```

Listing 1: Full Code