VILNIUS UNIVERSITY

FACULTY OF MATHEMATICS AND INFORMATICS

DATA SCIENCE STUDY PROGRAMME

Master thesis

Nowcasting Lithuanian Gross Domestic Product (GDP) Using Machine Learning Methods

Lietuvos bendrojo vidaus produkto (BVP) prognozavimas realiuoju laiku naudojant mašininio mokymosi metodus

Adomas Kulikauskas

Supervisor : Asist., Dr., Saulius Jokubaitis

Reviewer : Doc., Dr., Dmitrij Celov

Vilnius
2025

Abstract

Gross domestic product (GDP) is one of the main key indicators of a state's economy. Nowadays GDP rates are available only with weeks of delays, using nowcasting methods, GDP predictions are accessible in the first month of the quarter. In this work, a Lithuanian GDP nowcasting system was developed with ten different machine learning models. Also, data provided by the state data agency with appropriate data delays were considered. Practical implementations of nowcasting were introduced with a pseudo-real-time framework, where different data vintages performed different time periods of predictions. The improvements in model performance were introduced using maximal overlapping discrete wavelet transform.

Keywords: Gross domestic product (GDP), Nowcasting, Machine learning, Gradient Boosting Regressor, ARIMA, Maximal overlapping discrete wavelet transform (MODWT).

Santrauka

Bendrasis vidaus produktas (BVP) yra vienas pagrindinių valstybės ekonomikos rodiklių. Šiais laikais BVP rodikliai sužinomi tik po kelių savaičių vėlavimo, naudojant $nowcasting$ metodus BVP prognozės gali būti pasiekiamos jau pirmąjį ketvirčio mėnesį. Šiame darbe buvo sukurta Lietuvos BVP realaus laiko prognozių sistema su dešimčia skirtingų mašininio mokymosi modelių. Taip pat buvo atsižvelgta į valstybinės duomenų agentūros pateiktus duomenis su atitinkamais duomenų vėlavimais. Praktiniai realaus laiko prognozių metodų taikymai buvo pristatyti naudojant pseudo realaus laiko sistemą, kurioje skirtingos duomenų versijos buvo naudojamos skirtingų laikotarpių prognozėms. Šios versijos buvo prognozuojamos naudojant kelis mašininio mokymosi modelius. Modelių veikimo patobulinimai buvo pasiekti naudojant maksimaliai persidengiančią diskrečiąją bangelių transformaciją.

Raktiniai žodžiai: Bendrasis vidaus produktas (BVP), Nowcasting, Mašininis mokymasis, Gradiento didinimo regresorius, ARIMA, Maksimaliai persidengiančių diskrečiąjų bangelių transformacija.

Contents

List of notations

1 table. *Notations*

Notation	Definition
X_t	A vector of observed variables at time t.
L	The lag operator and the lag polynomial matrices $\lambda(L)$ and $\Psi(L)$.
f_t	A vector of unobserved common factors.
ϵ_t	A vector of idiosyncratic components or noise.
Y_{t+h}	The forecast of the low-frequency variable.
h	Periods ahead.
p_Y and q_X	The lag lengths for Y and X, respectively.
$X_{t-k}\theta$	A weighted aggregation of high-frequency data points with a parameter vector θ controlling the weighting scheme.
Z_t	A vector of variables at time t.
A_0	A vector of intercept terms.
A_i (for $i = 1, 2, \ldots, p$)	Matrices of coefficients for the lagged variables.
y_i	The dependent variable.
x_{ij}	The independent variables.
β_j and λ	The coefficients and the regularization parameter, controlling the extent of shrinkage applied to the coefficients.
$\phi(B)$ and $\theta(B)$	Polynomials in the back shift operator B.
$(1-B)^d$	The differencing operator.
Y_t	A time series.
T_t	Trend component (captures long-term changes in the data).
S_t	Seasonal component (captures periodic fluctuations with fixed periodicity).
I_t	Irregular component.
Δy_t	The first difference of the time series ($y_t - y_{t-1}$).
α	An intercept.
βt	A time trend.
γy_{t-1}	A lagged level of the series.
$\sum_{i=1}^{p} \delta_i \Delta y_{t-i}$	A lagged difference of the series.
$\beta_0, \beta_1, \ldots, \beta_p$	Coefficients.

Introduction

In today's geopolitical situation, led by significant complexity, conflicts across the regions, influence of different alliances and economic interests, it is crucial to understand the main macroeconomic indicators of the state, which greatly influence the government's fast decision-making and understanding of the state's opportunities to plan future strategies. Every day economists examine a diverse collection of financial insights from statistical agencies and private and public surveys to evaluate the economy's performance [8]. The practice of gathering expert forecasts has a long-established history. The oldest quarterly survey of macroeconomic forecasts is the Survey of Professional Forecasters (SPF), which began in 1968 and is currently conducted by the Federal Reserve Bank of Philadelphia [31].

Some macroeconomic indicators are calculated at the end of every quarter. The most comprehensive indicator of economic activity is gross domestic product (GDP), which is released with a significant delay and provides valuable insights into a nation's financial health and growth potential [12]. GDP by production approach is the net value of all goods and services produced within the country during the reporting period, i.e. the final result of production activity. GDP at market prices is the sum of the value added of all industries or institutional sectors at basic prices, plus taxes, less subsidies on products [3]. The national statistical agency calculates this indicator in a country, compiling information from many sources. For example, the Lithuanian Department of Statistics estimates GDP linearly, without using any machine learning models, and provides accurate estimates with a delay of weeks. However, this delay can be solved using the nowcasting method, which can help predict these indicators faster. Nowcasting is the prediction of current or near-future values of low-frequency outcome variables using high-frequency data with machine methods or other modelling approaches.

After reviewing the topic's relevance, a natural motivation arises to help the Lithuanian Statistics Department develop a GDP modelling and estimation framework using machine learning models.

Reasearch paper goal will be orientated specifically on nowcasting the Lithuanian gross domestic product (GDP) using machine learning methods and its approaches.

The main tasks of this work will be:

- Review the theory of nowcasting with various of different applications.
- Based on the scientific literature, create a framework for nowcasting the Lithuanian gross domestic product indicator and perform its analysis.
- Implement a maximal overlapping discrete transform (MODWT) into the nowcasting system and improve model predictions.

1 Literature review

The purpose of this section is to review scientific papers regarding the thesis topic and analyze and compare different approaches and techniques. This section will cover dynamic factor models, mixed frequency vector autoregressions and mixed data sampling regressions, sparse and dense techniques, and machine learning methods.

The significance of GDP in measuring the size and performance of the economy is enormous. In the last decade, GDP predictions and forecasts have been considerably researched. There are various nowcasting modelling approaches in the scientific literature. For example, dynamic factor models (DFMs), originally were proposed as a time-series extension of previously developed factor models for cross-sectional data [14]. DFMs are statistical models that assume that a small number of unobserved common factors and idiosyncratic components can explain a large set of observed variables. The basis of a dynamic factor model is that a few hidden dynamic factors, f_t, influence the comovements of a high-dimensional vector of time-series variables, X_t, which is also impacted by a vector of mean-zero idiosyncratic disturbances, e_t [28]. These disturbances result from measurement errors and unique factors specific to individual series [28]. The latent factors follow a time series process, typically represented by a vector autoregression (VAR) [28]. In equations, the dynamic factor model is:

$$X_t = \lambda(L) f_t + e_t \tag{1}$$

$$f_t = \Psi(L) f_{t-1} + \eta_t. \tag{2}$$

The author highlights that this model has some conditions which must be satisfied like both equations are stationary and the idiosyncratic disturbances are assumed to be uncorrelated with the factor innovations at all leads and lags, he also emphasizes an important motivation for considering DFMs is that, if one knew the factors f_t and if (e_t, η_t) are Gaussian, then one can make efficient forecasts for an individual variable using the population regression of that variable on the lagged factors and lags of that variable [28]. So, the problem of estimating these factors and determining their exact number constitutes one of the challenges scientists face in calculating them. This model has a lot of benefits like handling high-dimensional data or incorporation of mixed-frequency data, but it also has challenges with the complexity of the model structure and estimation [6, 8, 15]. For example, comparing DFM models with machine learning methods, such as neural networks and random forests, they can automatically detect complex, nonlinear relationships in data without requiring explicit model specification, they can adapt more flexibly to new data patterns and have shown promising results in handling high-dimensional data more efficiently than traditional DFMs [24].

Looking at more practical uses, nowcasting involves predicting the current state of an economy using the most recent data available. Usually, this data has hundreds of variables, which is hard to handle properly. DFMs reduce the dimensionality of the problem by sum-

marizing the information from a large number of variables into a few common factors. There are some examples, where one of them formalized the process of updating the nowcast and forecast on output, observing that survey variables from the Federal Reserve Bank of Philadelphia have a significant impact on nowcasting, this research was done using dynamic factor models, which became one of the most commonly used approaches [23]. Also, this study has shown that DFMs often outperform traditional models in nowcasting GDP and other economic indicators, especially in real-time settings [23].

Overall, DFMs have become a popular approach in nowcasting due to their ability to efficiently handle and extract information from large and different datasets. However, like any methodology, they face several challenges when compared to other approaches, such as MIDAS (Mixed Data Sampling) models or mixed frequency vector autoregressions (MF-VAR) models, which can handle mixed-frequency data efficiently and are simpler to estimate compared to DFMs or can be powerful for capturing the dynamics between variables at different frequencies, such as monthly and quarterly data, in a coherent framework [20]. Most macroeconomic data are not all sampled at the same frequency, they are sampled monthly or quarterly and the challenge is how to best use available data [5]. MIDAS regressions allow estimating dynamic equations that explain a low-frequency variable by high-frequency variables and their lags [26]. The main idea to handle those frequencies properly is a specific weighting function that aggregates the high-frequency information efficiently. Some researchers released an article that raised the question, do macroeconomists have to use financial data?[4] Doing that paper they provide a detailed explanation of the key fundamentals of MIDAS [4]. It started with the conventional Augmented Distributed Lag (ADL) model used for forecasting low-frequency variables, such as quarterly GDP growth [4]:

$$Y_{t+h} = \alpha + \sum \phi_j Y_{t-j} + \sum \beta_k X_{t-k} + u_{t+h} \tag{3}$$

The MIDAS regression model extended this approach and is formulated as follows [4]:

$$Y_{t+h} = \alpha + \sum \phi_j Y_{t-j} + \sum \beta_k X_{t-k}(\theta) + u_{t+h}. \tag{4}$$

Also, the author highlighted that the weighting scheme $\omega(k;\theta)$ is often specified using the exponential Almon lag polynomial [4]:

$$\omega(k;\theta) = \frac{\mathsf{exp}(\theta_1 k + \theta_2 k^2)}{\sum_{i=0}^{m-1} \mathsf{exp}(\theta_1 i + \theta_2 i^2)}. \tag{5}$$

This scheme ensures that the weights are non-negative and sum to one [4]. This paper showed the main ideas of the MIDAS approach, and how the weighting function handles different frequencies of data points. Similar methods are also used in the MF VAR approach, also known as MR VAR (Mixed-frequency Regression VAR). According to Foroni and Marcellino (2013), MF

VAR models provide a coherent framework for handling mixed-frequency data, leading to improving the accuracy of forecasts and economic analyses [13]. The MIDAS and MR VAR models deal with integrating data sampled at different frequencies, but their structure and methodology differ. For example, MR VAR models extend traditional VAR (Vector Autoregressive) models to handle mixed-frequency data, these models include mixed-frequency data into a VAR framework, often using state-space models and Bayesian techniques to manage the complexities of different sampling rates [13, 25]. The general form of an MR VAR model can be represented as follows:

$$Z_t = A_0 + A_1 Z_{t-1} + A_2 Z_{t-2} + \ldots + A_p Z_{t-p} + \epsilon_t. \tag{6}$$

So, the main difference comparing MIDAS and MF-VAR is mixed-frequency data point handling – MIDAS uses polynomial weighting schemes to integrate high-frequency data, while MF-VAR often applies state-space models and Bayesian techniques to handle mixed-frequency data [4, 25]. The comparison of those two similar models is done and the main conclusion is that there seems to be no clear winner in terms of forecasting performance, but noticed that a combination of forecasts from MIDAS and MF-VAR models yields better results than using single models alone [20]. Over time, combined model compounds have become common applications in research, several output combinations were used as a final product in different analyses [17]. For example, one of the applications is research which examines whether online search engine data are useful for improving the accuracy of tourism demand nowcasting when official statistical data are not available [16]. The study examined whether the LASSO-MIDAS model is effective for nowcasting tourism demand [16]. Authors compared different model approaches like same-frequency OLS-type models and only MIDAS-type models with LASSO-MIDAS extension gain the best outcomes possible [16]. The nowcasting accuracy of the LASSO-MIDAS model was significantly higher than that of other competing models, which confirms the effectiveness of applying the LASSO-MIDAS model to tourism demand nowcasting [16].

Over time, scientists began to be more interested not in the frequency of the received data, but in the amount of variables, they kept trying to answer the question, what is better to use, all the existing variables in the data array? Or to single out the most important ones and examine only their effects. Some researchers reviewed both method's pros and cons, they claimed that on the one hand, sparse modelling, such as LASSO (Least Absolute Shrinkage and Selection Operator), focuses on selecting key explanatory variables to create a predictive model, offering simplicity and clarity, using fewer parameters [7]. However, this technique might miss some intricate details present in the data. On the other hand, dense modelling, like ridge estimator and factor-augmented regression, considers all possible variables, which suggests a power for capturing complex patterns in the data and is often used when precision is less of a concern [7]. To compare those techniques other researchers tried to show the main differences and similarities between Ridge and LASSO approaches.[22] They described key things about ridge regression, which is a method used to analyze multiple regression data

that suffer from multicollinearity and defined the ridge regression estimator [22]:

$$\hat{\beta}^{\text{ridge}} = \arg\min_{\beta} \left(\sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right). \tag{7}$$

The authors highlighted the selection of λ, they claimed that the "ridge" parameter reduces variance and can result in a more reliable model and the main thing is that this approach shrinks all coefficients by the same proportion and does not set any coefficient to zero, ensuring that all variables are included in the model [22]. However, LASSO uses a quite different approach to the objective function [22]:

$$\hat{\beta}^{\text{lasso}} = \arg\min_{\beta} \left(\sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right). \tag{8}$$

which sets some coefficients to zero, effectively selecting a simpler model that includes only the most significant predictors. Authors conclude that ridge regression is more useful when all variables are believed to have an effect and multicollinearity needs to be managed, while LASSO performs when variable selection and model simplicity are more important [22].

Finally, today's most popular and trending machine learning (ML) methods include the decision tree, gradient-boosted trees, random forest, XGBoost, and autoregressive integrated moving average (ARIMA) models, which are common nowcasting approaches recently. One example of this is a research estimation of New Zealand's GDP using those particular models, the researchers compared nowcasts, against a naive ARIMA benchmark, a dynamic factor model, and the official forecasts by the Reserve Bank of New Zealand [24]. They found that the ML models produced more accurate estimates than the ARIMA and dynamic factor models [24]. Additionally, the results suggest that the Reserve Bank of New Zealand could have improved their forecast accuracy by utilizing ML models [24]. Also popular approach in ML is ensemble methods, which work by combining the outputs of multiple models to enhance the accuracy of predictions.[27] This increased accuracy has made ensemble methods very popular in machine learning. One of the examples showed practical applications of using tree-based ensemble models of nowcasting US GDP growth rates [27]. Authors used bagged decision trees, which aggregate multiple decision trees to reduce variance, random forests, which enhance bagged trees by decorrelating them and stochastic gradient boosting, which builds trees sequentially, each new tree correcting errors made by the previous ones [27]. In this study, the ensemble models approach was compared with DFMs in nowcasting US GDP, the results were significantly better when ensemble methods were applied [27].

Nowcasting methods include a wide range of different approaches, primarily based on statistical models such as factor models or MIDAS. All of them suggest various useful applications, factor models analyze latent structures to identify common drivers, and MIDAS models effectively integrate data of different frequencies, enhancing forecast accuracy.[28][4] Com-

pared with ML models like XGBoost or Decision Trees, they do not have such statistical applications as model systems, latency space or data of different frequencies. However, ML models have different strengths, for models based on decision trees their advantage is in capturing non-linear dependencies even with large datasets, they allow for ensemble extensions, such as Random Forest and XGBoost, which improve robustness and accuracy.[17] Also ML approaches like decision trees provide automated variable selection and adaptability to high-dimensional data structures.[11] Later on in this thesis, these ML models will be reviewed and used to forecast Lithuania's GDP.

2 Data preparation and methodology

This section will cover data preparation, the creation of the whole nowcasting algorithm, and an explanation of all the modelling methods used in this research. Data was received from the Lithuanian Department of Statistics. The dataset consists of several tables with various data points. A table with monthly data was chosen for the nowcasting framework because it is crucial to have the monthly frequency for better accuracy of nowcasting GDP during the months. Most of the columns have all values, missing points are filled using column means.

2.1 Seasonal decomposition using X-13 ARIMA SEATS

For nowcasting time-series macroeconomic data, it is crucial to use data without season impact and trends. There are several methods to decompose time series data and smooth their seasonality. The most suitable method is X-13-ARIMA-SEATS, it is a sophisticated statistical tool developed by the U.S. Census Bureau and the Bank of Spain for seasonal adjustment and time series analysis. Eurostat uses this method for seasonal adjustments [10]. Unfortunately, Lithuanian Statistics did not have all the necessary data after seasonal decomposition, therefore, this data seasonal smoothing was applied from my side, using R package seasonal.

X-13-ARIMA-SEATS method combines features of two earlier methods, X-11 and SEATS (Signal Extraction in ARIMA Time Series), with enhancements for automation, diagnostics, and flexibility. The main goal of this approach is to decompose a time series into trend, seasonal, and irregular components, for more accurate forecasting and decision-making. Using ARIMA (AutoRegressive Integrated Moving Average) model predicts the time series behaviour by expressing some features such as autoregressive terms, where the current value depends on previous values, integration feature, which handle non-stationary by differencing and moving average, which account for past forecast errors. The general ARIMA form is:

$$\phi(B)(1-B)^d Y_t = \theta(B)\epsilon_t. \tag{9}$$

The whole process is to decompose a time series into three main components:

$$Y_t = T_t + S_t + I_t$$

The SEATS part of the decomposition is treated as a signal extraction problem, using the ARIMA model, it separates the time series into three signals, respectively, as in seasonal decomposition. The extraction minimizes the mean squared error (MSE), ensuring optimal signal recovery. X-13 part uses symmetric and asymmetric moving averages filters from X-11 to filter and adjust short-term fluctuations and outliers. Also, this method has some diagnostic tools to evaluate the diagnostic such as autocorrelation function (ACF) and partial autocorrelation

function (PACF) for residual analysis, quality measures for seasonal adjustment and outlier detection for additive, level shift, and temporary change outliers.

2.2 Data stationary tests

After smoothing the seasonal effect of the data, stationary tests were checked. Stationarity is essential for many time-series modelling techniques used in nowcasting, it improves nowcasts accuracy. Removing non-stationarity helps isolate the true relationships between variables, ensuring that forecasts reflect actual economic dynamics.

2.2.1 Augmented Dickey-Fuller test

The Augmented Dickey-Fuller (ADF) test is a statistical method which examines the stationarity of a time series. The ADF test evaluates the null hypothesis (H_0) that the time series has a unit root, which implies nonstationarity. The alternative hypothesis (H_1) states that the series is stationary. Mathematically, the ADF test evaluates the following regression equation:

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \sum_{i=1}^{p} \delta_i \Delta y_{t-i} + \epsilon_t$$

The key parameter of interest is γ, which measures the presence of a unit root. The null and alternative hypotheses are expressed as:

$$H_0 : \gamma = 0 \quad \text{(the series has a unit root and is non-stationary)}$$
$$H_1 : \gamma < 0 \quad \text{(the series is stationary)}.$$

Using ordinary least squares (OLS), the parameter γ is estimated. This parameter determines the presence of a unit root. The test statistic is computed as:

$$\text{ADF Statistic} = \frac{\hat{\gamma}}{\text{SE}(\hat{\gamma})}$$

here $\hat{\gamma}$ is the estimated coefficient of y_{t-1}, and $\text{SE}(\hat{\gamma})$ is its standard error. Since the null hypothesis assumes that there is a unit root, the p-value obtained from the test should be less than the significance level to reject the null hypothesis. Thus, it can be concluded that the series is stationary.

An important thing in the ADF test is the selection of the lag length (p) for the augmented term, as it effects the test's power and accuracy. Lags are typically chosen using information criteria such as the Akaike Information Criterion (AIC) or the Bayesian Information Criterion (BIC). The difference between these two methods is how they penalize model complexity, AIC penalizes complexity less, leaning toward bigger models with more lags and BIC penalizes complexity more heavily, favouring simpler models with fewer lags.

2.2.2 Breusch-Pagan test

The Breusch-Pagan test is a statistical test used to detect heteroscedasticity in a regression model. When the variance of the residuals varies across all levels of the independent variables, this is known as heteroscedasticity, which can lead to wrong estimates and biased standard errors, effecting hypothesis test outcomes. This violates one of the basic assumptions of ordinary least squares (OLS) regression, which is that the errors are assumed to be homoscedastic. The main idea of the Breusch–Pagan test is to check any regression residuals heteroscedasticity. For example, we have linear regression:

$$y_n = \beta_0 + \beta^T X_n + \epsilon_n, \tag{10}$$

After that, express it variance of each reference point as a function of $f(\cdot)$, which does not depend on n:

$$\sigma_n^2 = f(\alpha_0 + \alpha^T X_n). \tag{11}$$

Here, $\alpha = [\alpha_1 ... \alpha_P]^T$ is a P-vector of coefficients which are independent of the coefficients β. We know that homoscedasticity can be written like the equivalence of a null hypothesis:

$$H_0 : \alpha_1 = \alpha_2 = ... = \alpha_P = 0. \tag{12}$$

Homoscedasticity conditional variance of each error term would not depend on n or X_n, for example:

$$\mathsf{V}[\varepsilon_n \mid X] = \sigma_n^2 = f(\alpha_0). \tag{13}$$

Regarding equation (13) the σ_n^2 is a constant. So the main idea of this test is to fit a regression (10), estimate the error terms variance using squared residuals, run another regression (11) to estimate α and check its limit to zero.

2.2.3 Phillips-Perron test

The Phillips-Perron test is similar to the ADF test, but it is a bit different, in how they deal with correlation and heteroskedasticity in the errors. ADF tests rely on a parametric autoregressive model to approximate the ARMA structure of the errors in the test regression, Phillips-Perron test ignores any serial correlation in the test regression. The regression of the test is:

$$\Delta y_t = \beta' D_t + \pi y_{t-1} + u_t, \tag{14}$$

here u_t is $I(0)$ and may be heteroskedastic. The Phillips-Perron test makes a change in any serial correlation and heteroskedasticity in the errors u_t of the test regression changing the

test statistics $t_{\pi=0}$ and T_π. Appears two statistics Z_t and Z_π:

$$Z_t = \left(\frac{\hat{\sigma}^2}{\hat{\lambda}^2}\right)^{1/2} \cdot t_{\pi=0} - \frac{1}{2}\left(\frac{\hat{\lambda}^2 - \hat{\sigma}^2}{\hat{\lambda}^2}\right) \cdot \left(\frac{T \cdot \mathsf{SE}(\hat{\pi})}{\hat{\sigma}^2}\right) \tag{15}$$

$$Z_\pi = T\hat{\pi} - \frac{1}{2}\left(\frac{T^2 \cdot \mathsf{SE}(\hat{\pi})}{\hat{\sigma}^2}\right)(\hat{\lambda}^2 - \hat{\sigma}^2) \tag{16}$$

The variables $\hat{\sigma}^2$ and $\hat{\lambda}^2$ are an outcomes of variance parameters

$$\sigma^2 = \lim_{T\to\infty} T^{-1} \sum_{t=1}^{T} E[u_t^2] \tag{17}$$

$$\lambda^2 = \lim_{T\to\infty} \sum_{t=1}^{T} E\left[T^{-1} S_T^2\right], \tag{18}$$

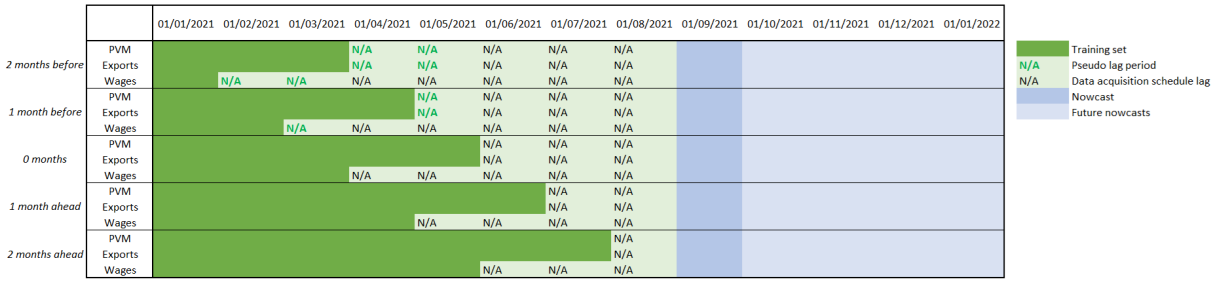
here $S_T = \sum_{t=1}^{T} u_t$. If $\pi = 0$ (null hypothesis), the Z_t and Z_π statistics have the same asymptotic distributions as the ADF t-statistic and normalized bias statistics. The main difference between the Phillips-Perron test and the ADF test is that the Phillips-Perron test is robust to general forms of heteroskedasticity in the error term u_t, also it does not require any specific lag length for the regression.

All these methods were applied and constructed a framework that evaluating the stationarity of a time series involves multiple steps. Firstly, the ADF test was applied, which used a regression model and calculated the t-statistic. Given that the outcome of the ADF test is dependent on the t-statistic, its reliability is checked by verifying whether the errors from the model meet the homoscedasticity assumption. To do this, the Breusch-Pagan test was applied to the residuals of the ADF model. If homoscedasticity is satisfied, the ADF test results can be trusted. However, if this assumption is rejected, stationarity was estimated using the Phillips-Perron test. Differentiation was applied to non-stationary columns and the whole process was applied again.

2.3 Pseudo real-time nowcasting framework

One of the main challenges of estimating GDP is data lag. Usually, economic data becomes available after several months or even quarters, this means that economists or government institutions could make late data-driven decisions and be behind others compared with global trends. To get around these problems, the nowcasting algorithm could be implemented in pseudo-real-time prediction. Creating a pseudo-nowcasting framework can help estimate data lags and evaluate prediction accuracy using different time vintages. For this, the data lag gathering schedule and lag vintages for each calculation should be done. According to Lithua-

nian Statistics [1], all data variables from different areas come at various times, usually, it is the estimation time. Lag vintages usually are 2 months before, 1 month before, current time (0 months), 1 month ahead and 2 months ahead. When all conditions are set, the main framework logic can be explained. Below (1 figure.) is a short visualization and explanation of how this framework works.



1 figure. *Pseudo real-time framework*

Firstly, the target month is chosen as the reference point for the prediction at the start of the nowcasting process. From this point, data acquisition schedules and pseudo lags are subtracted to account for the timing of data availability. Missing values are imputed using column means to ensure a complete dataset for modelling. With the prepared dataset, the nowcast is predicted using different models.

2.4 Machine learning models

This chapter will cover all machine learning (ML) models used in practical work. Today machine learning algorithms play a crucial role in data science and modeling, no exception in macroeconomic metrics such as GDP. ML models become a common approach in nowcasting GDP indicators, possibilities of handling huge amounts of data, finding difficult patterns there and predicting impressive accuracy prediction ML models are receiving more and more positive feedback. In the following sections, the ML models used in the research are described, and their main ideas are also explained.

2.4.1 XGBoost

XGBoost is a gradient-boosting algorithm that builds models by sequentially adding trees to minimize a specific loss function. The explanation of this model begins with the objective function which combines the loss function and a regularization:

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{t=1}^{T} \Omega(f_t),$$

here $l(y_i, \hat{y}_i)$ is the loss function measuring the error between the true value (y_i) and the predicted value $(\hat{y}_i)$ and $\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$ is a regularization, T number of leaves

in the tree, w_j is leaf weights and γ, λ are regularization parameters. This equation includes functions as parameters and cannot be optimized using traditional optimization methods in Euclidean space [9]. This equation is upgraded in an additive manner:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \sum_{t=1}^{T} \Omega(f_t),$$

by adding the most improved model f_t value. To optimize the objective function this model uses a second-order Taylor expansion of the loss function:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t),$$

here $g_i = \frac{\partial l(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}}$ is a first-order gradient and $h_i = \frac{\partial^2 l(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)^2}}$ is a second-order derivative. After optimization of an equation, the tree could be created using:

$$\mathcal{L}_{tree} = \sum_{j=1}^{T} \left[\frac{\left(\sum_{i \in I_j} g_i \right)^2}{\sum_{i \in I_j} h_i + \lambda} \right] - \gamma T,$$

here I_j is the set of instances in leaf j. Trees are added one at a time, and each tree is designed to reduce the residual errors from the previous trees. This is achieved by learning from the gradient of the loss function. The gradient measures how much the prediction needs to change to reduce the error. The more trees XGBoost has, the better it can capture complex patterns in the data. Regularization ensures that the model does not overfit by limiting tree depth.

2.4.2 Lasso Regression

Lasso regression (Least Absolute Shrinkage and Selection Operator) is a linear regression type that uses a penalty term to simplify the models.[29] It is useful for feature selection and regularization when datasets have many features. Lasso regression aims to balance the model's simplicity and accuracy properly. This is accomplished by including a penalty term in the linear regression model, which has sparse solutions by requiring some coefficients to be zero. The model starts with the linear regression model, which has a linear relationship between the independent and dependent variables.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \epsilon$$

Next, the Lasso model uses the penalty term for the linear regression model coefficients. The L1 regularization term is a tuning parameter λ multiplied by the sum of the absolute values of

the coefficients.[29]

$$L_1 = \lambda \sum_{j=1}^{p} |\beta_j|$$

The whole Lasso objective function consists of two parts: L_1 regularization and Least Square Fit, which is the residual sum of squares (RSS).[29] It measures how well the model fits the data. RSS can be expressed as:

$$\text{RSS} = \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^\top \beta\right)^2,$$

here y_i is the observed value, $\mathbf{x}_i^\top \beta$ is the predicted value. By adding these two equations into one we have the objective function in the Lasso regression:

$$J(\beta) = \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^\top \beta\right)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

This function ensures that the model fits the data well by minimizing prediction errors and penalizes large coefficients to control model complexity and reduce overfitting.[29] After constructing the objective function it is crucial to minimize this function and apply a variant of gradient descent because the L_1 norm $\sum |\beta_j|$ is non-differentiable at zero.[29] Instead of standard gradient descent, the coordinate descent or subgradient methods should be used:

$$\frac{\partial}{\partial \beta_j} \left(\frac{1}{2n} \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^\top \beta\right)^2 \right) = -\frac{1}{n} \sum_{i=1}^{n} x_{ij} \left(y_i - \mathbf{x}_i^\top \beta\right),$$

$$\frac{\partial}{\partial \beta_j} \left(\lambda \sum_{j=1}^{p} |\beta_j| \right) = \begin{cases} \lambda & \text{if } \beta_j > 0 \\ -\lambda & \text{if } \beta_j < 0 \\ \text{undefined} & \text{if } \beta_j = 0 \end{cases}$$

For $\beta_j = 0$, the subgradient method finds an optimal value by balancing between shrinking to zero and the effect of the RSS term.[29] Also coordinate descent method might be used, fixing all coefficients except one β_j and updating it by solving a one-dimensional optimization problem:

$$\beta_j = \text{sign}\left(z_j\right) \max\left(|z_j| - \frac{\lambda}{2n}, 0 \right)$$

here:

$$z_j = \frac{1}{n} \sum_{i=1}^{n} x_{ij} \left(y_i - \mathbf{x}_i^\top \beta + \beta_j x_{ij}\right)$$

This formula uses the soft thresholding operator to shrink β_j to zero, this flow should be repeated for all coefficients until convergence. When the optimization is complete, some of the

coefficients are zero, they become irrelevant for future prediction, and non-zero coefficients indicate important features that contribute to the model.

2.4.3 Ridge regression

Ridge regression is a linear regression technique that adds L_2 regularization to the ordinary least squares (OLS) regression. The goal of this method is to prevent overfitting by shrinking the coefficients of the model. This approach is very similar to Lasso regression, but it has one major difference, Ridge regression does not perform feature selection, predictors remain in the model. The first part of the objective function from Lasso is the same as in Ridge, it differs only through the regularization function, Ridge uses L_2 which is represented as:

$$L_2 = \lambda \sum_{j=1}^{p} \beta_j^2$$

The larger the value of λ, the more the coefficients are penalized. For minimizing the objective function, the Ridge model applies gradient descent, which is an iterative method for finding the minimum of a function. The gradient of the RSS term is the same as the Lasso regression, and the gradient of the L_2 regularization term is:

$$\frac{\partial}{\partial \beta_j} \left(\lambda \sum_{j=1}^{p} \beta_j^2 \right) = 2\lambda \beta_j$$

The total gradient for the objective function is the sum of the gradients of the two terms:

$$\frac{\partial J(\beta)}{\partial \beta_j} = -\frac{1}{n} \sum_{i=1}^{n} x_{ij} \left(y_i - \mathbf{x}_i^\top \beta \right) + 2\lambda \beta_j$$

Using gradient descent, the coefficients β_j are updated iteratively as follows:

$$\beta_j^{(t+1)} = \beta_j^{(t)} - \eta \left(-\frac{1}{n} \sum_{i=1}^{n} x_{ij} \left(y_i - \mathbf{x}_i^\top \beta \right) + 2\lambda \beta_j^{(t)} \right)$$

here η is the learning rate, $\frac{1}{n} \sum_{i=1}^{n} x_{ij} \left(y_i - \mathbf{x}_i^\top \beta \right)$ is the gradient of the RSS term. The λ dependent penalty term shrinks the coefficients β_j. As λ increases, the coefficients are shrunk more toward zero, but unlike than Lasso regression, Ridge does not set any coefficients exactly to zero, this means that this model does not perform feature selection.

2.4.4 K-Nearest Neighbors (KNN) Regressor

The K-Nearest Neighbors (KNN) Regressor is a simple, non-parametric machine learning algorithm used for regression tasks. Unlike parametric models, KNN makes predictions based

on the k nearest data points in the feature space. The goal of the KNN model is to k training examples that are nearest to the new data point x_* and predict y_* as the average of the corresponding target values of those k nearest neighbours.

$$y_* = \frac{1}{k} \sum_{i \in N_k(x_*)} y_i$$

here y_* is the predicted target value for the new data point x_*, $N_k(x_*)$ is the set of indices of the k nearest neighbours to x_*, y_i is the target value and k is the number of nearest neighbours considered for the prediction. Next, the closest x_* must be found, for this the distance metric should be used, for example, Euclidean distance:

$$d(\mathbf{x}_i, \mathbf{x}_*) = \sqrt{\sum_{j=1}^{p} (x_{ij} - x_{*j})^2}$$

after calculating each x distance the smallest one (k) should be picked and computed for predicted value y_* by averaging the target value y_i of the k nearest neighbors. The choice of k is crucial, the small value leads to a model that is sensitive to noise and outliers, and the larger value is less sensitive and smoother, but it may not capture important features. For better selection, the cross-validation function is used. For time series data common cross-validation function is time series split (from Scikit-learn), this method is similar to the expanding window approach but is implemented efficiently to split the data into multiple training and test sets. It creates k splits, and each split consists of a training set and a test set, where the training set progressively grows.

2.4.5 Linear Regression

Linear Regression is a machine learning method used to model the relationship between a dependent variable y and one or more independent variables x_1,x_2,...x_p. This model is widely used in statistical and machine learning modelling due to its simplicity and efficiency. Multiple linear regression, with multiple independent variables:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \epsilon$$

The objective function of linear regression is to find values of the β_0,β_1,...β_p coefficients that minimize the error between the predicted values and observed values. The error is typically measured using the Residual Sum of Squares (RSS):

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}))^2$$

optimization is done using the function:

$$\min_{\beta_0,\beta_1,\ldots,\beta_p} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

the solution of the optimization problem is solved using matrix notation:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

here **y** is the $n \times 1$ vector of observed values, **X** is the $n \times (p+1)$ matrix of predictors and $\boldsymbol{\beta}$ is the $(p+1) \times 1$ vector of coefficients. Coefficients are solved:

$$\boldsymbol{\beta} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

after fitting the model several accuracy metrics could be calculated, such as mean squared error or R-squared, these metrics evaluate the model's accuracy and performance.

2.4.6 Elastic Net

Elastic Net is a type of regularized regression that linearly combines the penalties of Lasso Regression (L_1 regularization) and Ridge Regression (L_2 regularization). This combination allows Elastic Net to handle a larger number of features with high correlation. The objective function is:

$$\mathcal{L}(\beta) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \left[\alpha \sum_{j=1}^{p} |\beta_j| + \frac{(1-\alpha)}{2} \sum_{j=1}^{p} \beta_j^2 \right]$$

here α is a mixing parameter, which balances the L_1 (Lasso) and L_2 (Ridge) penalties, when $\alpha = 1$ it is a Lasso regression, when $\alpha = 0$ it is Ridge regression, when $0 < \alpha < 1$ it is a combination of both. To solve β coefficients Elastic Net uses optimization functions such as:

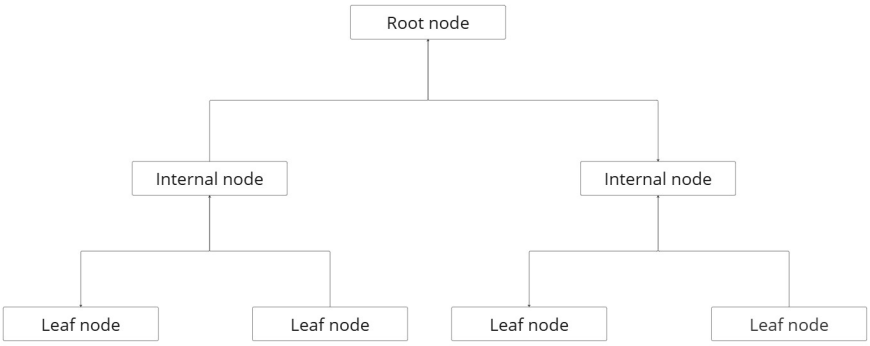
$$\min_{\boldsymbol{\beta}} \ \frac{1}{2n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \left[\alpha \|\boldsymbol{\beta}\|_1 + \frac{(1-\alpha)}{2} \|\boldsymbol{\beta}\|_2^2 \right]$$

here $\|\boldsymbol{\beta}\|_1 = \sum_{j=1}^{p} |\beta_j|$ is L_1 norm, $\|\boldsymbol{\beta}\|_2^2 = \sum_{j=1}^{p} \beta_j^2$ is $L_1 = 2$ norm. Optimize this using coordinate descent or gradient-based optimization in the same way as in Lasso or Ridge regression. Elastic Net is a combination of two powerful models, it allows to be a flexible method for better accuracy and performance. Selection of either L_1 or L_2 penalties or using both at the same time shows efficiency and strength.

2.4.7 Decision Tree

A decision tree is a non-parametric supervised learning algorithm used for classification and regression tasks. This approach has a tree structure, where internal nodes correspond to

decision tests, branches represent outcomes of those tests, and leaf nodes indicate predictions. A basic model structure is defined in 2 figure.



2 figure. *Decision tree structure*

here the root node is the original choice or a feature from which the tree branches begin, internal nodes stand for the nodes in the tree whose choices are determined by the values of particular attributes, and leaf nodes are decided upon. The main idea of this method is to predict a constant value for each region R_m, which is the mean of the target values in that region:

$$\hat{y}(x) = \frac{1}{|R_m|} \sum_{x_i \in R_m} y_i$$

here R_m is the region to which x belongs and $|R_m|$ is the number of samples R_m. Constructing a decision tree it is important to find the appropriate variables from all attributes. Solving this problem there are several approaches such as entropy, Gini index or variance reduction. For regression task variance reduction is a common choice, and splits are evaluated based on the reduction in variance:

$$\text{Variance} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \bar{y})^2$$

here $\bar{y}$ is the mean of y_i in the current node. This approach finds the best split of all nodes. Another crucial problem for the decision tree model is overfitting, to avoid this kind of problem regularization methods could be applied, such as restrictions of tree depth, the requirement for a minimum number of samples in each leaf node or removing branches with low importance.[30]

2.4.8 Support Vector Regression (SVR)

Support Vector Regression (SVR) is an extension of Support Vector Machines (SVM) for regression tasks. The main goal of this method is to find a function that approximates the target values within a margin of tolerance and minimizes the model complexity. SVR model

aims to find a function at time t such as:

$$\hat{y}_t = f(x_t) = w^\top \phi(x_t) + b$$

here $\phi(x_t)$ is a kernel-transformed feature vector (for non-linearity), w and b are model parameters learned during training. The objective function of SVR is to minimize a function:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C \sum_{t=p+1}^{n} (\xi_t + \xi_t^*)$$

here:

$$y_t - f(x_t) \leq \epsilon + \xi_t$$

$$f(x_t) - y_t \leq \epsilon + \xi_t^*$$

$$\xi_t, \xi_t^* \geq 0$$

and ξ_t, ξ_t^* are slack variables for deviations beyond ϵ, C is a regularization parameter controlling the trade-off between model complexity and tolerance to error, $\frac{1}{2}\|w\|^2$ is a regularization term. SVR model is capable of finding non-linear patterns over time, for this reason, SVR could apply a kernel trick to model these relationships effectively:

$$K(x_i, x_j) = \phi(x_i)^\top \phi(x_j)$$

common kernel methods are linear, polynomial or RBF kernels, it is useful to run all of them and find the most suitable one. After finding a solution from the objective function, the performance metrics like mean absolute error, root mean squared error or mean absolute percentage error could be calculated. This model is a powerful approach for its capability to handle non-linear relationships and provide robust predictions.

2.4.9 Gradient Boosting Regressor

Gradient Boosting Regressor (GBR) is an ensemble learning technique that combines the prediction of weaker decision trees sequentially. The GBR model is a powerful model for building predictive models for both classification and regression problems. The model explanation begins from a naive prediction of $F_0(x)$ on the target for a starting point when it iteratively improves the model by adding a new base learner $h_m(x)$ at each step:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x),$$

here $\gamma_m h_m(x)$ is a new tree, which is trained to correct the errors made by $F_{m-1}(x)$. At each m stage, the loss function L is minimized using gradient descent:

$$r_i^{(m)} = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\bigg|_{F(x)=F_{m-1}(x)},$$

here $r_i^{(m)}$ is the pseudo-residual at stage m for the i-th sample. After optimization of the tree, a new tree is fitted to the pseudo-residuals:

$$h_m(x) = \arg\min_h \sum_{i=1}^{n} \left(r_i^{(m)} - h(x_i)\right)^2.$$

This tree attempts to approximate the gradient of the loss function. Next, the weighting of the base learner should be applied for the new tree $h_m(x)$ by calculating:

$$\gamma_m = \arg\min_\gamma \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

After finding the optimal weight γ_m for the new tree $h_m(x)$, the model is updated:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

this flow repeats until the stopping criterion. For the Gradient Boosting Regressor model, it is crucial to select a proper loss function. The different loss functions can vary depending on the problem. The most common function is a mean squared error (MSE) for regression problems:

$$L(y, F(x)) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - F(x_i)\right)^2.$$

for handling variable outliers common approach is Huber loss:

$$L(y, F(x)) = \begin{cases} \frac{1}{2}(y - F(x))^2 & \text{, if } |y - F(x)| \le \delta, \\ \delta|y - F(x)| - \frac{\delta^2}{2} & \text{, otherwise.} \end{cases}$$

also, this model has some regularization like learning rate, it scales the contribution of each tree, or maximum depth of tress or minimum samples per leaf, these all things after m iterations train and improve a final prediction for a sample x:

$$\hat{y} = F_M(x) = \sum_{m=1}^{M} \gamma_m h_m(x).$$

The main difference between the XGBoost model is that the Gradient Boosting Regressor is much slower, it builds trees sequentially instead of utilizing parallel processing, also this model

does not use any regularization techniques like L1 and L2 (Ridge and Lasso) for overfitting as XGBoost does. However, the Gradient Boosting Regressor is much simpler and effective for smaller datasets.

2.4.10 ARIMA

ARIMA is a widely used statistical model, and the variety of applications is huge. Starting from financial market forecasting and ending healthcare, where ARIMA could be applied to analyze disease outbreaks and predict future infection rates. The general form of ARIMA (9) was previously described in X-13 ARIMA SEATS applications. Using this model data must be stationary due to this the data stationary test was applied first. The whole modelling process begins with a time series Y_t fitting to the autoregressive model:

$$Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \ldots + \phi_p Y_{t-p} + \epsilon_t$$

here p is the previous values of time series Y_t. Then the moving average component models the current value of the series as a function of past forecast errors (ϵ_t):

$$Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \ldots + \phi_p Y_{t-p} + \epsilon_t.$$

ARIMA combines these models into one framework and the final form is:

$$\phi(B)(1-B)^d Y_t = \theta(B)\epsilon_t$$

here $\phi(B)$ is an autoregressive polynomial of order p, $(1-B)^d$ is a differencing operator applied d times, $\theta(B)$ is a moving average polynomial of order q. The crucial thing is to select good values of p, d and q parameters. For this could be used Auto ARIMA function, which automates the process of selecting the best ARIMA model by determining the optimal values of p, d and q based on information criteria such as AIC (Akaike information criterion). Auto ARIMA performs a grid search over combinations of p, d and q to find the model that minimizes AIC. The AIC is calculated as:

$$AIC = -2\ln(L) + 2k$$

here L is the maximum likelihood of the model and k is several parameters. Also, this function automatically applies unit root tests for data stationary determines the order of d parameter and can work with seasonal data (SARIMA). Finally, once the model fitting is done, the future values could be predicted iteratively using the fitted autoregressive and moving average components, for h-step ahead forecasts:

$$\hat{Y}_{t+h} = \phi_1 \hat{Y}_{t+h-1} + \phi_2 \hat{Y}_{t+h-2} + \ldots + \theta_1 \epsilon_{t+h-1} + \ldots$$

This model is a good statistical method to forecast time series data and the method is suitable for use in conjunction with other mathematical applications.

2.5 Diebold-Mariano test

For evaluation and decision on which model is the best one and has the most significant impact, the Diebold-Mariano (DM) test was selected. It is a statistical test used to compare the predictive accuracy of two competing forecasting models. It estimates the significant difference between the forecast errors. This approach is popular for time series analysis to check which one of the forecasts outperforms another in terms of prediction quality. For instance, we have two forecast errors $\{e_{1,t}\}_{t=1}^{T}$ and $\{e_{2,t}\}_{t=1}^{T}$ at time t. Now let's define the loss function $g(e_t)$ where its equal to e_t^2 difference:

$$d_t = g(e_{1,t}) - g(e_{2,t}),$$

here d_t measures the difference in accuracy between the two models at time t. For the null hypothesis part of the test is defined that there is no difference in predictive accuracy between the two models:

$$H_0 : \mathbb{E}[d_t] = 0,$$

the alternative hypothesis claims that:

$$H_a : \mathbb{E}[d_t] \neq 0,$$

a significant difference could be found in the accuracy perspective. To estimate the final test statistic the mean and variance losses are missing. These metrics are defined respectively:

$$\bar{d} = \frac{1}{T}\sum_{t=1}^{T} d_t.$$

$$\mathsf{Var}(\bar{d}) = \frac{\gamma_0 + 2\sum_{k=1}^{h-1}\gamma_k}{T},$$

here γ_k is the k-lag autocovariance of d_t, capturing temporal dependence. Finally, test statistics could be solved as:

$$DM = \frac{\bar{d}}{\sqrt{\mathsf{Var}(\bar{d})}}.$$

Under the null hypothesis, DM asymptotically follows a standard normal distribution:

$$DM \sim N(0, 1).$$

It is crucial to select critical values to estimate the final insights. Also, there are several choices of loss function, such as absolute error which may be preferred for robustness against outliers or mean absolute percentage error.

2.6 Maximal overlap discrete wavelet transform

In this study, it was chosen to conduct several experiments on how to improve the performance of nowcasting models in ways that no one had tried before. After reviewing the literature and discovering several effective uses of wavelets in time series data, it was decided to try to investigate the integration of MODWT into an existing nowcasting system.[18][2] The Maximal overlap discrete wavelet transform (MODWT) is a wavelet transform variant designed to expand the time series into multiple resolutions while maintaining the original structure and timing of the dataset. Unlike the traditional Discrete wavelet transform (DWT) MODWT is not orthonormal and is defined for all sample sizes. The main idea of MODWT is to decompose any signal $X(n)$ of length N, in our case a time series, into detailed coefficients $W_j(n)$ and approximation coefficients $V_j(n)$, it could be expressed like this:

$$W_j(n) = \sum_{k=0}^{L-1} h_k^{(j)} X(n-k) \mod N$$

$$V_j(n) = \sum_{k=0}^{L-1} g_k^{(j)} X(n-k) \mod N$$

here h_k is a wavelet filter coefficients, g_k is a scaling filter coefficients, L is a length of the filter, $(\mod N)$ ensures periodic boundary conditions.[32] Also unlike DWT, MODWT does not perform downsampling, which makes the length of the coefficient equal to the original length N at each scale.[32] For wavelet filters:

$$h_k^{(j)} = h_k^{(j-1)} \uparrow 2^{j-1}$$

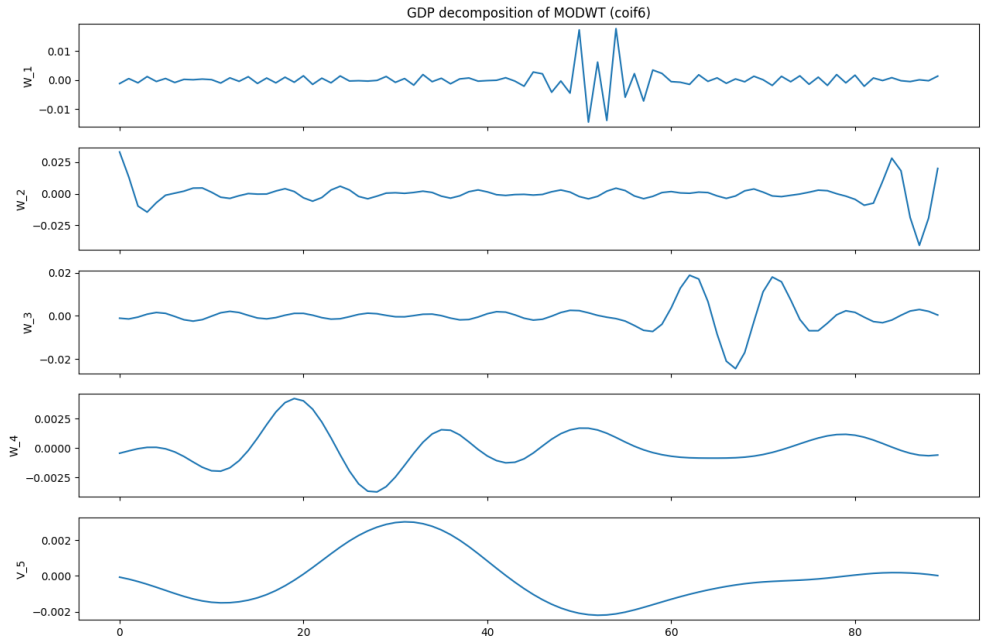
for scaling filters:

$$g_k^{(j)} = g_k^{(j-1)} \uparrow 2^{j-1}$$

here $\uparrow 2^{j-1}$ denotes upsampling by inserting zeros between coefficients. The reconstruction of the original series can be done by summing contributions from all scales:

$$X(n) = \sum_{j=1}^{J} D_j(n) + A_J(n)$$

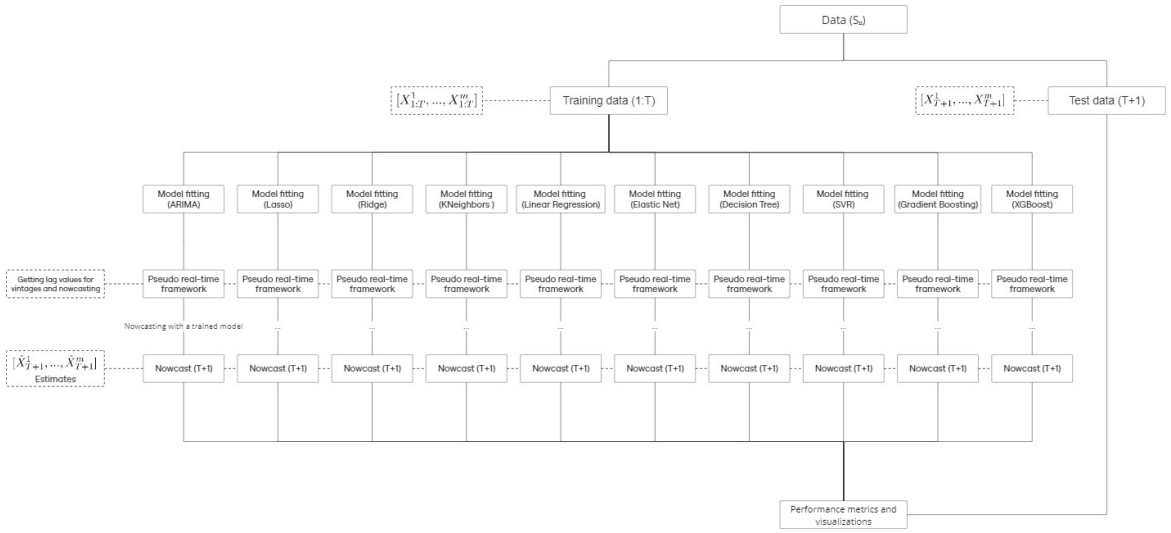
Also, it is crucial to mention, that there are various types of wavelets. Each of them has a different application approach, such as Haar, Daubechies, Coiflet, Symlet and others.[19][21] Practical implementation of MODWT (coif6) using GDP data is in 3 figure.



3 figure. *GDP decomposition of MODWT (coif6)*

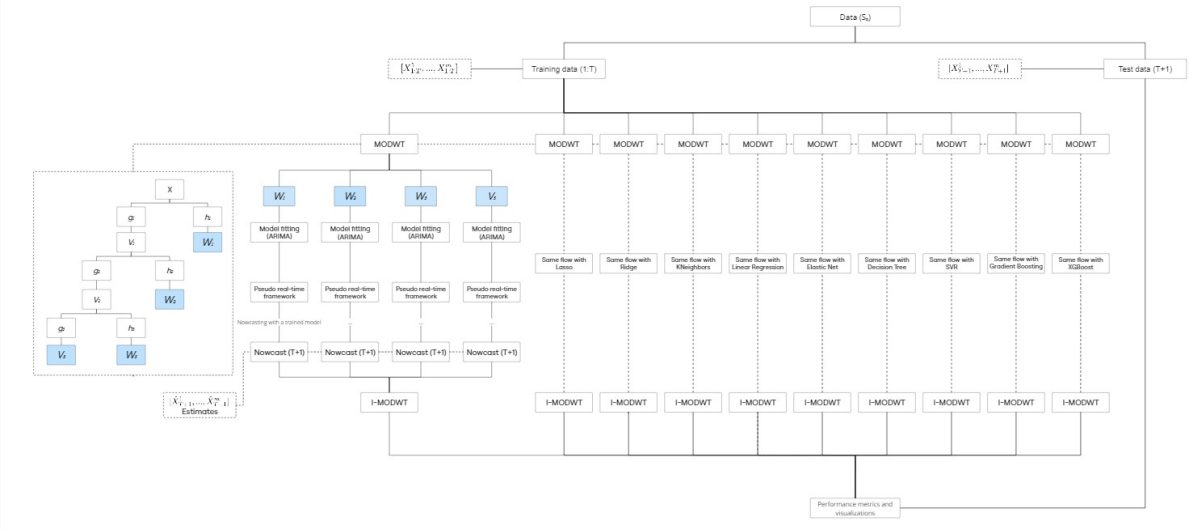
2.7 Final nowcasting algorithm

By summarizing all the methods used, we can create a scheme of the nowcasting system. This might help better understand the structure of the work and smooth operation. The aim of the research is to nowcast the Lithuanian GDP using machine learning methods, but also implement a novelty that could somehow improve these research outcomes. For this the implementation of MODWT was used in collaboration with the ARIMA model due to its effective applications observed in other areas, such as short-term wind speed forecasting [18] or daily snow depth forecasting [2]. In addition, there is practically no research on nowcasting GDP using MODWT-ARIMA. However, the entire system is designed to be easily used with both simple nowcasting, integrated with MODWT-ARIMA, and MODWT transformations, and in conjunction with all other available models. Firstly, the basic nowcasting framework is designed to automatically calculate all desired models and provide nowcasting results and visualizations in 4 figure.



4 figure. *Basic nowcasting framework*

In addition to that, the implementation of MODWT is added to the current scheme and now are expressed in 5 figure.



5 figure. *Nowcasting framework using MODWT*

In the next chapter, the entire logic will be tested and the outcomes analyzed using Lithuanian GDP data.

3 Results and Comparisons

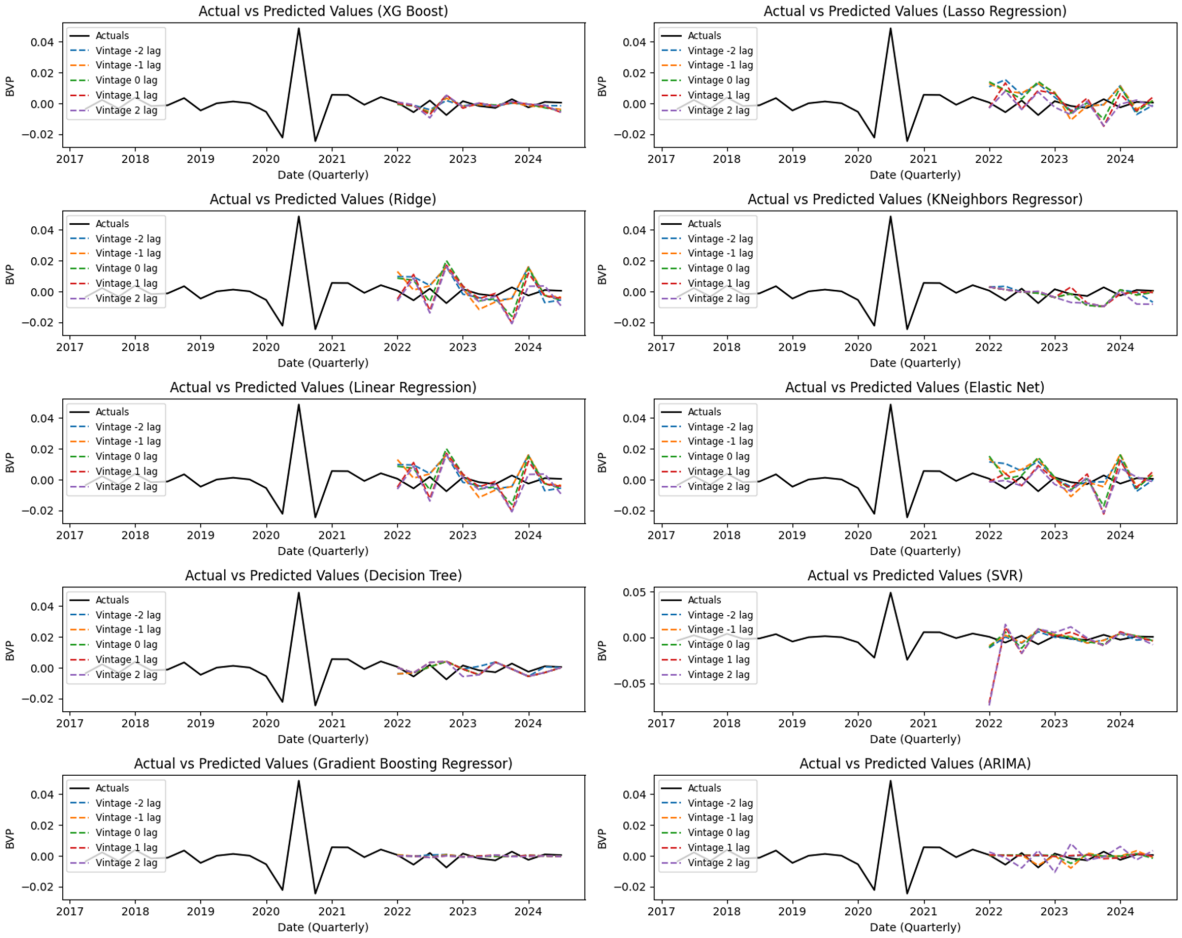
This chapter will close the research analysis with the results of nowcasting the Lithuanian GDP using machine learning approaches and implementations of the MODWT.

Firstly, the main algorithm for nowcasting was executed. Here it was also interesting to track the results through different data vintages, the vintages have a lag from variables schedule and created pseudo lag framework. Lags were five types: 2 months before the real-time, 1 month before the real-time, 0-month stands for real-time (nowcast) and it is also interesting to see the forecasts for the future, 1 month ahead of the real-time and 2 months ahead of the real-time. The pseudo-real-time framework results using Lithuanian GDP data are in 2 table. and practical visualizations are in 6 figure.

2 table. *Performance metrics across models for different vintage values*

Vintage	MAE	RMSE	Model
-2	0.0025	0.0034	ARIMA
-2	0.0026	0.0035	Gradient Boosting Regressor
-2	0.0032	0.0040	XG Boost
-2	0.0035	0.0047	Decision Tree
-2	0.0051	0.0062	KNeighbors Regressor
-2	0.0058	0.0071	SVR
-2	0.0077	0.0101	Elastic Net
-2	0.0082	0.0108	Lasso Regression
-2	0.0089	0.0113	Ridge
-2	0.0089	0.0113	Linear Regression
-1	0.0028	0.0034	ARIMA
-1	0.0028	0.0036	Gradient Boosting Regressor
-1	0.0032	0.0047	XG Boost
-1	0.0039	0.0049	Decision Tree
-1	0.0045	0.0056	KNeighbors Regressor
-1	0.0060	0.0073	SVR
-1	0.0083	0.0101	Lasso Regression
-1	0.0084	0.0108	Ridge
-1	0.0084	0.0108	Linear Regression
-1	0.0085	0.0110	Elastic Net
0	0.0028	0.0035	Gradient Boosting Regressor
0	0.0029	0.0037	ARIMA
0	0.0035	0.0047	Decision Tree
0	0.0042	0.0053	XG Boost
0	0.0045	0.0056	KNeighbors Regressor
0	0.0061	0.0073	SVR
0	0.0081	0.0109	Lasso Regression
0	0.0091	0.0120	Elastic Net
0	0.0102	0.0129	Ridge
0	0.0102	0.0129	Linear Regression

Vintage	MAE	RMSE	Model
1	0.0027	0.0035	ARIMA
1	0.0029	0.0035	Gradient Boosting Regressor
1	0.0035	0.0047	Decision Tree
1	0.0043	0.0057	XG Boost
1	0.0045	0.0056	KNeighbors Regressor
1	0.0061	0.0073	SVR
1	0.0086	0.0103	Lasso Regression
1	0.0088	0.0112	Elastic Net
1	0.0104	0.0132	Ridge
1	0.0104	0.0132	Linear Regression
2	0.0028	0.0035	Gradient Boosting Regressor
2	0.0040	0.0051	Decision Tree
2	0.0043	0.0059	XG Boost
2	0.0061	0.0073	ARIMA
2	0.0068	0.0087	Lasso Regression
2	0.0074	0.0099	Elastic Net
2	0.0098	0.0127	Ridge
2	0.0098	0.0127	Linear Regression
2	0.0161	0.0254	SVR



6 figure. *Visualization across models for different vintage values*

The results highlighted some insights into performance via different models tested across various vintages. The hyperparameters were selected from various possible variants using GridSearchCV. For each model, a set of different parameters was suggested and the GridSearchCV method selected the best ones, which had the lowest MAE metric. A TimeSeriesSplit was a cross-validator for this approach. Below (3 table.) is a table of the best hyperparameters for each model.

3 table. *Model parameters for hyperparameter tuning*

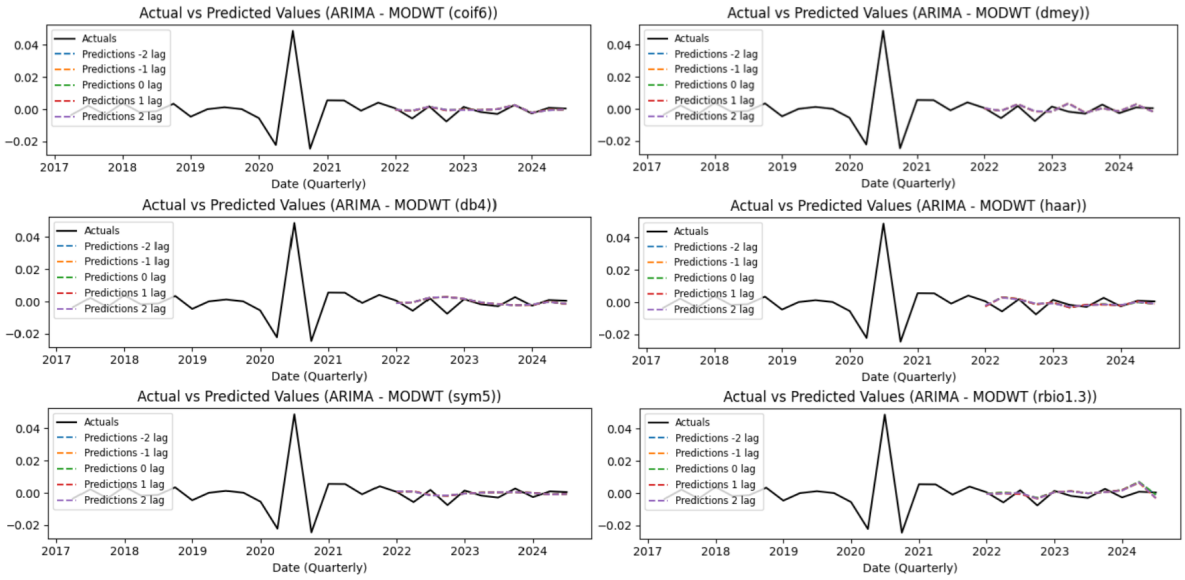
Model	**Parameters**
XG Boost	learning_rate: 0.01 max_depth: 6 n_estimators: 200 booster: gblinear
Lasso Regression	alpha: 100
Ridge	alpha: 100
KNeighbors Regressor	n_neighbors: 3 weights: uniform p: 1
Linear Regression	No parameters
Elastic Net	alpha: 100 l1_ratio: 0.5
Decision Tree	max_depth: 3 min_samples_split: 3 min_samples_leaf: 3 max_features: log2 criterion: absolute_error splitter: best
SVR	C: 1 epsilon: 0.01 kernel: linear gamma: scale degree: 2 shrinking: True tol: 1e-4
Gradient Boosting Regressor	learning_rate: 0.01 n_estimators: 25 max_depth: 3 alpha: 0.005 loss: squared_error
ARIMA	seasonal: False stepwise: True suppress_warnings: True

Gradient Boosting Regressor and ARIMA had the best performance, with the lowest errors (MAE and RMSE) in nearly all cases. Gradient Boosting Regressor slightly outperformed ARIMA, especially in vintage 0, which is most interesting in our case, making it the most reliable model

overall. Ensemble models like XG Boost, Decision Tree, and KNeighbors Regressor provide moderate performance but fail to match the accuracy of Gradient Boosting and ARIMA. XG Boost performs well for older vintages, while Decision Tree returned better outcomes and selected the best ones in nowcast and future vintages. Linear models, including Lasso Regression, Ridge, and Elastic Net, fall behind ensemble methods, indicating limitations in capturing non-linear patterns. From linear-based models, Elastic Net and Lasso performed better than other linear models but underperformed with Gradient Boosting. SVR and linear regression were the least effective models, particularly for newer vintages, the errors were significantly high. So, Gradient Boosting Regressor and ARIMA models are the most reliable choices for predicting and nowcasting Lithuanian GDP. However, linear methods and SVR should be avoided unless computational simplicity is a priority because ensemble approaches require longer calculation time and higher computational costs. Focusing only on the nowcasting part, where the novelty part about MODWT is implemented, we have quite different results. Firstly, the MODWT-ARIMA integration has been observed and results are in 4 table., visualizations are in 7 figure.

4 table. *Performance metrics of MODWT-ARIMA through different wavelet types*

Model	MAE	RMSE
MODWT-ARIMA (db4)	0.0026	0.0039
Direct ARIMA	0.0029	0.0037
MODWT-ARIMA (haar)	0.0027	0.0037
MODWT-ARIMA (coif6)	0.0020	0.0028
MODWT-ARIMA (sym5)	0.0028	0.0033
MODWT-ARIMA (dmey)	0.0026	0.0031
MODWT-ARIMA (rbio1.3)	0.0027	0.0037



7 figure. *Visualization of MODWT-ARIMA through different wavelet types*

The output claims that MODWT-ARIMA (coif6) achieves the best results with the lowest MAE

and RMSE. This indicates that the implementation of MODWT improves ARIMA's predictive accuracy compared to the direct approach. The wavelet type coif6 (Coiflet) showed the best results in contrast with other wavelet types (dmey, db4, hear, sym5, rbio1.3), demonstrating the importance of selecting the appropriate wavelet transform. Of course, there are many different types of wavelets, with different parameters and variables, that can further improve the model errors, but these types were chosen based on a literature review.[18][2] Overall, all different wavelets showed better results than the direct ARIMA model, but the coif6 transform highlighted a much better positive impact. The MODWT decomposed the time series into different frequency components, so high-frequency noise and trends can be treated separately and ARIMA can be applied to these individual components. This approach of multiple decompositions allowed for more accurate nowcasting of each series, rather than forcing the direct ARIMA model to deal with the entire series at once.

On the other hand, the combinations of MODWT with other used models also had some positive improvements in 5 table.

5 table. *Performance metrics of different models and MODWT types.*

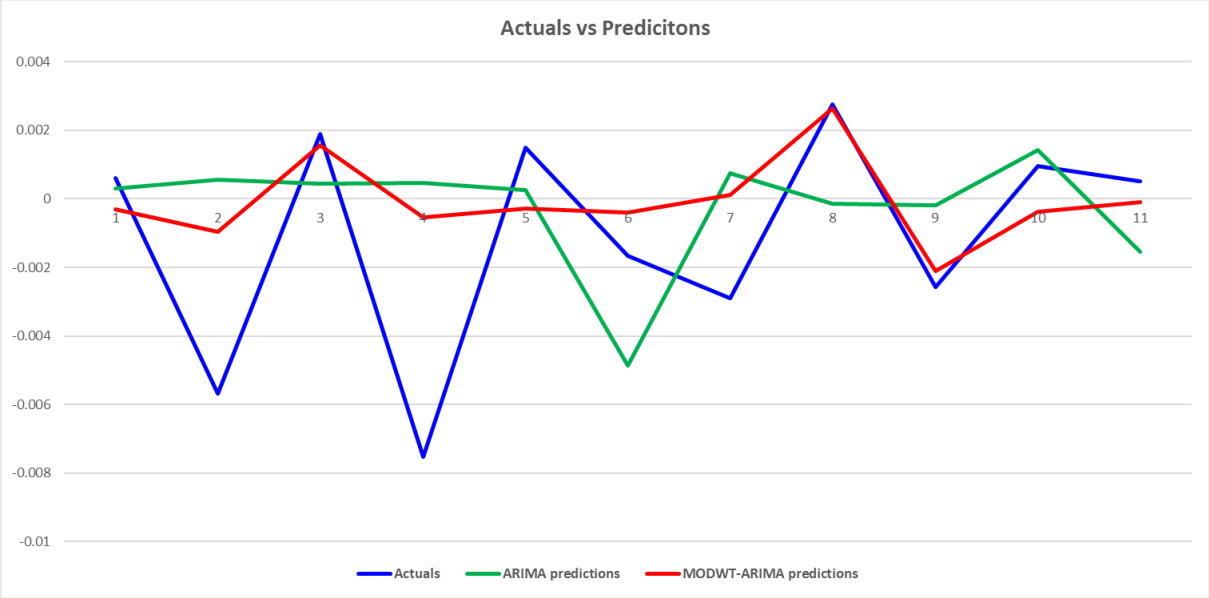
Model	MODWT type	MAE	RMSE
XGBoost	Direct	0.0042	0.0053
	coif6	0.0032	0.0040
	dmey	0.0045	0.0059
	db4	0.0036	0.0040
	haar	0.0045	0.0058
	sym5	0.0030	0.0043
	rbio1.3	0.0036	0.0044
Lasso Regression	Direct	0.0088	0.0105
	coif6	0.0137	0.0159
	dmey	0.0118	0.0143
	db4	0.0249	0.0286
	haar	0.0140	0.0204
	sym5	0.0164	0.0197
	rbio1.3	0.0140	0.0191
Ridge	Direct	0.0102	0.0125
	coif6	0.0042	0.0054
	dmey	0.0099	0.0124
	db4	0.0187	0.0281
	haar	0.0241	0.0319
	sym5	0.0115	0.0144
	rbio1.3	0.0173	0.0320
KNeighbors Regressor	Direct	0.0045	0.0056
	coif6	0.0027	0.0034
	dmey	0.0026	0.0034
	db4	0.0027	0.0034

Model	MODWT type	MAE	RMSE
	haar	0.0026	0.0035
	sym5	0.0026	0.0034
	rbio1.3	0.0026	0.0034
Linear Regression	Direct	0.0102	0.0125
	coif6	0.0657	0.1626
	dmey	0.0156	0.0215
	db4	0.0204	0.0345
	haar	0.0238	0.0314
	sym5	0.0131	0.0159
	rbio1.3	0.0140	0.0303
Elastic Net	Direct	0.0091	0.0129
	coif6	0.0157	0.0186
	dmey	0.0208	0.0232
	db4	0.0200	0.0225
	haar	0.0208	0.0291
	sym5	0.0139	0.0165
	rbio1.3	0.0148	0.0200
Decision Tree	Direct	0.0035	0.0047
	coif6	0.0069	0.0084
	dmey	0.0053	0.0069
	db4	0.0034	0.0043
	haar	0.0045	0.0060
	sym5	0.0029	0.0037
	rbio1.3	0.0037	0.0044
SVR	Direct	0.0073	0.0092
	coif6	0.0051	0.0060
	dmey	0.0049	0.0059
	db4	0.0095	0.0097
	haar	0.0049	0.0058
	sym5	0.0041	0.0046
	rbio1.3	0.0077	0.0094
Gradient Boosting Regressor	Direct	0.0028	0.0035
	coif6	0.0024	0.0033
	dmey	0.0028	0.0035
	db4	0.0025	0.0032
	haar	0.0024	0.0033
	sym5	0.0027	0.0034
	rbio1.3	0.0026	0.0033

After this kind of testing MODWT integration showed more positive results for ensemble-based models compared to linear methods. Huge improvements were especially accomplished with the MODWT-XGBoost(coif6), MODWT-KNeighbors Regressor(haar), and MODWT-Gradient Boosting Regressor(coif6) models. The MODWT-KNeighbors Regressor(haar) model almost outperformed MODWT-Gradient Boosting Regressor(coif6) in terms of MAE and RMSE, but compared MODWT-Gradient Boosting Regressor(coif6) with MODWT-ARIMA(coif6) it showed

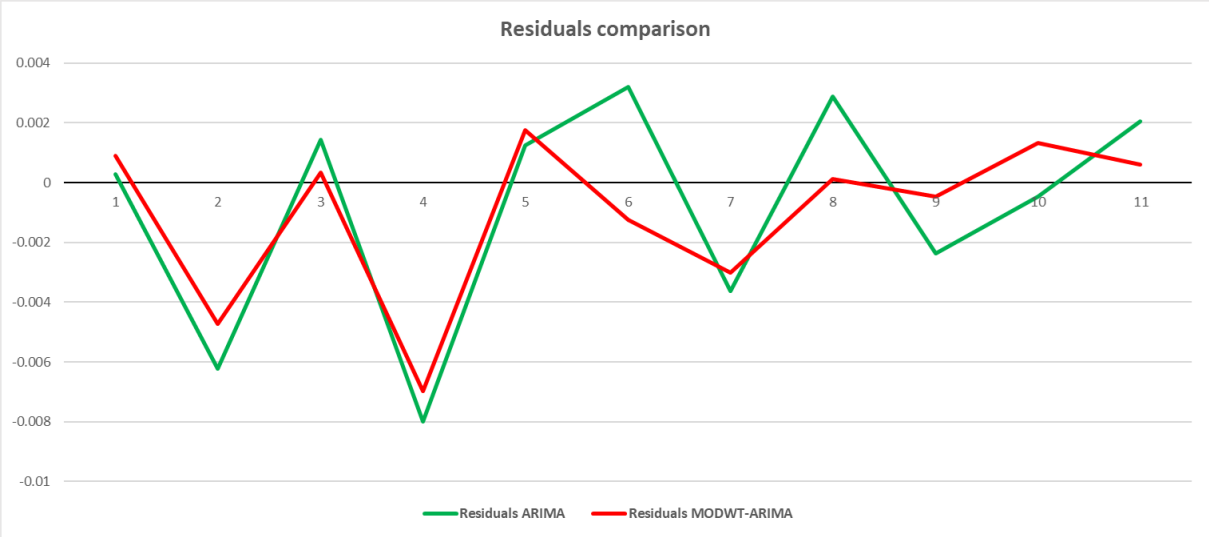
poorer results, which means that overall best model in this such case is MODWT-ARIMA(coif6) with 0.0020 MAE and 0,0028 RMSE discovered most accurate approach.

However, a natural question arises: Why are these errors of MODWT-ARIMA better than using the model ARIMA directly? Firstly, the numeric side of the outputs was analyzed. Diebold-Mariano test was used to determine the significance between the models and which is better. Also, the visualization (8 figure.) of actuals and both predictions were observed.



8 figure. *Visualization of Actual values, ARIMA nowcast and MODWT-ARIMA nowcast*

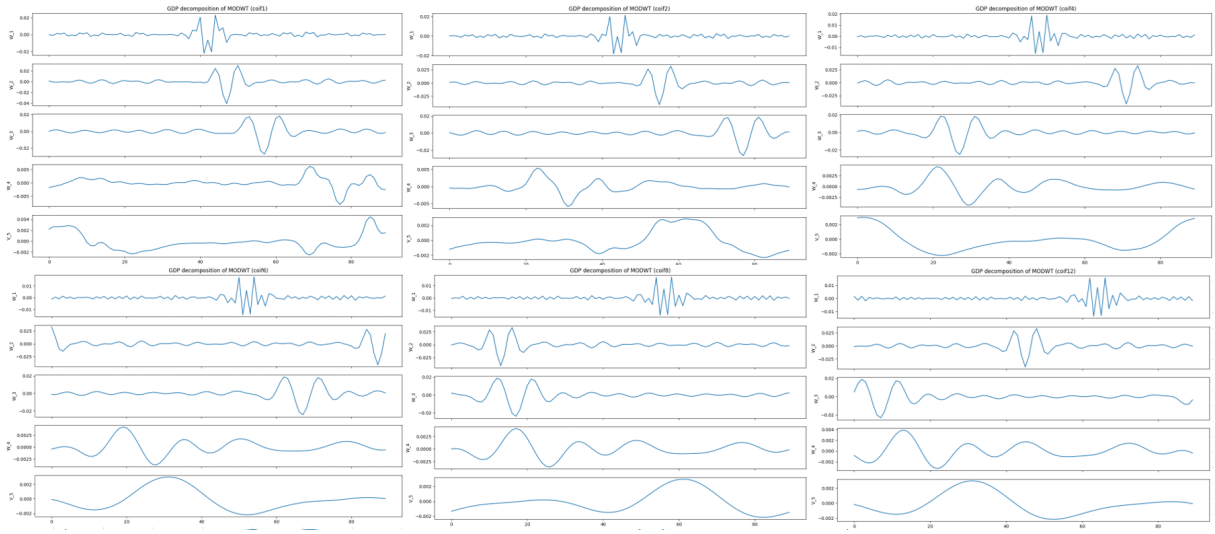
After comparing predictions using the Diebold-Mariano test, it returned a p-value of 0.0153, indicating a significant difference between the predictions and the outputs of MODWT-ARIMA are considerably better and the best overall of all models tested. The visualization (9 figure.) of prediction residuals also confirmed the following.



9 figure. *Visualization of ARIMA nowcast and MODWT-ARIMA nowcast residuals*

During the over-time, nowcasts of the MODWT-ARIMA model cope better with actual value fluctuations due to MODWT decompositions which reduce noise and improve model performance.

Secondly, the better MODWT-ARIMA model performance is due to MODWT decomposition. Coiflet wavelets were selected as the best wavelet type for this research. Due to this, analysis of different Coiflet levels was done as well. Here, it was analyzed each Coiflet wavelet through different levels (coif1, coif2, coif4, coif6, coif8, coif12). The main goal of this was to find why the coif6 wavelet had the best nowcasting accuracy and to see any patterns of this. Below (10 figure.) is the representation of Coiflet wavelets decompositions.



10 figure. *Visualization of different Coiflet wavelets on GDP series*

From the visualization coif6 wavelet (bottom left) has smoother and more consistent details and approximation, this smoothness helps to perform better compared with other wavelets. An approximation decomposition shows a clear and steady trend, it helps the ARIMA model to capture more accurate forecasts. Another interesting insight is that every wavelet signal shifts to the right as the decomposition level increases. From the performance metrics (6 table.), the top three wavelet types (coif6, coif4, and coif3) exhibit very similar distributions in their first three components. In these wavelets, the signals are more concentrated toward the right side, and the remaining components share a similar representation. In contrast, wavelets like coif1, coif8, and coif10 show different signal distributions, which correlates with their poorer performance. Also, additional modelling with different wavelet types was done as well, below (6 table.) is a results table

6 table. *MODWT-ARIMA performance metrics using different Coiflet wavelet types*

Model	MAE	RMSE
ARIMA-MODWT (coif1)	0.0029	0.0035
ARIMA-MODWT (coif2)	0.0033	0.0038
ARIMA-MODWT (coif4)	0.0029	0.0039
ARIMA-MODWT (coif6)	0.0020	0.0028
ARIMA-MODWT (coif8)	2.25E+18	7.38E+13
ARIMA-MODWT (coif10)	1.14E+58	3.80E+58

results showed that higher-level Coiflet wavelets led to massive errors. Coif6 had a good balance between capturing high and low frequencies, while coif8 and coif10 overfit with unnecessary details and coif1, coif2, and coif4 returned quite higher performance errors. All in all, as in the research examined in the literature, in this study, the coif6 wavelet type was the most effective method and had the best metrics, which allows for improvement in the accuracy of the model by a significant difference.[18][2]

4 Conclusion

This thesis aimed to develop a model for nowcasting Lithuania's GDP using machine learning approaches to provide more accurate and newest estimates. The research explored the potential of many machine learning methods to nowcast GDP growth and provide economic indicators from various regressors. The theory analysis of nowcasting showed that there are many great methods for calculating GDP under any circumstances. The DFMs have become a popular approach in nowcasting due to their ability to efficiently handle and extract information from large and different datasets, MIDAS can handle mixed-frequency data efficiently and be more simpler and powerful than DFM or even a combination of two separate methods like MIDAS-LASSO, which over time become more common applications due to its effective on a final product. Looking at today's trend, the machine learning approaches are the most popular and newest methods, many researchers of different countries have tried these methods on nowcasting GDP values and saw that the outcomes are more promising compared with older applications. These findings raised a motivation to create a nowcasting GDP framework using machine learning methods. Also, inspired additional thoughts and ideas about potential future research using combinations of statistical models such as DFM or MIDAS with various ML models and wavelet implementations.

The framework of nowcasting Lithuanian GDP was created with the possibility to have supported prediction vintages, which shows the GDP change during the time. The implemented pseudo-real-time feature also helped to have a more realistic picture of real-time scenarios. These improvements were introduced to different machine learning models. The outcomes analysis demonstrated that machine learning ensemble methods, particularly Gradient Boosting Regressor outperformed traditional linear models like Linear Regression or Elastic Net in terms of nowcast accuracy. The results revealed that this model was able to provide a more realistic estimate of GDP growth with smaller errors. Also, the ARIMA model was one of the best approaches in this case, the performance metrics were slightly better than the Gradient Boosting Regressor model.

The novelty of this research was introduced as the implementation of maximal overlapping discrete wavelet transform to the ARIMA model, but it was extended to other machine learning models as well. This showed how effective and well-adapted this framework is. The analysis of results after a combination of MODWT and any model highlighted that MODWT has a huge influence on ensemble models, the performance metrics were better for MODWT-XGBoost(coif6), MODWT-KNeighbors Regressor(haar), and MODWT-Gradient Boosting Regressor(coif6) models. However, the best performance improvement was for MODWT-ARIMA(coif6) application, this integration changed the MAE from direct ARIMA model MAE 0.0029 to an impressive -31 percent drop of 0.0020 MAE. The Diebold-Mariano test showed a significant change among the existing MAEs, confirming that the MODWT implementation made a significant change in the nowcast topic.

References and sources

[1] *2024 m. statistinės informacijos skelbimo kalendoriai*. Accessed: 2024-12-01.

[2] A. Adib, A. Zaerpour, M. Lotfirad. "On the reliability of a novel MODWT-based hybrid ARIMA-artificial intelligence approach to forecast daily Snow Depth (Case study: The western part of the Rocky Mountains in the U.S.A)." In: *Cold Regions Science and Technology* 189 (2021), page 103342. ISSN: 0165-232X. https://doi.org/https://doi.org/10.1016/j.coldregions.2021.103342. URL: https://www.sciencedirect.com/science/article/pii/S0165232X21001233.

[3] V. duomenų agentūra. *Verslas Lietuvoje (2022 m. leidimas)*. 2022. URL: https://osp.stat.gov.lt/statistikos-terminu-zodynas?popup=true&termId=9012 (viewed 2024-05-30).

[4] E. Andreou, E. Ghysels, A. Kourtellos. "Should Macroeconomic Forecasters Use Daily Financial Data and How?" In: *Journal of Business and Economic Statistics* 31.2 (2013), pages 240–251. ISSN: 07350015. URL: http://www.jstor.org/stable/43701607 (viewed 2024-06-04).

[5] M. T. Armesto, K. Engemann, M. Owyang. "Forecasting with mixed frequencies." In: *Review* 92.Nov (2010), pages 521–536. URL: https://EconPapers.repec.org/RePEc:fip:fedlrv:y:2010:i:nov:p:521-536:n:v.92no.6.

[6] J. Bai, S. Ng. "Determining the Number of Factors in Approximate Factor Models." In: *Econometrica* 70.1 (2002), pages 191–221. https://doi.org/https://doi.org/10.1111/1468-0262.00273. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/1468-0262.00273.

[7] J. Beyhum, J. Striaukas. *Sparse plus dense MIDAS regressions and nowcasting during the COVID pandemic*. Papers 2306.13362. arXiv.org, 2023. URL: https://ideas.repec.org/p/arx/papers/2306.13362.html.

[8] B. Bok, D. Caratelli, D. Giannone, A. M. Sbordone, A. Tambalotti. *Macroeconomic nowcasting and forecasting with big data*. Staff Reports 830. Federal Reserve Bank of New York, 2017. URL: https://ideas.repec.org/p/fip/fednsr/830.html.

[9] T. Chen, C. Guestrin. "XGBoost: A Scalable Tree Boosting System." In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. ACM, 2016, pages 785–794. https://doi.org/10.1145/2939672.2939785. URL: http://dx.doi.org/10.1145/2939672.2939785.

[10] E. (Commission). *Handbook on seasonal adjustment*. Publications Office of the European Union, 2018.

[11] A. Destrero, S. Mosci, C. Mol, A. Verri, F. Odone. “Feature selection for high-dimensional data.” In: *Computational Management Science* 6 (2009), pages 25–40. https://doi.org/10.1007/s10287-008-0070-7.

[12] E. C. S. O. of the European Union. *Euro area and European Union GDP flash estimates at 30 days: 2016 edition.* eng. LU: Publications Office, 2016. https://doi.org/10.2785/30494. URL: https://data.europa.eu/doi/10.2785/30494.

[13] C. Foroni, M. Marcellino. *A survey of econometric methods for mixed-frequency data*. Economics Working Papers ECO2013/02. European University Institute, 2013. URL: https://EconPapers.repec.org/RePEc:eui:euiwps:eco2013/02.

[14] J. Geweke. *The dynamic factor analysis of economic timeseries models*. Amsterdam, 1977. URL: https://www.econbiz.de/Record/the-dynamic-factor-analysis-of-economic-timeseries-models-geweke-john/10002419858.

[15] D. Giannone, L. Reichlin, M. Bańbura. *Nowcasting*. Working Paper Series 1275. European Central Bank, 2010. URL: https://ideas.repec.org/p/ecb/ecbwps/20101275.html.

[16] L. Han, Y. Liu, G. Li, L. Wen. “Tourism demand nowcasting using a LASSO-MIDAS model.” In: *International Journal of Contemporary Hospitality Management* 33 (2021), pages 1922–1949. https://doi.org/10.1108/IJCHM-06-2020-0589.

[17] D. Hopp. *Benchmarking econometric and machine learning methodologies in nowcasting GDP*. en. 2023. https://doi.org/10.1007/s00181-023-02515-6. URL: http://dx.doi.org/10.1007/s00181-023-02515-6.

[18] M. U. Yousuf, I. Al-Bahadly, E. Avci. “Short-Term Wind Speed Forecasting Based on Hybrid MODWT-ARIMA-Markov Model.” In: *IEEE Access* 9 (2021), pages 79695–79711.

[19] R. Islam. “Performance analysis of Coiflet-type wavelets for a fingerprint image compression by using wavelet and wavelet packet transform.” In: *International Journal of Computer Science and Engineering Survey* 3 (2012), pages 79–87. https://doi.org/10.5121/ijcses.2012.3209.

[20] V. N. Kuzin, M. Marcellino, C. Schumacher. *MIDAS versus mixed-frequency VAR: nowcasting GDP in the euro area*. Discussion Paper Series 1: Economic Studies 2009,07. Deutsche Bundesbank, 2009. URL: https://ideas.repec.org/p/zbw/bubdp1/7576.html.

[21] MathWorks. *Choose a Wavelet*. Accessed: 2025-01-03. 2025. URL: https://uk.mathworks.com/help/wavelet/gs/choose-a-wavelet.html.

[22] L. Melkumova, S. Shatskikh. “Comparing Ridge and LASSO estimators for data analysis.” In: *Procedia Engineering* 201 (2017). 3rd International Conference “Information Technology and Nanotechnology”, ITNT-2017, 25-27 April 2017, Samara, Russia, pages 746–755. ISSN: 1877-7058. https://doi.org/https://doi.org/10.1016/j.proeng.2017.09.615. URL: https://www.sciencedirect.com/science/article/pii/S1877705817341474.

[23] L. Reichlin, D. Giannone, D. Small. *Nowcasting GDP and Inflation: The Real Time Informational Content of Macroeconomic Data Releases*. CEPR Discussion Papers 5178. C.E.P.R. Discussion Papers, 2005. URL: https://ideas.repec.org/p/cpr/ceprdp/5178.html.

[24] A. Richardson, T. van Florenstein Mulder, T. Vehbi. “Nowcasting GDP using machine-learning algorithms: A real-time assessment.” In: *International Journal of Forecasting* 37.2 (2021), pages 941–948. ISSN: 0169-2070. https://doi.org/https://doi.org/10.1016/j.ijforecast.2020.10.005. URL: https://www.sciencedirect.com/science/article/pii/S016920702030159X.

[25] F. Schorfheide, D. Song. “Real-Time Forecasting With a Mixed-Frequency VAR.” In: *Journal of Business and Economic Statistics* 33.3 (2015), pages 366–380. https://doi.org/10.1080/07350015.2014.954707. URL: https://doi.org/10.1080/07350015.2014.954707.

[26] C. Schumacher, M. Marcellino, C. Foroni. *U-MIDAS: MIDAS regressions with unrestricted lag polynomials*. CEPR Discussion Papers 8828. C.E.P.R. Discussion Papers, 2012. URL: https://ideas.repec.org/p/cpr/ceprdp/8828.html.

[27] B. Soybilgen, E. Yazgan. “Nowcasting US GDP Using Tree-Based Ensemble Models and Dynamic Factors.” In: *Computational Economics* 57.1 (2021), pages 387–417. ISSN: 1572-9974. https://doi.org/10.1007/s10614-020-10083-5. URL: https://doi.org/10.1007/s10614-020-10083-5.

[28] J. Stock, M. Watson. “Dynamic Factor Models.” In: *Oxford Handbook on Economic Forecasting*. Edited by M. J. Clements, D. F. Hendry. Oxford: Oxford University Press, 2011.

[29] R. Tibshirani. “Regression shrinkage and selection via the lasso: a retrospective.” In: *Journal of the Royal Statistical Society. Series B (Statistical Methodology)* 73.3 (2011), pages 273–282. ISSN: 13697412, 14679868. URL: http://www.jstor.org/stable/41262671 (viewed 2025-01-03).

[30] K. H. Torsten Hothorn, A. Zeileis. “Unbiased Recursive Partitioning: A Conditional Inference Framework.” In: *Journal of Computational and Graphical Statistics* 15.3 (2006), pages 651–674. https://doi.org/10.1198/106186006X133933. URL: https://doi.org/10.1198/106186006X133933.

[31] V. Zarnowitz. “The New ASA–NBER Survey of Forecasts by Economic Statisticians.” In: *Supplement to NBER Report Four*. NBER Chapters. National Bureau of Economic Research, Inc, 1969, pages 1–8. URL: https://ideas.repec.org/h/nbr/nberch/9930.html.

[32] L. Zhu, Y. Wang, Q. Fan. “MODWT-ARMA model for time series prediction.” In: *Applied Mathematical Modelling* 38.5 (2014), pages 1859–1865. ISSN: 0307-904X. https://doi.org/https://doi.org/10.1016/j.apm.2013.10.002. URL: https://www.sciencedirect.com/science/article/pii/S0307904X13006148.

Appendix 1.

Here are all cited types of sources:

- web pages (*@online*) [1, 3, 21]
- articles (*@article*) [5, 6, 11, 16, 19, 22, 24, 27, 29, 30, 32]
- articles from conferences (*@inproceedings*) [9]
- books (*@book*) [10, 12, 28]
- electronic publications (*@misc*) [14, 17]
- technical reports (*@techreport*) [7, 8, 13, 15, 20, 23, 26]
- book chapters (*@incollection*) [31]

Appendix 2.

In the preparation of this thesis, several external tools were employed to enhance the quality and readability of the text. These tools were utilized to assist in refining language, improving vocabulary, and ensuring grammatical accuracy.

ChatGPT by OpenAI was used as a supportive tool for improving sentence structure and suggesting alternative word choices to enhance clarity and coherence. Grammarly was employed for proofreading and grammar correction. This tool was instrumental in identifying and addressing typographical errors, punctuation issues, and stylistic inconsistencies. It also assisted in maintaining a formal and professional tone throughout the text.

While these tools significantly aided in the writing process, all intellectual contributions, visualizations, research findings, and conclusions presented in this thesis are solely the result of the author's work.

Appendix 3.

Here is a Python code which was used in the practical experiment of the research.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import STL
from statsmodels.stats.diagnostic import het_breuschpagan
from arch.unitroot import PhillipsPerron
from statsmodels.api import OLS, add_constant
import statsmodels.api as sm
from itertools import combinations,permutations
from dm_test import dm_test
import warnings
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from modwt import modwt, modwtmra, imodwt
import matplotlib.pyplot as plt
from tabulate import tabulate
from sklearn.model_selection import GridSearchCV, TimeSeriesSplit
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.impute import SimpleImputer
from xgboost import XGBRegressor
from statsmodels.tsa.arima.model import ARIMA
from pmdarima import auto_arima
import inspect

warnings.filterwarnings("ignore")
plt.rcParams['figure.figsize'] = [15, 10]

# Data loading and definitions of test/train sample strat/end
data = pd.read_excel('BVP_men_SA.xlsx', sheet_name='Regresoriai_men', parse_dates=[0])
schedule = pd.read_excel('variable_schedule.xlsx', sheet_name='schedule_SA')
training_s  = "2017-01-01"
testing_s = "2022-01-01"
testing_e = "2024-09-01"
```

```
lags = [-2, -1, 0, 1, 2]

# List of functions which were used in further calculations.
# It consists column fill with column mean function, stationary tests, modwt
# transformations and lag steps generator function for each column.
def mean(training, test, date_column):
    training[date_column] = pd.to_datetime(training[date_column])
    test[date_column] = pd.to_datetime(test[date_column])
    date_last = training[date_column].max()
    last_3_months = training[training[date_column] >= (date_last -
    pd.DateOffset(months=3))]
    mean_dict = {}
    for c in training.columns[1:]:
        mean_dict[c] = np.nanmean(last_3_months[c])
    filled = test.copy()
    for c in training.columns[1:]:
        filled.loc[pd.isna(filled[c]), c] = mean_dict[c]
    return filled

def stationary_tests(data):
    def stationary_flow(data, columns):
        stat_c = []
        non_stat_c = []
        for c in columns:
            x_series = data[c].dropna()
            result_adf = adfuller(x_series, autolag='AIC')
            adf_p_value = result_adf[1]
            X = np.arange(len(x_series))
            X = sm.add_constant(X)
            y = x_series
            model = sm.OLS(y, X).fit()
            result_bp = het_breuschpagan(model.resid, model.model.exog)
            bp_p_value = result_bp[1]
            if bp_p_value < 0.05:
                result_pp = PhillipsPerron(x_series)
                pp_p_value = result_pp.pvalue
                if pp_p_value >= 0.05:
                    non_stat_c.append(c)
                else:
                    stat_c.append(c)
            else:
```

```
                if adf_p_value < 0.05:
                    stat_c.append(c)
                else:
                    non_stat_c.append(c)
        return stat_c, non_stat_c
    columns_wo_date_BVP = [column for column in data.columns
    if column !="date" and column != 'BVP_SA']
    stat_c_before, non_stat_c_before = stationary_flow(data, columns_wo_date_BVP)
    for c in non_stat_c_before:
        if (data[c] <= 0).any():
            data[c] = data[c].diff()
        else:
            data[c] = np.log(data[c]).diff()
    data.dropna(inplace=True)
    stat_c_after, non_stat_c_after = stationary_flow(data, non_stat_c_before)
    i = 1
    while non_stat_c_after:
        for c in non_stat_c_after:
            if (data[c] <= 0).any():
                data[c] = data[c].diff()
            else:
                data[c] = np.log(data[c]).diff()
        data.dropna(inplace=True)
        stat_c_after, non_stat_c_after = stationary_flow(data, non_stat_c_after)
        i += 1
    return data

def lag_data(schedule, data, date_last, lag):
    lag_data = data[data.date <= date_last].reset_index(drop=True)
    for c in lag_data.columns[1:]:
        if c in schedule['all_variables'].values:
            lag_ = schedule.loc[schedule.all_variables == c, "month_lag"].values[0]
        else:
            lag_ = 0
        lag_data.loc[(len(lag_data) - lag_ + lag) :, c] = np.nan
    return lag_data

def modwt_transform(data, wavelet, level, step):
    data_wavelet = pd.DataFrame(index=data.index)
    for c in data.columns:
        coeffs = modwt(data[c].values, wavelet, level)
```

```
        data_wavelet[f"{c}_wavelet_{step}"] = coeffs[step]
    return data_wavelet

# Data preparation steps for remaining nowcasting framework
data = stationary_tests(data)
data.set_index('date')
predicted = "BVP_SA"
nowcasting_data = data.loc[(data.date >= training_s) & (data.date <= testing_e), :]
.reset_index(drop=True)
dates = pd.date_range(start=testing_s, end=testing_e, freq="3MS").strftime("%Y-
%m-%d").tolist()
true_values = [value for value in nowcasting_data.loc[nowcasting_data['date']
.isin(dates), predicted].values]

# Nowcasting framework
# List of models with different parameters which were used in the nowcasting framework.
models = [
    {
        'name': 'XG Boost',
        'model': XGBRegressor(),
        'parameters': {
            'learning_rate': [0.01],
            'max_depth': [6],
            'n_estimators': [200],
            'booster': ["gblinear"],
        }
    },
    {
        'name': 'Lasso Regression',
        'model': Lasso(),
        'parameters': {
            'alpha': [0.0001, 0.01, 1, 100]
        }
    },
    {
        'name': 'Ridge',
        'model': Ridge(),
        'parameters': {
            'alpha': [0.0001,0.1, 1, 100]
        }
    },
```

```
{
    'name': 'KNeighbors Regressor',
    'model': KNeighborsRegressor(),
    'parameters': {
       'n_neighbors': [3],
        'weights': ['uniform'],
        'p': [1]
    }
},
{
    'name': 'Linear Regression',
    'model': LinearRegression(),
    'parameters': {
    }
},
{
    'name': 'Elastic Net',
    'model': ElasticNet(),
    'parameters': {
       'alpha': [0.0001,0.1, 1, 100],
        'l1_ratio': [0.1, 0.5]
    }
},
{
    'name': 'Decision Tree',
    'model': DecisionTreeRegressor(),
    'parameters': {
        'max_depth': [3],
        'min_samples_split': [3],
        'min_samples_leaf': [3],
        'max_features': ["log2"],
        'criterion': ["absolute_error"],
        'splitter': ["best"]
    }
},
{
    'name': 'SVR',
    'model': SVR(),
        'parameters':{
        'C': [1],
        'epsilon': [0.01],
```

```
            'kernel': ['linear'],
            'gamma': ['scale'],
            'degree': [2],
            'shrinking': [True],
            'tol': [1e-4]
        }
    },
    {
        'name': 'Gradient Boosting Regressor',
        'model': GradientBoostingRegressor(),
        'parameters' : {
            'learning_rate': [0.01],
            'n_estimators': [25],
            'max_depth': [3],
            'alpha': [0.005],
            'loss': ['squared_error']
        }
    },
    {
        'name': 'ARIMA',
        'model': "",
        'parameters': {
            'seasonal': [False],
            'stepwise': [True],
            'suppress_warnings': [True]
        }
    }
]

# Array for results
results = {
    'Model': [],
    'RMSE': [],
    'MAE': [],
}
fig, axes = plt.subplots(5, 2, figsize=(15, 12))
axes = axes.flatten()

# Nowcasting framework for each model. Nowcasting each point from the tests sample
# start point, outputs
# are compared with true values.
```

```
for i, model_info in enumerate(models):
    model_name = model_info['name']
    model = model_info['model']
    param_grid = model_info['parameters']
    # print(f"Running framework for {model_name}")
    pred_dict = {k: [] for k in lags}
    if model_name != "ARIMA":
        # Nowcasting for each point of the test sample. Each time model is created from
        scratch with
        # rolling variables respectively for loop iteration.
        for date in dates:
            training_data = nowcasting_data.loc[nowcasting_data.date <=
            str(pd.to_datetime(date) - pd.tseries.offsets.DateOffset(months=3))[:10],:]
            feature_engineering = mean(training_data, training_data, "date")
            feature_engineering = feature_engineering.loc[feature_engineering.date.dt
            .month.isin([1,4,7,10]),:].dropna(axis=0, how="any").reset_index(drop=True)
            x = feature_engineering.drop(["date", predicted], axis=1)
            y = feature_engineering[predicted]
            tscv = TimeSeriesSplit(n_splits=5)
            # Best model parameters are selected using the GridSearch function with
            time series split and scoring.
            grid_search = GridSearchCV(model, param_grid, cv=tscv,
            scoring='neg_mean_squared_error', error_score='raise')
            grid_search.fit(x, y)
            best_model = grid_search.best_estimator_
            # Pseudo-real-time creation with lags
            for lag in lags:
                lag_calculations = lag_data(schedule, nowcasting_data, date, lag)
                lag_calculations = mean(training_data, lag_calculations, "date")
                x = lag_calculations.loc[lag_calculations.date ==
                date, :].drop(["date", predicted], axis=1)

                # Nowcast
                prediction = best_model.predict(x)[0]
                pred_dict[lag].append(prediction)
    # ARIMA model has quite a different model structure, due to this this model has a
    # little bit different flow, but the main idea remains the same.
    elif model_name == 'ARIMA':
        for date in dates:
            training_data = nowcasting_data.loc[nowcasting_data.date <=
            str(pd.to_datetime(date) - pd.tseries.offsets.DateOffset(months=3))[:10], :]
```

```
        feature_engineering = mean(training_data, training_data, "date")
        feature_engineering = feature_engineering.loc[feature_engineering.date.dt
        .month.isin([1,4,7,10]), :].dropna(axis=0, how="any").reset_index(drop=True)
        for lag in lags:
            lag_calculations = lag_data(schedule, nowcasting_data, date, lag)
            lag_calculations = mean(training_data, lag_calculations, "date")
            y = lag_calculations[predicted]
            auto_model = auto_arima(
                y,
                seasonal=param_grid['seasonal'][0],
                stepwise=param_grid['stepwise'][0],
                suppress_warnings=param_grid['suppress_warnings'][0]
            )
            best_order = auto_model.order
            best_model = ARIMA(y, order=best_order).fit()
            prediction = best_model.forecast(steps=1)
            pred_dict[lag].append(prediction)
performance = pd.DataFrame(columns=["Vintage", "RMSE","MAE"])
# Results and accuracy metrics are stored for each vintage
for lag in lags:
    x = pd.DataFrame({
    "Vintage": lag,
    "RMSE": np.sqrt(mean_squared_error(true_values, pred_dict[lag])),
    "MAE": mean_absolute_error(true_values, pred_dict[lag]),
    "Model": model_name
    }, index=[0])
    performance = pd.concat([performance, x]).reset_index(drop=True)
print(performance.round(4))
quarterly_dates = data['date'][data['date'].dt.month.isin([1, 4, 7, 10])]
test_dates = quarterly_dates[(quarterly_dates >= testing_s) &
(quarterly_dates <= testing_e)]
true_values_all = list(data.loc[data.date.isin(quarterly_dates), predicted].values)

# Data visualizations
axes[i].plot(quarterly_dates, true_values_all, label='Actuals', color='black')
for lag, predictions in pred_dict.items():
    if lag in lags:
        axes[i].plot(test_dates, predictions, label=f'Predictions {lag} lag',
        linestyle='--')
axes[i].set_title(f'Actual vs Predicted Values ({model_name} - MODWT ({wavelet}))')
axes[i].set_xlabel("Date (Quarterly)")
```

```
    axes[i].set_ylabel("BVP")
    axes[i].legend(loc='upper left', fontsize='small')
plt.tight_layout()
plt.show()

# Nowcasting framework with MODWT
# This framework is implemented on the basic nowcasting framework.
# So the main ideas are the same as it was.
# The new things will be highlighted.
results = {
    'Model': [],
    'RMSE': [],
    'MAE': [],
}

fig, axes = plt.subplots(5, 2, figsize=(15, 12))
axes = axes.flatten()
# Wavelet steps
level = 4
wavelet_dict = {}

# Here the wavelet type must be selected.
List of the wavelets which were used in this work:
# db4, haar, sym5, dmey, rbio1.3, coif6
wavelet='coif6'

# Nowcasting framework is used to calculate any model interaction with MODWT features.
for i, model_info in enumerate(models):
    model_name = model_info['name']
    model = model_info['model']
    param_grid = model_info['parameters']
    # print(f"Running algorithm for {model_name}")
    pred_dict = {k: [] for k in lags}
    pred_dict_w = {k: [] for k in lags}
    if model_name != "ARIMA":

        # New for loop is used to calculate each level of wavelet
        for j in range(0, level+1):
            for date in dates:
                training_data = nowcasting_data.loc[nowcasting_data.date <=
                str(pd.to_datetime(date) - pd.tseries.offsets
```

```
            .DateOffset(months=3))[:10],:]
            feature_engineering = mean(training_data, training_data, "date")
            feature_engineering = feature_engineering.loc[feature_engineering.date
            .dt.month.isin([1,4,7,10]),:]
            .dropna(axis=0, how="any").reset_index(drop=True)
            x = feature_engineering.drop(["date", predicted], axis=1)

            # Wavelet transformation for x and y
            x = modwt_transform(x, wavelet=wavelet, level=level, step=j)
            y = modwt_transform(feature_engineering[predicted].to_frame(),
            wavelet=wavelet, level=level, step=j)
            tscv = TimeSeriesSplit(n_splits=5)
            grid_search = GridSearchCV(model, param_grid, cv=tscv,
            scoring='neg_mean_squared_error', error_score='raise')
            grid_search.fit(x, y)
            best_model = grid_search.best_estimator_
            # print(f'Running wavelet {j} for {date}')
            for lag in lags:
                lag_calculations = lag_data(schedule, nowcasting_data, date, lag)
                lag_calculations = mean(training_data, lag_calculations, "date")
                lag_calculations_wo_date = lag_calculations.drop(["date",
                predicted], axis=1)
                lag_calculations_modwt = modwt_transform(lag_calculations_wo_date,
                wavelet=wavelet, level=level, step=j)
                lag_calculations_new = pd.concat([lag_calculations_modwt,
                lag_calculations["date"]], axis=1)
                x = lag_calculations_new.loc
                [lag_calculations_new.date == date, :].drop(["date"], axis=1)

                # Nowcasting every wavelet level step, every vintage at
                every test sample point.
                prediction = best_model.predict(x)[0]
                pred_dict[lag].append(prediction)
        wavelet_dict[j] = pred_dict.copy()
        pred_dict = {k: [] for k in lags}

# This logic is very complicated because there are 3 different loops.
# Every loop step should be distributed according to their places.
lag_dicts = {lag: [] for lag in lags}
for sub_dict in wavelet_dict.values():
    for lag, value in sub_dict.items():
```

```
            if lag in lag_dicts:
                lag_dicts[lag].append(value)
    for lag, lag_values in lag_dicts.items():

        # When distribution is done. The transformation from wavelet
        signal prediction to the
        # time series is done using imodwt.
        # Inverse Maximal Overlap Discrete Wavelet Transform  is used in
        wavelet analysis to
        # reconstruct a signal from its wavelet coefficients that were
        generated using the MODWT.
        pred_w = imodwt(lag_values, wavelet)
        pred_dict_w[lag].append(pred_w)

# The same flow is done using the ARIMA model.
elif model_name == 'ARIMA':
    for j in range(0, level+1):
        for date in dates:
            training_data = nowcasting_data.loc[nowcasting_data.date <=
            str(pd.to_datetime(date) - pd.tseries.offsets
            .DateOffset(months=3))[:10], :]
            feature_engineering = mean(training_data, training_data, "date")
            feature_engineering = feature_engineering.loc[feature_engineering.date.dt
            .month.isin([1,4,7,10]), :]
            .dropna(axis=0, how="any").reset_index(drop=True)
            # print(f'Running wavelet {j} for {date}')
            for lag in lags:
                lag_calculations = lag_data(schedule, nowcasting_data, date, lag)
                lag_calculations = mean(training_data, lag_calculations, "date")
                y = modwt_transform(lag_calculations[predicted].to_frame(),
                wavelet=wavelet, level=level, step=j)
                auto_model = auto_arima(
                    y,
                    seasonal=param_grid['seasonal'][0],
                    stepwise=param_grid['stepwise'][0],
                    suppress_warnings=param_grid['suppress_warnings'][0]
                )
                best_order = auto_model.order
                best_model = ARIMA(y, order=best_order).fit()
                prediction = best_model.forecast(steps=1)
                pred_dict[lag].append(prediction)
```

```
            wavelet_dict[j] = pred_dict.copy()
            pred_dict = {k: [] for k in lags}
        lag_dicts = {lag: [] for lag in lags}
        for sub_dict in wavelet_dict.values():
            for lag, value in sub_dict.items():
                if lag in lag_dicts:
                    lag_dicts[lag].append(value)
        for lag, lag_values in lag_dicts.items():
            pred_w = imodwt(lag_values, wavelet)
            pred_dict_w[lag].append(pred_w)
    performance = pd.DataFrame(columns=["Vintage", "RMSE","MAE"])

    # Performance metrics for each vintage.
    for lag in lags:
        x = pd.DataFrame({
        "Vintage": lag,
        "RMSE": np.sqrt(mean_squared_error(true_values, pred_dict_w[lag][0])),
        "MAE": mean_absolute_error(true_values, pred_dict_w[lag][0]),
        "Model": model_name
        }, index=[0])
        performance = pd.concat([performance, x]).reset_index(drop=True)
    print(performance.round(4))
    quarterly_dates = data['date'][data['date'].dt.month.isin([1, 4, 7, 10])]
    test_dates = quarterly_dates[(quarterly_dates >= testing_s) &
    (quarterly_dates <= testing_e)]
    true_values_all = list(data.loc[data.date.isin(quarterly_dates), predicted].values)

    # Data visualizations.
    axes[i].plot(quarterly_dates, true_values_all, label='Actuals', color='black')
    for lag, predictions in pred_dict_w.items():
        if lag in lags:
            axes[i].plot(test_dates, predictions[0],
            label=f'Predictions {lag} lag', linestyle='--')
    axes[i].set_title(f'Actual vs Predicted Values ({model_name} - MODWT ({wavelet}))')
    axes[i].set_xlabel("Date (Quarterly)")
    axes[i].set_ylabel("BVP")
    axes[i].legend(loc='upper left', fontsize='small')
plt.tight_layout()
plt.show()

# The best model implementation of MODWT-ARIMA is compared with the direct
```

```
ARIMA approach to find
# if there is a significant change in the accuracy (MAE) metrics

true_values = np.array([0.000602368, -0.005682667, 0.001891908,
-0.007530638, 0.001485916, -0.001654852, -0.002894649, 0.002746955,
-0.002570982, 0.000951662, 0.000512185])
direct_arima_predictions = np.array([0.000311116, 0.000552692,
0.00045351, 0.000470809, 0.000246758, -0.004863303,
0.000741207, -0.00014329, -0.0001959, 0.001428993, -0.001551229])
modwt_arima_predictions = np.array([-0.000306175, -0.000949065,
0.001557645, -0.000543094, -0.000281345, -0.000404364, 0.000105832,
0.002637581, -0.002099504, -0.000373153, -0.0001])
db_mse = dm_test(actuals, direct_arima_predictions,
modwt_arima_predictions, h = 1, crit="MSE")
pvalue = db_mse[1]

# P-value is 0.015285942010315922, which indicates
a significant difference between the errors.
```

Here is an R code of seasonal adjustments using the X-13 ARIMA SEATS method:

```
library(readxl)
library(writexl)
library(dplyr)
library(seasonal)
library(x13binary)

file_path <- "C:/Users/admin/Desktop/Magistras/BVP.xlsx"
sheet_name <- "Regresoriai_men"

data <- read_excel(file_path, sheet = sheet_name)

data <- data %>%
  mutate(across(where(is.numeric), ~ ifelse(is.na(.), mean(., na.rm = TRUE), .)))

data$date <- as.Date(data$date)

adjusted_data_list <- list()

for (col_name in names(data)[-1]) {
```

```
  ts_data <- ts(data[[col_name]], frequency = 4, start = c(2010, 1))
  seas_adjustment <- seas(ts_data)
  adjusted_values <- final(seas_adjustment)
  adjusted_data_list[[col_name]] <- adjusted_values
}

adjusted_data <- data.frame(date = data$date)

for (col_name in names(adjusted_data_list)) {
  adjusted_data[[paste0(col_name, "_SA")]] <- adjusted_data_list[[col_name]]
}

output_file <- "C:/Users/admin/Desktop/Magistras/BVP_men_SA.xlsx"
write_xlsx(adjusted_data, output_file)
```