Graduate Institute of International and Development Studies
International Economics Department
Working Paper Series

Working Paper No. HEIDWP05-2025

Forecasting Banking System Liquidity Using Payment System Data in Uzbekistan

Shakhzod Abdullaevich Makhmudov
The Central Bank of Uzbekistan

Chemin Eugène-Rigot 2
P.O. Box 136
CH - 1211 Geneva 21
Switzerland

Bilateral Assistance
& Capacity Building
for Central Banks

Forecasting Banking System Liquidity Using Payment System Data in Uzbekistan

Shakhzod Abdullaevich Makhmudov
(maxmudovshahzod@gmail.com)
The Central Bank of Uzbekistan

Abstract

Forecasting banking system liquidity is crucial for the effective monetary policy implementation. This study investigates the effectiveness of various econometric and machine learning models in predicting the autonomous factors of banking system liquidity. The research compares widely used econometric models such as SARIMA, Exponential Smoothing, and Prophet alongside machine learning models like Random Forest, applying various preprocessing techniques, including power transformations, scaling, and trend-cycle decomposition. Moreover, ensemble methods, like weighted blending and stacking, were used to improve accuracy. Experimental results indicate that SARIMA was the best individual model, but ensemble with Prophet and Random Forest further improved forecast performance. Neural network models underperformed potentially due to challenges in optimizing their architectures. Future research intends to explore multivariate and structural models, as well as advanced neural architectures, to enhance predictive accuracy.

Keywords: Monetary Policy, Time-Series Models, Model Evaluation and Selection, Forecasting and Other Model Applications, Payment Systems
JEL: E52, C22, C52, C53, E42

The author thanks Jean-Charles Rochet from the University of Toulouse and Davide La Vecchia from the University of Geneva for the academic supervision of this paper. This research took place through the visiting program under the Bilateral Assistance and Capacity Building for Central Banks (BCC), financed by SECO, and the Graduate Institute in Geneva. The views expressed in this paper are solely those of the author and do not necessarily reflect those of the Central Bank of Uzbekistan.

Contents

1 Introduction

The Central Bank of Uzbekistan (CBU) has made significant improvements in revising its monetary policy framework with changing its monetary policy regime into Inflation Targeting (IT) since early 2018, prioritizing price stability and setting an inflation target of 5 percent annually.

Managing the banking system liquidity and interbank lending interest rates are of utmost importance for a central bank with a mandate of price stability since these are the direct ways of translating the central bank's decisions into the economy. The main instrument used for determining the monetary operations' rates is the policy rate of the CBU, which is set by the Managing Board of the Central Bank eight times per year. By conducting monetary operations, the central bank seeks to achieve its operational target — ensuring the formation of money market interest rates on overnight deposits within an interest rate corridor close to the key rate of the Central Bank, which is expected to assist in achieving the main target — inflation.

The research aims to examine a variety of econometric and machine learning models for time series analysis to forecast autonomous factors affecting banking system liquidity using large payment system data. The objective is to identify the most effective models and utilize them to predict the overall liquidity of the banking system. This research objective is both scientifically significant and policy-relevant, addressing a critical need within the central bank's monetary operations framework.

Let us clarify the usage of the term *banking system liquidity.* The CBU, as a bank of banks, provides banking services to commercial banks (the only type of depository institution legally allowed to operate in Uzbekistan), one of which is opening correspondent accounts for the commercial banks to facilitate the payment system.

It is worth mentioning that the term *liquidity* can have multiple meanings in different contexts of economics and finance. For example, the liquidity of an asset refers to the speed (ease) with which it can be converted into cash without reducing its market value. Additionally, there are liquidity requirements stipulated by the Basel Committee on Banking Supervision and subsequently adhered to by the CBU, which set certain ratios for commercial banks to ensure they are always prepared to fulfill payment obligations to their customers promptly. However, in the context of monetary policy and hence in this research, *liquidity* refers exclusively to the total amount of the balance stock of correspondent accounts of commercial banks in the central bank.

What are liqudity factors? When banks carry out transactions on behalf of their clients or themselves, the stock of their individual correspondent accounts changes accordingly. When the transaction is between two commercial banks, their accounts change, but the overall liquidity is not affected since it is just the sum of all the stocks of individual banks' accounts. The other case is when one of the banks is the central bank. In this case, when the central bank conducts financial operations with a commercial bank, it simply adds or subtracts funds from the commercial bank's account, changing the total amount of liquidity in the banking system. For example, when the central bank extends a loan to a commercial bank, it increases the amount of "Extended loans to commercial banks" on the asset side of its balance sheet and also increases the stock of "Corresponding accounts of commercial banks" on the liabilities side. Thus, the operations only with the central bank affect the banking system liquidity. In the context of monetary operations, the focus is on forecasting the factors that affect the total amount of liquidity, not the individual stock of each bank's correspondent account, as that might be of lesser importance for fine-tuning monetary operations.

The operations, or in other words, factors, affecting the total amount of liquidity can be divided into two broad groups: autonomous factors—operations that are beyond the direct impact of the central bank; and non-autonomous factors—transactions that the central bank conducts on its own initiative and thus can alter as it pleases.

Autonomous factors, such as currency in circulation, government operations, and net foreign assets (purchase of monetary gold by the central bank from the local producers, as well as, foreign exchange interventions), play a crucial role in determining the overall liquidity in the banking system. The ability to accurately forecast these factors is essential for central banks to implement precise and timely monetary interventions through open market operations.

So, this work tries to find the answer for research question of "How can econometric and machine learning models be effectively employed in time series analysis to forecast autonomous factors affecting banking system liquidity, and which models most accurately predict the overall liquidity of the banking system using large payment system data?"

The study aims to bridge the gap between traditional time series methodologies, such as SARIMA and Exponential Smoothing, and modern machine learning techniques like Random Forest and artificial neural networks. By leveraging these advanced models, we aim to uncover patterns and trends in autonomous factors that were previously challenging to identify.

A comprehensive framework is proposed to evaluate the performance of these models based on key forecasting metrics, including Mean Absolute Percentage Error (MAPE), R-squared, Theil's U, and a composite score. The analysis incorporates feature engineering techniques to enhance model performance, including the extraction of time-based features, seasonal decomposition, and treatment of heteroskedasticity.

In addition, this research explores ensembling techniques such as blending and stacking to combine the strengths of individual models, aiming to achieve a robust and reliable forecasting framework. By addressing challenges such as heteroskedasticity and model uncertainty, the study seeks to provide policymakers with actionable insights and accurate forecasts.

The findings of this research will contribute to the existing literature on liquidity forecasting while providing practical tools for central banks to better manage monetary policy implementation and ensure financial stability.

The structure of this paper is organized to provide a basic introduction to the methodologies, experiments, and findings in this research. After a brief literature review on current stance of liquidity forecasting in Section 2, the main part of the paper begins with the Section 3, outlining the data preprocessing techniques employed, which include power transformations and scalers, that have been employed to prepare the time series for modeling. This section also presents the individual forecasting models, including common econometric approaches such as SARIMA and machine learning models like Random Forest and neural network based methods, along with their configurations and evaluation metrics, as well as, ensemble learning methods, detailing the implementation of blending and stacking techniques to combine individual model forecasts. Section 4 discusses the results of the experiments, comparing the performance of individual models and ensemble approaches on key metrics. Finally, the Conclusion part provides a discussion of the findings, highlighting achievements, limitations, and directions for future research, emphasizing the potential of multivariate and structural models, as well as neural network architectures, for improving forecasting accuracy.

2 Literature Review

Forecasting the autonomous factors of banking system liquidity is a critical component of effective monetary policy implementation. Over the years, researchers have explored a wide range of econometric and machine learning models to address this challenge.

Understanding the importance of liquidity forecasting in central banking operations is crucial. Works like Borio [1], which discusses the role of central banks in managing liquidity, and Bindseil [2], which provides insights into central bank monetary policy implementation, offer foundational knowledge.

Research by Kashyap et el. [3] explores the dual role of banks as liquidity providers through both their lending activities and deposit-taking operations. The authors provide a theoretical framework that explains how banks manage liquidity risk and contribute to the transmission of monetary policy.

Econometric models have traditionally been the primary tools for time series forecasting in the financial domain. Seasonal Autoregressive Integrated Moving Average (SARIMA) models, for instance, are well-suited for capturing linear relationships and seasonal patterns in time series data. Studies such as [4] have demonstrated the efficacy of SARIMA in financial forecasting. Similarly, exponential smoothing models, including Holt-Winters methods, have been widely adopted for their simplicity and effectiveness in capturing trends and seasonality [5] in combination with a dynamic computational graph neural network system.

Studies like Sims [6] on VAR models lay the groundwork for analyzing interdependencies among financial time series, while Lütkepohl [7] provides advanced insights into VAR modeling techniques that can be applied to liquidity forecasting.

Stock and Watson [8] introduce Dynamic Factor Models (DFM) as a tool for analyzing economic indicators, which can be adapted to study the common factors affecting banking liquidity, as demonstrated by Banbura et al. [9] in their exploration of large datasets.

Despite their widespread use, econometric models are often limited in their ability to handle non-linear relationships and complex interactions among variables. This limitation has motivated the exploration of machine learning approaches.

Bao et al. [10] illustrate the application of LSTM networks in stock market prediction, offering insights into the potential of RNNs and LSTMs for liquidity forecasting. Research by Varian [11] on the use of machine learning in economic forecasting provides a basis for employing ensemble methods like Random Forests and LASSO for predicting liquidity factors.

Advanced techniques like stacking and blending, which combine the strengths of multiple models, have further enhanced the performance of machine learning models in forecasting tasks. These ensemble methods mitigate the risk of overfitting and model-specific biases by aggregating predictions from diverse models [12].

Studies focusing on the use of payment system data for liquidity analysis, such as Bech and Garratt [13], highlight the potential of high-frequency transaction data in enhancing liquidity forecasts, as it can provide real-time insights into banks' liquidity management strategies and their responses to different central bank policies.

Berger and Bouwman [14] offer insights into liquidity creation in banks, suggesting that a data-driven approach, leveraging large datasets, can provide a more nuanced understanding of liquidity dynamics.

The literature reveals a growing interest in applying advanced econometric and machine learning techniques to financial forecasting, with a particular emphasis on liquidity management in the banking sector. However, there appears to be a gap in comprehensive studies that integrate these methodologies with large payment system data to forecast banking system liquidity.

This research, aiming to examine and compare the effectiveness of these models in predicting liquidity factors, addresses this gap and contributes to both the theoretical and practical aspects of liquidity management in central banking.

3 Data and Methodology

3.1 Data Description

The dataset that was used for model estimations and testing the results contains transaction-level data from two systems: Interbank Payment System that operates as a RTGS (Real-Time Gross Settlement) system and ANOR (Instant Payment System), which are the main platforms for interbank payments.

The Interbank Payment System (RTGS) of the Central Bank of Uzbekistan is a critical component of the country's financial infrastructure. It is an electronic payment system designed for the real-time settlement of interbank payments, which ensures the instant transfer of funds between banks. All commercial banks are connected to this system, which operates on the principle of gross settlement, meaning each transaction is settled individually rather than being grouped together.

RTGS functions between 9:00 and 17:00 on weekdays, during which banks can send payment instructions to the Central Bank's Clearing Center, which is responsible for the continuous processing and settlement of transactions. This system plays a crucial role in maintaining liquidity positions of commercial banks as they can use the funds that are transferred to their accounts instantly.

The ANOR system functions as Uzbekistan's instant payment system, providing 24/7 processing on a netting basis. Launched to complement RTGS from February of 2020, ANOR supports transactions among business entities, as well as, government operations, even on the weekends and holidays.

The data structure for the study includes:

- Transaction metadata (date, time, sending and receiving bank IDs).
- Transaction value (total amounts transferred) and purpose.
- Client information (clients' account numbers, names, taxpayer ID).

Daily transaction data from the RTGS and ANOR systems provides rich, granular information about interbank liquidity movements. In the context of this research, only the transactions between the central bank and commercial banks will be used to generate time series data. The reason for using only these transactions is because they directly influence the overall level banking system's liquidity.

In 2024, in each of RTGS and ANOR, there were on average approximately 200,000 transactions daily between the Central Bank (including the government operations) and commercial banks, which contribute to the autonomous factors of liquidity. For this research, we aim to analyze combined daily transaction data from both payment systems starting from the year 2018 (ANOR data is available from February 2020). Each transaction includes the information about the date, transaction value, and purpose, as well as the initiating and receiving banks' identifiers (a 3-digit code), clients' names, bank account numbers (a 20-digit code), and taxpayer identification numbers (a 9-digit code).

For each autonomous factor of liquidity, a time series were constructed from daily bank transaction data. The time series corresponding to *cash transactions* is shown as *a representative example* to demonstrate the data preprocessing steps and modeling results for clarity and to streamline the analysis.

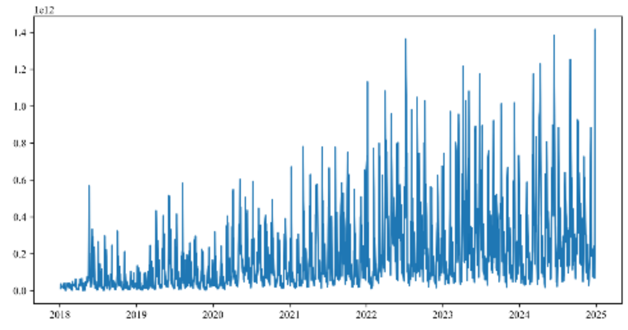


Figure 1: Original series

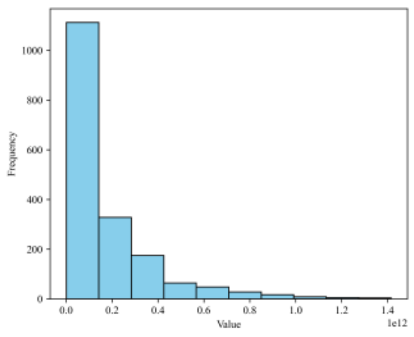


Figure 2: Distribution

As evident from the Figures 1 and 2, the data exhibits significant daily variability and a highly positively skewed distribution. For the purpose of liquidity forecasting in monetary operations, using weekly or 10-day aggregated data (Figure 3) and forecasts proves to be more practical. Additionally, increased aggregation of the data reduces variability, enhancing the stability of the forecasts.

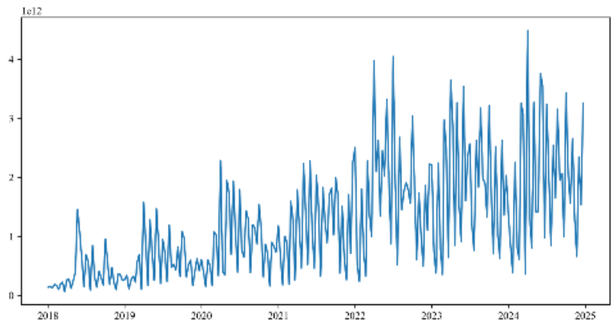


Figure 3: 10-day aggregated series

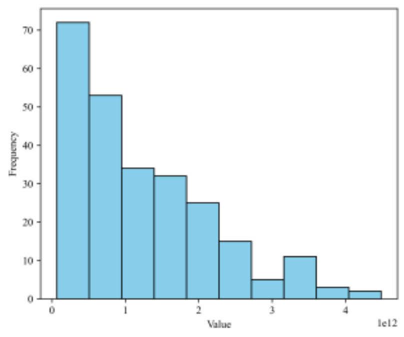


Figure 4: Distribution

For the majority of econometric and machine learning models, it is essential that the time series exhibits both trend and variance stationarity [15]. However, this requirement conflicts with the patterns observed in the plots and statistical tests, such as the Augmented Dickey-Fuller (ADF) test [16] for mean stationarity, which yielded a p-value of 0.3675, and Engle's test for Autoregressive Conditional Heteroscedasticity (ARCH) [17], which indicated statistically significant heteroscedasticity with a near-zero p-value. Furthermore, most machine learning models perform more effectively when the data is centrally distributed with no heavy tails. To achieve a more desirable data representation, we present various power transformations and data scaling techniques below.

3.2 Data Transformation Techniques

Data transformation involves modifying the format, structure, or values of data to make it easier for interpretation or make it more suitable for algorithmic processing. Key transformation techniques include:

3.2.1 Power transformations

- **Logarithmic Transformation** is useful for managing skewed data by compressing large values, reducing the relative difference between small and large values, thereby stabilizing variance and reducing the influence of extreme outliers [18].

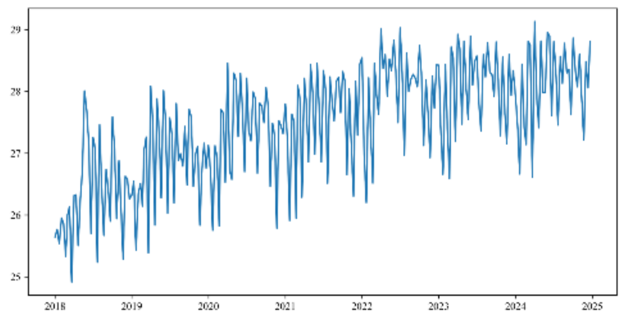


Figure 5: Logarithmic transformed

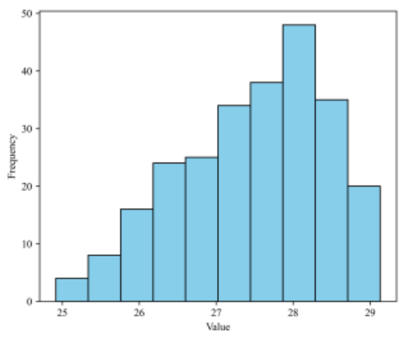


Figure 6: Distribution

- **Square Root Transformation** also helps mitigate the effects of high variance in the data by reducing the spread of larger values relative to smaller ones.

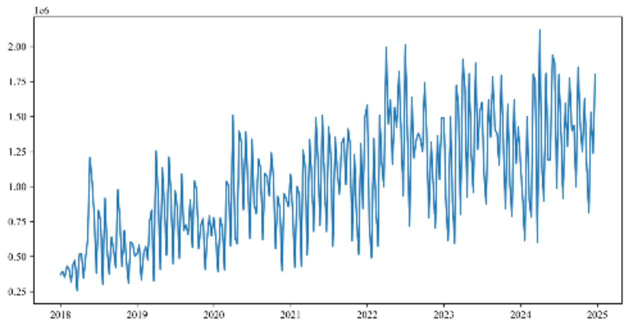


Figure 7: Square Root transformed

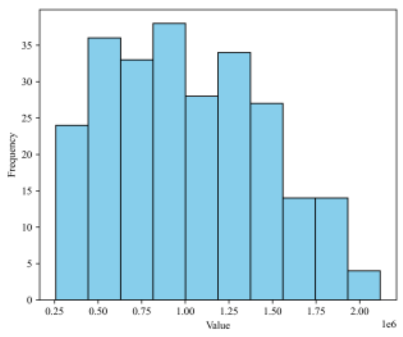


Figure 8: Distribution

- **Box-Cox Transformation** is widely applied to convert non-normal data into a more normal-like distribution [19]. It is defined as:

$$y = \begin{cases} \frac{x^\lambda - 1}{\lambda}, & \text{if } \lambda \neq 0 \\ \log(x), & \text{if } \lambda = 0 \end{cases}$$

  Here, λ is determined by maximizing the log-likelihood function. It is important to note that the Box-Cox transformation requires all input values to be strictly positive.

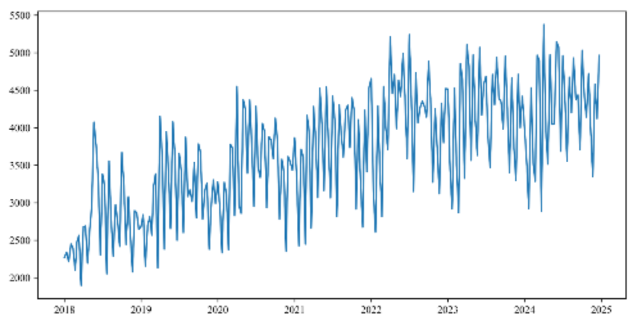


Figure 9: Box-Cox transformed

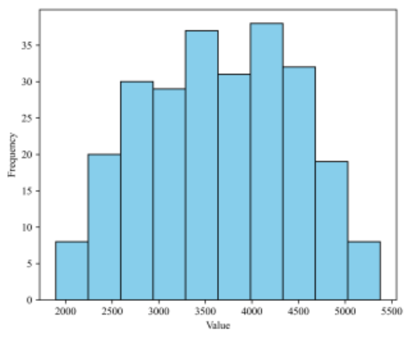


Figure 10: Distribution

- **Yeo-Johnson Transformation** is an extension of the Box-Cox method, allowing power transformations on variables that include zero or negative values, alongside positive values [20]. This transformation is particularly useful when preprocessing the net effect of an autonomous factor on overall liquidity, which may include negative values. Additionally, it is effective after applying scalers, such as the standard or robust scaler, which may shift the positive data into the negative axis.

3.2.2 Scaling

Scaling adjusts the range of values in the data without distorting the relative proportions between them (Figures 11-16). Common scaling methods include:

- **Min-Max Scaling:** Rescales the values to a specified range, typically [0, 1]. It is defined as:

$$x' = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$$

  where x_{min} and x_{max} are the minimum and maximum values of the feature, respectively.

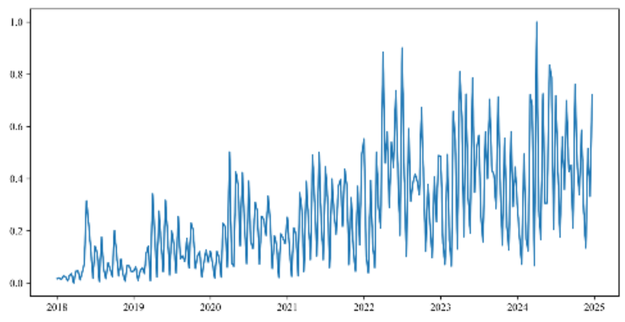


Figure 11: Min-Max scaled

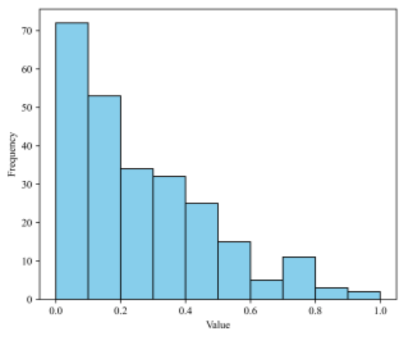


Figure 12: Distribution

- **Standard Scaling:** Centers the data to mean zero with a standard deviation of one:

$$x' = \frac{x - \mu}{\sigma}$$

  where μ is the mean and σ is the standard deviation of the feature.

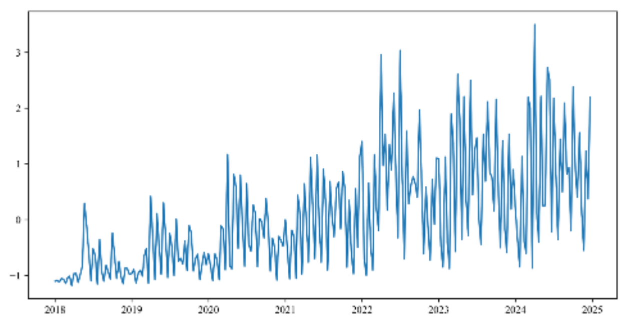


Figure 13: Standard scaled

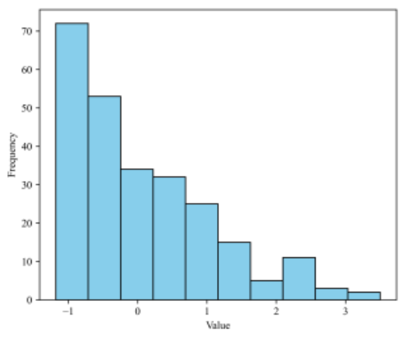


Figure 14: Distribution

- **Robust Scaling:** Scales the data using statistics that are robust to outliers. It subtracts the median and scales according to the interquartile range (IQR):

$$x' = \frac{x - \text{median}(x)}{\text{IQR}(x)}$$

where $\text{IQR} = Q_3 - Q_1$ is the difference between the 75th (Q_3) and 25th (Q_1) percentiles.

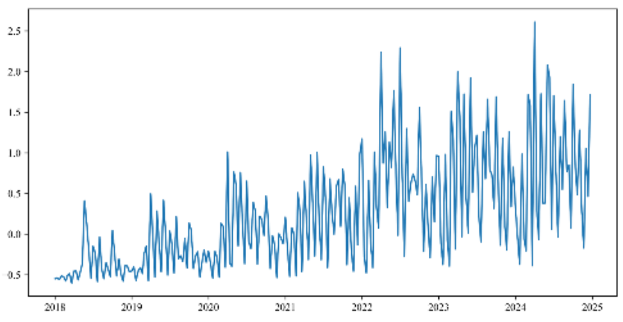


Figure 15: Robust scaled

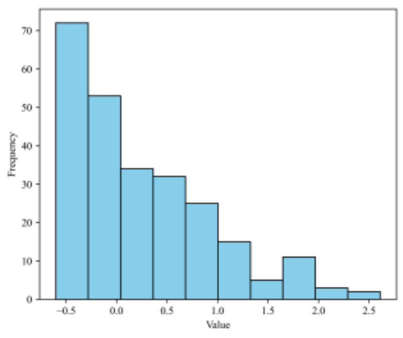


Figure 16: Distribution

These methods are essential for ensuring uniformity in feature magnitudes when dealing with multiple features, thereby improving the performance and convergence of machine learning models. Even in the case of univariate methods, in the absence of scaling, large input values can lead to a ML model that has learned excessively large weight values, which is often unstable meaning that it may suffer from poor performance during learning (training), on top of that resulting in higher generalization error [21].

3.2.3 Trend-cycle split

To ensure the mean stationarity of the data for some models, we apply detrending techniques and compare two approaches for calculating trends: the Hodrick-Prescott (HP) filter [22] and Generalized Least Squares (GLS) detrending [23].

Description of Methods

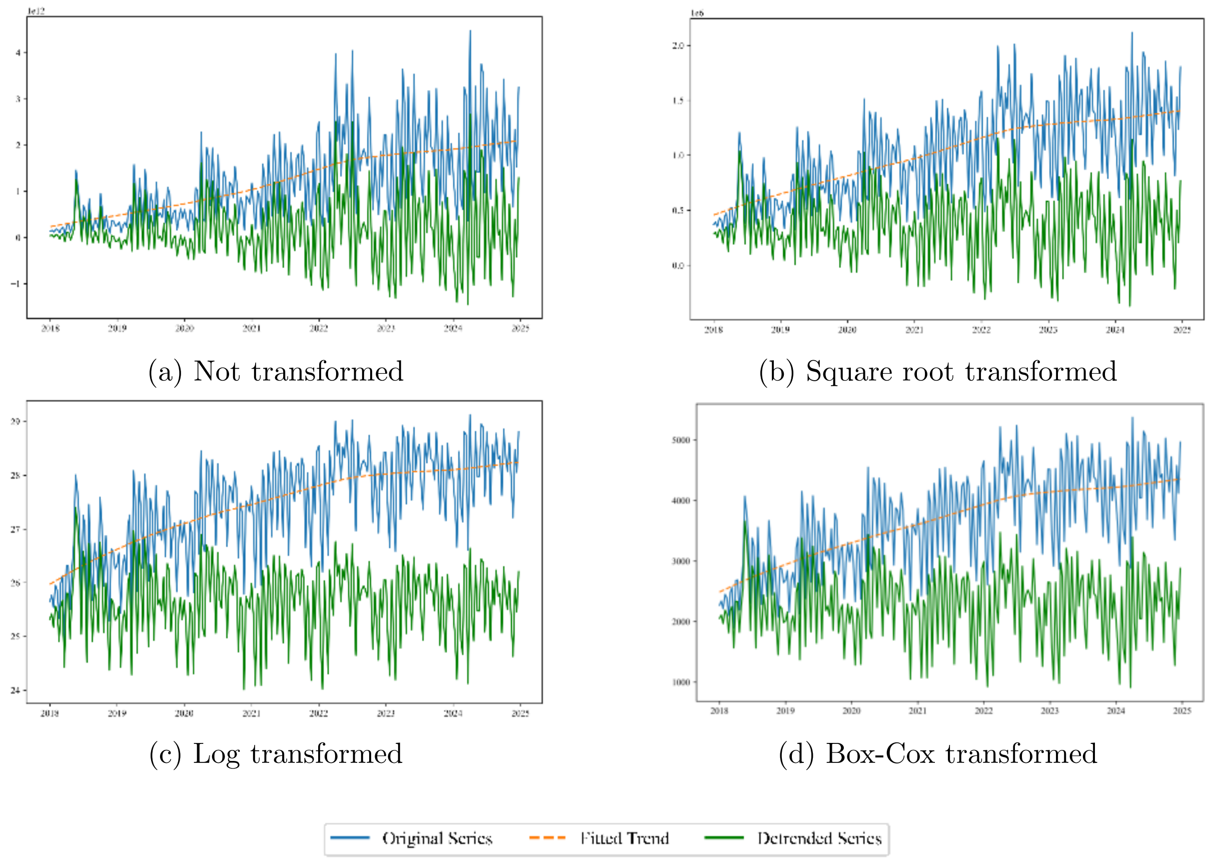
1. **Hodrick-Prescott (HP) Filter**: The HP filter decomposes a time series y_t into a trend component τ_t and a cyclical component c_t (Figure 17 [*Note:* the detrended series in the plots were vertically shifted upward by adding a constant to enhance visual clarity]), such that:

$$y_t = \tau_t + c_t$$

The trend component is estimated by minimizing the following objective function:

$$\min_{\tau} \sum_{t=1}^{T} (y_t - \tau_t)^2 + \lambda \sum_{t=2}^{T-1} ((\tau_{t+1} - \tau_t) - (\tau_t - \tau_{t-1}))^2$$

where λ is a smoothing parameter controlling the trade-off between the goodness of fit and the smoothness of the trend. The value of λ is set to 129600 $(1600 \cdot 3^4)$ after experimentation.



(a) Not transformed

(b) Square root transformed

(c) Log transformed

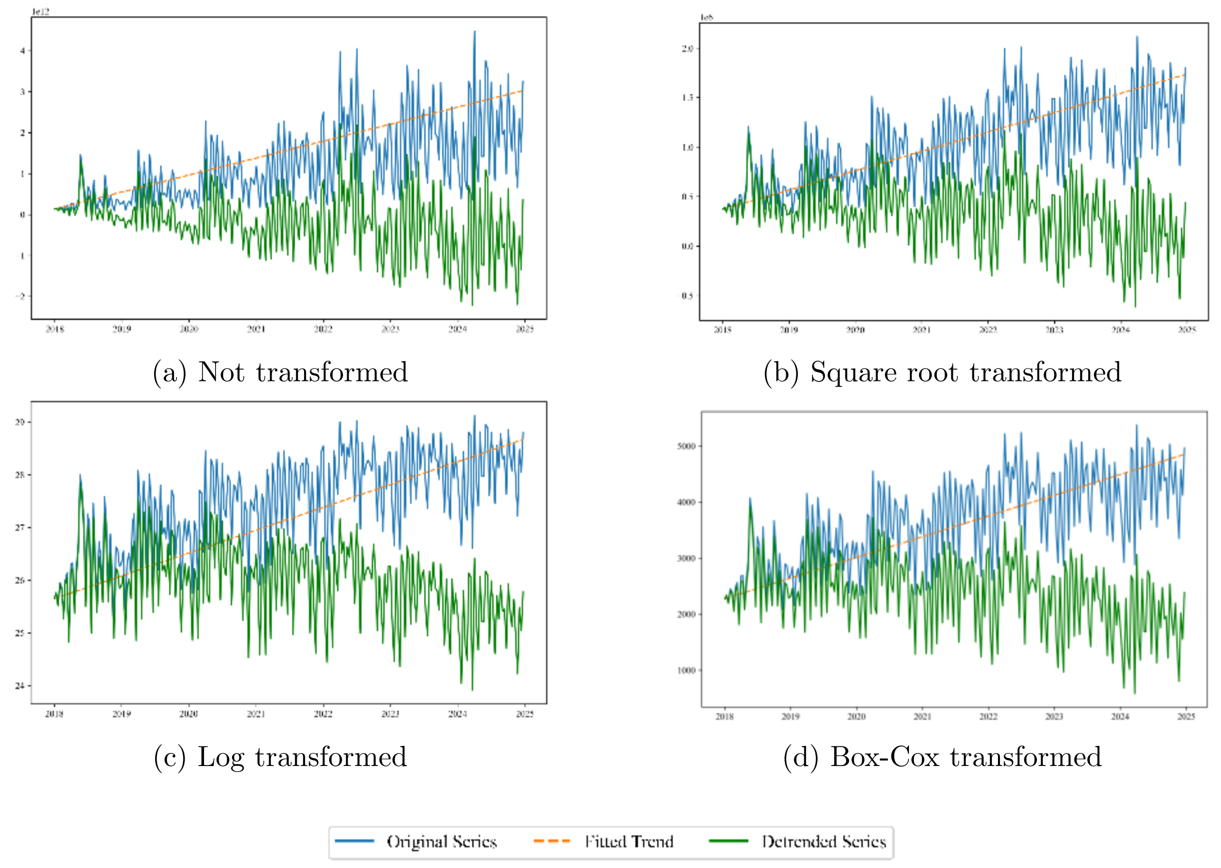
(d) Box-Cox transformed

Figure 17: HP filter detrending

2. **Generalized Least Squares (GLS) Detrending**: GLS detrending removes the trend by regressing the time series y_t on a deterministic time trend using generalized least squares (Figure 18 [*Note:* the trend values in the plots were vertically shifted upward by adding a constant to enhance visual clarity]). The detrended series is obtained as the residuals of the regression. For example, for a linear trend:

$$y_t = \beta_0 + \beta_1 t + \epsilon_t$$

The trend $\beta_0 + \beta_1 t$ is subtracted from y_t, leaving the stationary residual ϵ_t.



(a) Not transformed (b) Square root transformed

(c) Log transformed (d) Box-Cox transformed

Figure 18: GLS detrending

By employing and comparing these methods, we evaluate the effectiveness of each in achieving mean stationarity for further analysis.

3.3 Model taxonomy

There is a vast array of models available for forecasting time series in both econometric and machine learning domains. These models can be classified based on the variables they utilize, the model's outcome, and whether they incorporate underlying economic or financial theoretical structures, among other criteria. The following provides a structured classification of these models [24], which is not an exhaustive list:

1. By the Number of Variables

    - *Local Univariate Models*: separate models are trained independently for each time series, with no shared parameters or data across series.
    - *Global Univariate Models*: a single, global model is trained using data from all available time series, but it is designed to predict only a univariate target.
    - *Multivariate Models*: a single model is trained using all available data from multiple time series to predict multivariate outcomes, capturing relationships and dependencies across series.

2. By the Forecast Outcome
   - *Probabilistic Forecast Models*: these models predict the entire distribution of possible future values, providing uncertainty estimates.
   - *Point Forecast Models*: these models focus on specific values, such as the mean, median, or other quantiles, without explicitly modeling uncertainty.
3. By the Forecast Horizon
   - *One-Step Forecast Models*: these predict the next value in the series.
   - *Multi-Step Forecast Models*: these predict multiple future values, either iteratively or directly.
4. By Structural Incorporation
   - *Structural Models*: incorporate theoretical frameworks, such as economic or financial principles, to guide the modeling process.
   - *Non-Structural Models*: rely solely on data patterns without embedding domain-specific theories.

In this research, we first aim to conduct experiments using only local univariate models, which do not include any structural elements, with one-step forecasts at a time and build confidence intervals around the point estimate. The reason for starting with such models is that they are simple and interpretabe, making them easier for gaining initial insights and understanding the data patterns without introducing additional complexity.

In the next step of this long-term research work, we plan to experiment with other kinds of models as well, namely, more structural and multivariate models to better capture the interplay of economic and financial indicators.

3.4 Econometric Models

3.4.1 SARIMA

We begin our experimentation, often referred to as the "model horse racing," with the SARIMA (Seasonal Autoregressive Integrated Moving Average) model, as the data exhibits pronounced seasonal patterns. SARIMA is an extension of the ARIMA (Autoregressive Integrated Moving Average) model, initially introduced by Box and Jenkins [25]. It has been widely applied in many domains practically so far, for example, economics [26, 27], meteorology [28, 29], and retail [30, 31].

SARIMA is a univariate model that uses its own past values (lags) – autoregressive part, differencing to impose stationarity when necessary – integrated part, error terms – moving average part, to predict future values of a time series.

The general mathematical form of the SARIMA model is given as follows [32, 33]:

$$\Phi_P(B^s)\phi_p(B)(1-B)^d(1-B^s)^D y_t = \Theta_Q(B^s)\theta_q(B)\epsilon_t,$$

where:

- B is the backward shift operator, such that $B^k y_t = y_{t-k}$.
- d and D represent the non-seasonal and seasonal differencing orders, respectively.

- s denotes the length of the seasonal period.
- $\phi_p(B)$ and $\Phi_P(B^s)$ are the non-seasonal and seasonal autoregressive (AR) operators of orders p and P, respectively.
- $\theta_q(B)$ and $\Theta_Q(B^s)$ are the non-seasonal and seasonal moving average (MA) operators of orders q and Q, respectively.
- ϵ_t is the white noise error term.

The model's hyperparameters for the non-seasonal component are (p, d, q) and for the seasonal component are (P, D, Q, s). SARIMA model can identify the interdependencies within the data over both short-term lags and seasonal cycles, which allows it to effectively model and forecast time series where seasonal patterns are profound.

Model Identification and estimation

To identify SARIMA model's hyperameters for seasonal and non-seasonal components, there is a well-known iterative Box-Jenkins procedure [25], which includes visually examining the data and fine-tuning hyperparameters accordingly. Since we plan to experiment with different scalers, power transformations and many time series, such a procedure can become relatively time-intensive and impractical for our purposes.

Instead of that, we utilize the `auto_arima` function from Python's `pmdarima` library. This function efficiently determines the optimal hyperparameters by using statistical tests and an iterative model-fitting process. Specifically, in our case, `auto_arima` employs the *Akaike Information Criterion (AIC)* to assess the goodness-of-fit and penalize model complexity. It begins by performing a differencing test to estimate the optimal order of differencing (d) for non-seasonal components. Once d is established, the function iteratively fits models within parameter ranges for p (autoregressive order) and q (moving average order) defined by the user.

For seasonal models, `auto_arima` extends this functionality by identifying the seasonal parameters P and Q. It first applies the *Canova-Hansen test* [34] to determine the seasonal differencing order (D) and subsequently fits models over a grid of seasonal hyperparameters.

To enhance computational efficiency, we employ the **stepwise algorithm** option within `auto_arima`. This approach systematically searches through the parameter space, intelligently skipping certain combinations based on prior results. While a non-stepwise method, which exhaustively explores all parameter combinations (similar to a grid search), can be used, it is significantly slower, especially for datasets with pronounced seasonality. The theoretical foundation for the stepwise algorithm is detailed in the work of Hyndman and Khandakar [35].

By automating the hyperparameter selection process with `auto_arima`, we save considerable time and ensure consistency across our experiments, enabling us to focus on evaluating transformations and comparing models effectively.

Once the hyperparameters of the SARIMA model are identified, the next crucial step is to **estimate** the model's parameters. Common methods for parameter estimation include Maximum Likelihood Estimation (MLE) [32], Generalized Method of Moments (GMM) [36], and Bayesian Estimation [37].

While each method has its advantages and drawbacks, some, like Bayesian Estimation, are often preferred in certain contexts due to their ability to explicitly handle parameter uncertainty and provide a probabilistic framework for modeling. However, Bayesian approaches can be computationally intensive and sensitive to the choice of prior distributions, making them less practical for models with large datasets.

In this research, we adopt the Maximum Likelihood Estimation (MLE) method, which is the most widely used approach for SARIMA models. The choice of MLE aligns with our focus on accuracy and computational feasibility, especially when working with multiple time series.

3.4.2 Prophet

Prophet, developed by Facebook [38], is a robust and flexible time series forecasting tool designed to address common challenges such as missing data, outliers, and multiple seasonality.

Prophet decomposes a time series into three main components: trend ($g(t)$), seasonality ($s(t)$), and holidays ($h(t)$). The model can be represented as:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t$$

where:
- $y(t)$: Observed value at time t
- $g(t)$: Trend component modeling long-term growth
- $s(t)$: Seasonal component capturing periodic patterns
- $h(t)$: Holiday component capturing effects of special events
- ϵ_t: Residual error term, assumed to be normally distributed with mean 0 and variance σ^2, i.e., $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$

- The trend component can take two forms:

  1. Piecewise Linear Trend:

     $$g(t) = (k + a(t)^\top \delta)t + b$$

     where:
     - k: Initial growth rate
     - $a(t)$: Indicator function for change points
     - δ: Adjustments to growth rates at change points
     - b: Offset parameter

  2. Logistic Growth Trend:

     $$g(t) = \frac{C}{1 + \exp(-k(t - m))}$$

     where:
     - C: Carrying capacity
     - k: Growth rate
     - m: Time at which growth transitions to saturation

- The seasonality component is modeled using a Fourier series:

$$s(t) = \sum_{n=1}^{N} \left[a_n \cos\left(\frac{2\pi nt}{P}\right) + b_n \sin\left(\frac{2\pi nt}{P}\right) \right]$$

where:

- P: Period of the seasonality (e.g., 365.25 for yearly seasonality)

- N: Number of Fourier terms

- a_n, b_n: Fourier coefficients

- The holiday effects are modeled as:

$$h(t) = \sum_{i=1}^{L} \mathbf{1}_{\text{holiday}_i}(t)\, \beta_i$$

where:

- $\mathbf{1}_{\text{holiday}_i}(t)$: Indicator function that equals 1 if t is within the holiday period i, 0 otherwise

- β_i: Impact of holiday i

The parameters $\theta = \{k, \delta, a_n, b_n, \beta_i, \sigma^2\}$ are **estimated** using maximum a posteriori (MAP) estimation [39], leveraging regularization to prevent overfitting.

A key advantage of Prophet is its user-friendly parameter tuning, which enables domain experts with minimal statistical expertise to achieve accurate forecasts. Prophet is particularly well-suited for business applications, such as cash flow prediction [40], food prices forecasting [41] and anomaly detection [42], where handling irregularities in data is essential.

In this research, we utilized Python's `prophet` library to implement the forecasting model with a linear trend. Three configurations were evaluated: the default Prophet model (vanilla Prophet), Prophet with added monthly seasonality, and Prophet enhanced with both monthly seasonality and holiday effects specific to Uzbekistan.

3.4.3 Exponential Smoothing

Exponential Smoothing is a forecasting technique that applies exponentially decreasing weights to past observations, assigning more importance to recent data. This method is highly versatile and can accommodate various data patterns, including level, trend, and seasonality. Variations such as Holt's Linear Trend Model [43] and Holt-Winters Seasonal Model [44] extend the basic approach to capture trends and seasonal effects, respectively. We make use of a Holt-Winters Seasonal Model with additive seasonality in this research as per [45] using `ExponentialSmoothing` class from Python's `statsmodels.tsa.holtwinters` module:

$$\begin{aligned}
L_t &= \alpha(Y_t - S_{t-m}) + (1-\alpha)(L_{t-1} + T_{t-1}), \\
T_t &= \beta(L_t - L_{t-1}) + (1-\beta)T_{t-1}, \\
S_t &= \gamma(Y_t - L_t) + (1-\gamma)S_{t-m}, \\
\hat{Y}_{t+h} &= L_t + hT_t + S_{t-m+h},
\end{aligned}$$

where:
- L_t: Smoothed level at time t,
- T_t: Trend at time t,
- S_t: Seasonal component for period m,

- $\hat{Y}_{t+h}$: Forecast for h periods ahead,
- $\alpha, \beta, \gamma \in [0, 1]$: Smoothing parameters,
- m: Length of the seasonality.

The parameters are estimated by maximizing the likelihood function of the model under the assumption of normally distributed errors. The likelihood function is defined as:

$$L(\theta) = \prod_{t=1}^{T} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(Y_t - \hat{Y}_t)^2}{2\sigma^2}\right),$$

where $\theta = (\alpha, \beta, \gamma)$ are the smoothing parameters, and σ^2 is the error variance.

Log-likelihood is used for computational convenience:

$$\log L(\theta) = -\frac{T}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{t=1}^{T}(Y_t - \hat{Y}_t)^2.$$

The parameters are chosen to maximize $\log L(\theta)$.

Exponential Smoothing is computationally efficient and is often favored for its simplicity and effectiveness in short-term forecasting.

3.5 Machine Learning Models

The application of machine learning models in time series forecasting has gained significant attention due to their ability to handle complex patterns, non-linear relationships, and high-dimensional data. This section explores key machine learning models used in this research.

3.5.1 Tree-based methods

Decision trees (DT) are a completely different technique for predicting compared to linear methods. Tree-based methods attempt to recursively segment the feature space (independent variables) into simple regions, allowing decisions to be made about the values of the predictor space (dependent variable). In its simpler forms, a single tree mimics the process of human decision-making, and this process can be visualized, for example, as a flowchart, which makes it interpretable. One disadvantage of single tree-based models is that, in more complex problems, a tree may grow too large, leading to overfitting the data, resulting in good performance on the training set but poor test error (bias-variance trade off [46]).

For DT, there are several ways to avoid overfitting. The first technique is called pruning [47], which involves growing a large tree and then pruning it back to obtain a subtree. This is also known as cost-complexity pruning or weakest-link pruning. While pruning can reduce the risk of overfitting, if not conducted with great care, it may lead to underfitting due to the loss of potentially important information.

Another method is bootstrap aggregation, or bagging [48], which is implemented by bootstrapping — sampling from the data with replacement — to create multiple training sets. Large trees are then grown without pruning to obtain predictive models with high variance. These possibly overfitted trees are averaged to produce a bagged model with reduced variance. In practice, bagging has been shown to significantly improve prediction accuracy [49]. However, one disadvantage of bagging is that the member trees are not entirely independent from each other due to sampling with replacement during the bootstrapping process. This can result in dramatically different reductions in variance across datasets.

Random forest [50] directly addresses this limitation by extending the concept of bagging. To eliminate dependence (decorrelate) between the trees grown through bootstrapping, constraints are introduced during the tree-growing process by randomly selecting a subset of predictors from all possible ones at each split. In this process, fully grown trees are used, as more important features may be selected in later splits. This approach results in lower overall variance because each bootstrapped tree is now split differently. For the final prediction, as in bagging, the predictions from all bootstrapped trees are averaged for regression problems.

Considering the abovementioned advantages of the Random Forest models, in this study, we use Python's `RandomForestRegressor` model from the `sklearn` library to forecast time series. The forecasting process consists of several steps (Figure 21):

- After loading the series, it is split into train and test sets. Approximately 5 percent of the data is used for testing the performance.
- Feature engineering: year, month number and the decade number are extracted from the dates of the time series as new features (independent variables). A Min-Max scaler is applied for normalization. Although one-hot encoding is commonly used for such categorical features, it resulted in poorer performance in our case.
- Different power transformations and scalers (Subsections 3.2.1 and 3.2.2) are applied to the time series values in the train set.
- The processed series is split into trend and cycle components (detrending) using the HP filter (Subsection 3.2.3) due to its ability to better capture a non-linear trend evident in the series.
- For the trend component, Python's `auto_arima` function is used to find the ARIMA model's best hyperparameters.
- For the cycle component, to maintain temporal order in the dataset, `TimeSeriesSplit` is employed with number of splits of 5 with expanding window instead of commonly used `KFold` split for the cross-validation (CV) process (Figures 19-20).

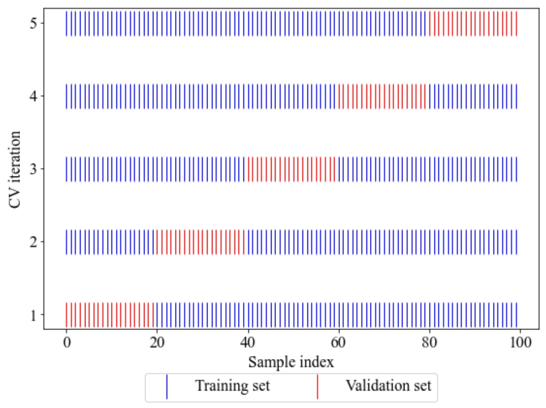


Figure 19: `KFold`

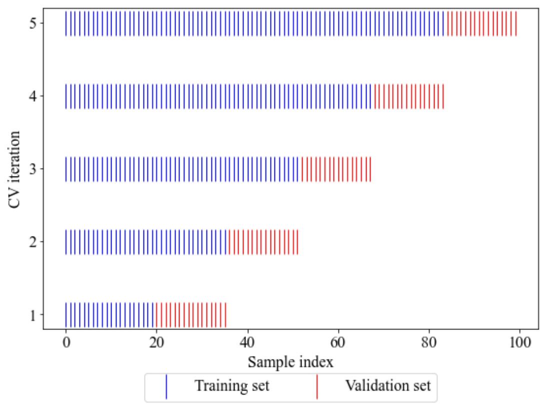


Figure 20: `TimeSeriesSplit`

- For hyperparameter optimization, `Optuna` [51] is a powerful tool for exploring large and complex parameter spaces efficiently, but its effectiveness depends on the parameter space definition, the number of trials, and evaluation consistency. On the other hand, `GridSearchCV`, although computationally expensive, can outperform `Optuna` in smaller, well-defined parameter spaces. Therefore, to achieve stable and improved results, the optimal hyperparameters of the Random Forest model are determined in two steps:

  1) Using `GridSearchCV`, a strong initial set of hyperparameters is identified from predefined possible set of parameters (Table 1).

Parameter name	**Description**	**Values**
`bootstrap`	Whether bootstrapping is needed	`True, False`
`n_estimators`	Number of trees in the forest	50, 150, 200, 250, 300, 400, 600
`max_depth`	Maximum depth of each tree	1, 3, 5, 10, 20, 30, 40, 60, 80, 100
`min_samples_split`	Minimum samples required to split a node	1, 2, 5, 10
`min_samples_leaf`	Minimum samples required at each leaf node	1, 2, 4
`max_features`	Number of features to consider at each split	1.0, `'sqrt', 'log2'`

Table 1: Parameter grid for `GridSearchCV`

  2) Incorporating the best parameters from `GridSearchCV` as initial trials in `Optuna`, which performs a more flexible search using its trial-based optimization framework, hyperparameters are further refined with the number of trials of 250.

- To enhance robustness and diversity, a set of 10 best hyperparameters, consisting of a hyperparameter set from `GridSearchCV` and 9 best hyperparameter sets from `Optuna`, is selected to find the optimal number of Random Forest models to compute the average forecast.

- the best Random Forest models are retrained on the entire train set and the cycle component forecast is produced by averaging predictions from the optimal number of models. This ensemble approach improves the stability and accuracy of the forecast by mitigating the variance inherent in individual models.

- Trend forecast from ARIMA and cycle forecast from Random Forest are combined.

- To the resulting forecast, respective inverse scaler and power transformation are applied to return the data to its original scale and distribution.

- Forecast performance metrics are calculated against the test set to estimate the model's accuracy.

Since random sampling is an inherent feature of Random Forest models, ensuring reproducible and stable results requires setting a consistent random state. Therefore, a consensus random state of 42 is used across all instances where providing a random seed is applicable to the models.

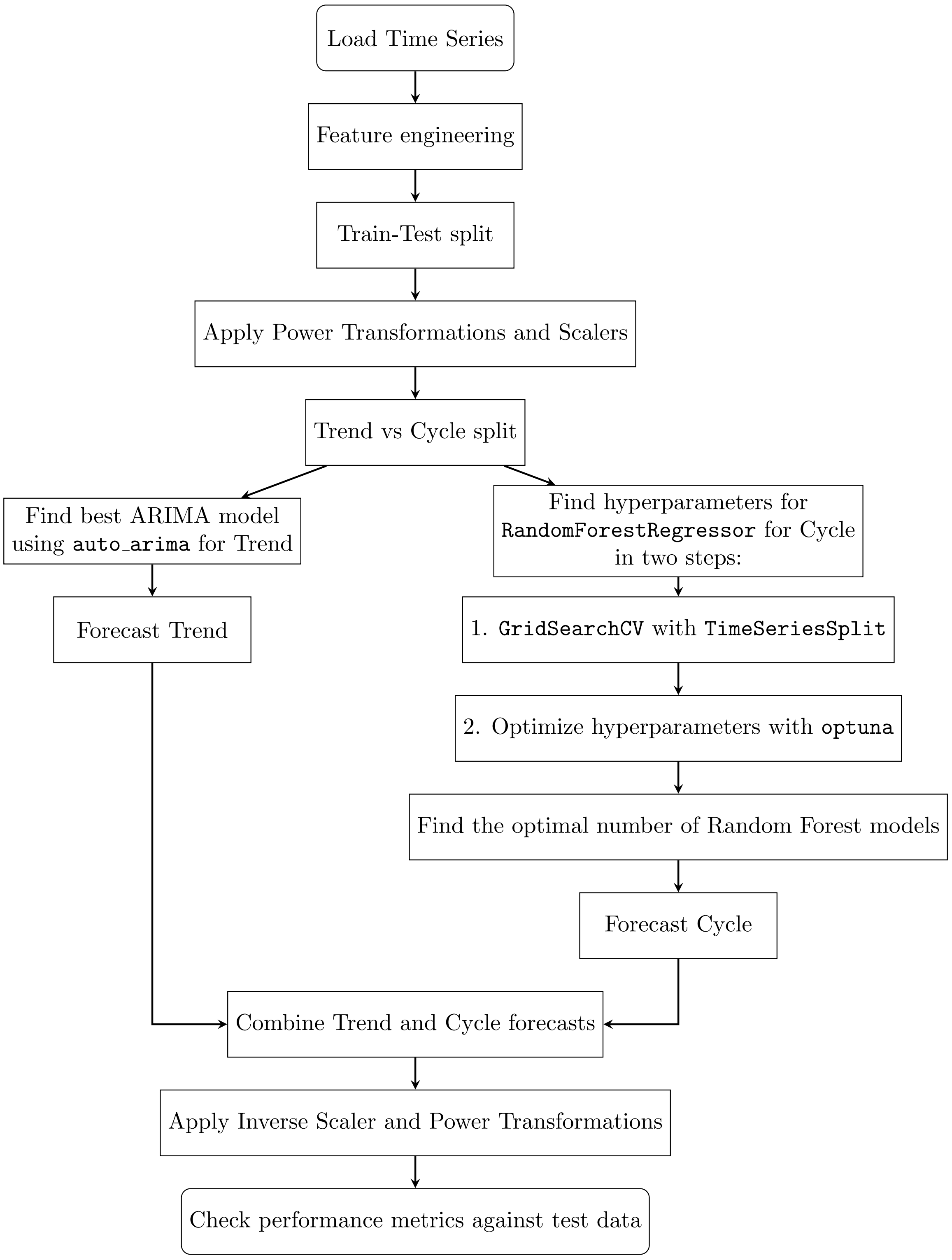

Load Time Series
Feature engineering
Train-Test split
Apply Power Transformations and Scalers
Trend vs Cycle split
Find best ARIMA model using auto_arima for Trend
Forecast Trend
Find hyperparameters for RandomForestRegressor for Cycle in two steps:
1. GridSearchCV with TimeSeriesSplit
2. Optimize hyperparameters with optuna
Find the optimal number of Random Forest models
Forecast Cycle
Combine Trend and Cycle forecasts
Apply Inverse Scaler and Power Transformations
Check performance metrics against test data

Figure 21: Flowchart of the Forecasting Process with Random Forest

3.5.2 Neural Networks

In recent years, neural networks have demonstrated remarkable predictive performance when applied to sequential data, such as natural language in the form of text and audio, as well as music, video, and time series.

At their core, neural networks are just the chains of tensor operations, which are at the same time complex geometric transformations of the input data in a high-dimensional space [52]. Neural networks with deep architectures learn from data by determining the reverse geometric transformations that best map the input to the desired output, using a series of sequential and simple transformation steps. In practice, this is achieved through a specialized optimization algorithm known as backpropagation [53].

The two widely used Machine learning (ML) techniques for sequential data, particularly within the deep learning sub-field of ML, are Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM). These are indispensable building blocks of larger artificial neural network architectures that currently achieve state-of-the-art performance [54, 55, 56], excelling at capturing temporal patterns and dependencies within the given data.

Recurrent Neural Networks are able to process sequential data over multiple steps thanks to their ability to maintain a memory unit, which is often called as a hidden state. A memory unit is taken into account when processing the input data to generate output for the next layers within the neural network's architecture.

However, ordinary RNNs often struggle to address more complex tasks due to fundamental issues, such as information morphing, and technical challenges, such as the vanishing gradient problem [57]. While the literature offers potential solutions [58, 59], such as preserving memory through addition or subtraction operations [60] — which shift rather than scale the memory — an improved variant of RNNs, designed to mitigate these issues, is often preferred.

One of such kind of specially tailored neural networks, which was built specifically to address the problem of vanishing gradient, is a *Gated Recurrent Unit (GRU)* [61]. It employs the gating mechanisms to control the flow of information and to prevent the issue of vanishing gradient. They are a simplified variant of LSTM units, with a lower number of parameters, making them computationally less demanding at the same time retaining their ability to learn the long-term patterns.

In this study, we employ an encoder-decoder approach, also known as autoencoders, to construct a simple Sequence-to-Sequence (S2S) architecture using GRU units for time series forecasting. Peng et al. [62] utilized a similar architectural model for host load prediction on the Google Clusters dataset and the Dinda dataset and compared its performance against two other highly proficient models, namely, Recurrent Neural Networks, which was an LSTM-based network, and Echo State Networks, drawing the conclusion that the GRU model with an autoencoder architecture outperformed both models across both datasets.

3.5.3 Wavelet Transforms Combined with Neural Networks

The Wavelet-ANN method was introduced by Chan and Mehralizadeh [63], which is a hybrid method that combines wavelet denoising with artificial neural network to forecast financial futures markets. The main idea is to denoise the time series with wavelets transforms, feature engineering from the financial technical indicators, then feed these into a neural network model. The difference in our research from the paper's approach for forecasting is that the authors employ principal component analysis (PCA) to extract principal components from financial indicators' denoised Open, High, Low, Close price values, while we do not conduct PCA because only single series is utilized at a time.

As additional features for the neural network model a range of financial technical indicators were computed. These include Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), MACD Signal, MACD Histogram, Stochastics (slow %K, and %D):

$$\text{RSI} = 100 - \frac{100}{1 + \frac{\text{Average Gain}}{\text{Average Loss}}}$$

$$\text{MACD} = \text{EMA}_{\text{short}} - \text{EMA}_{\text{long}}$$

where:

- $\text{EMA}_{\text{short}}$ is the Exponential Moving Average with a shorter period (12 days).
- EMA_{long} is the Exponential Moving Average with a longer period (26 days).

$$\text{MACD Signal} = \text{EMA}_{\text{signal}}(\text{MACD})$$

where $\text{EMA}_{\text{signal}}$ is typically a 9-day Exponential Moving Average of the MACD.

$$\text{MACD Histogram} = \text{MACD} - \text{MACD Signal}$$

$$\text{Slow } \%K = \text{SMA}_{\text{slow}}(\text{Fast } \%K)$$

where SMA_{slow} is a Simple Moving Average applied to the Fast %K.

$$\%D = \text{SMA}_{\text{d}}(\text{Slow } \%K)$$

where SMA_{d} is a Simple Moving Average applied to the Slow %K.

As in the work of [63], we utilize 23 types of wavelet settings[1] (see Table 2), with decomposition levels ranging from 1 to 8 and five thresholding strategies: soft, hard, garrote, greater, and less. For the artificial neural network component of the framework, a sequential model comprising 3 to 5 LSTM layers (each with 50 units) was implemented using Python's `keras` module with default configuration settings. The model was trained over 20 epochs with a batch size of 32.

Wavelet name	Subsets
Haar	haar
Daubechies	db2, db3, db4, db5, db6, db7, db8, db9, db10
Symlets	sym2, sym3, sym4, sym5, sym6, sym7, sym8
Coiflets	coif1, coif2, coif3, coif4, coif5
DMeyer	dmey

Table 2: Wavelet families and subsets

To sum up, each of these models offers distinct advantages, and their selection depends on the characteristics of the time series and the forecasting objectives. By integrating wavelet transforms, GRUs, and Random Forest, this research explores both traditional and innovative approaches to time series modeling, aiming to identify the most effective methodologies for forecasting autonomous factors of banking system liquidity.

[1]Subsets are presented in their shortened form as been appeared in Python's `wavedec` function of the `pywt` library

3.6 Ensemble Methods

In the practice of machine learning modeling, an ensemble of multiple models often outperforms the best-performing individual model [64, 65]. The reason for this is that the variance in the final individual model's performance can be reduced by adding bias. This is achieved by combining predictions from multiple models, a process commonly referred to as *ensemble learning.* It is advised that the models used for ensembling have different skills [21], for example, linear versus non-linear models or tree-based models versus neural networks. Additionally, since the training process of most machine learning models involves stochastic elements, averaging the predictions from different runs helps ensure consistent performance.

In this work, blending and stacking methods of ensemble learning [21], as well as, inverse expected variance based weighting [66] are employed to combine forecasts from multiple models. In blending, each model in the ensemble is assigned a weight, which reflect the blending model's confidence in its members' predictions. Blending weights are determined based on train Mean Absolute Error (MAE). Stacking uses another model (meta-model) to learn optimal combinations of predictions. A linear regression model was used as the meta-learner in this research. In the inverse expected variance method, the weights of ensemble members are determined based on the inverse of their predicted model variances.

3.7 Performance measurement

In this study, we select three commonly used indicators for evaluating the forecasting performance of the models in financial time series modeling [67, 68, 69]:

- **The Mean Absolute Percentage Error (MAPE)** measures the average percentage difference between actual and predicted values. It is defined as:

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

  A lower MAPE value indicates better forecasting accuracy.

- **The R^2 metric**, or the coefficient of determination, indicates the proportion of variance in the actual values that is predictable from the model. It is defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

  An R^2 value closer to 1 indicates a better fit of the model.

- **Theil's U Statistic** measures the relative accuracy of a forecasting model compared to a naive model. It is defined as:

$$U = \frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n} y_i^2} + \sqrt{\frac{1}{n}\sum_{i=1}^{n} \hat{y}_i^2}}$$

  Theil's U value ranges between 0 and 1. A value closer to 0 indicates higher accuracy, while values near 1 suggest performance close to a naive model.

In these equations, y_i is an actual value at time i, $\hat{y}_i$ is a predicted value at time i, $\bar{y}$ is the mean of actual values, n is the length of the forecasting period. Since the data is split into training and test sets, all performance metrics are calculated using the test set, meaning that the measurements reflect **out-of-sample performance**.

Choosing the best-performing models among potential candidates when they exhibit conflicting metrics (i.e., MAPE, R-squared, Theil's U) can be challenging. To address this, we aggregate the metrics into a single performance score. A weighted composite score [70] is calculated by normalizing the metrics and assigning appropriate weights. This approach allows for ranking and selecting the optimal model configuration based on a holistic evaluation of all metrics.

Calculating the composite score is conducted in the following steps:

- **Step 1:** Normalize each metric to a comparable scale (between 0 and 1).
- **Step 2:** Assign weights to each metric based on their importance for the application. In our case, we set $w_{\text{MAPE}} = 2/3$, $w_{R^2} = 1/6$, $w_{\text{Theil's U}} = 1/6$. MAPE has more weight because accuracy is prioritized.
- **Step 3:** Compute a composite score for each model using the weighted average of the normalized metrics.
- **Step 4:** Choose the model with the lowest composite score.

The composite score can be computed as:

$$\text{Score} = w_{\text{MAPE}} \cdot \text{MAPE}_{\text{norm}} + w_{R^2} \cdot (1 - R^2)_{\text{norm}} + w_{\text{Theil's U}} \cdot \text{Theil's U}_{\text{norm}}$$

3.8 Estimating uncertainty of the forecasts

The models described so far produce only point forecasts. However, for policymakers, it is more insightful to understand the uncertainty surrounding the forecast, expressed as confidence intervals.

For SARIMA and Prophet, the confidence intervals provided by their respective model forecasts were utilized [71, 72]. For Exponential Smoothing, constructing confidence intervals around the point forecasts requires calculating the standard error of the forecast. In this study, the standard error of the model's forecast is determined under two scenarios:

1. **Presence of Autoregressive Conditional Heteroscedasticity (ARCH) Effects**:
   - If the residuals exhibit ARCH effects based on Engle's ARCH test, we model the variance using a GARCH process.
   - The order of the GARCH model is determined using grid search optimization.
2. **Absence of ARCH Effects**:
   - If ARCH effects are not detected, the forecast variance is calculated as the sum of the variance of the fitted values and the variance of the residuals. The formula is given by (under the conditions of linearity, independence and zero mean residuals):

$$\text{Var}(y_{forecast}) = \text{Var}(y_{\text{fitted}}) + \sigma^2_{\text{residuals}}$$

From the forecast variance, the standard error is computed, and the lower and upper bounds of the confidence interval are calculated using the formulas:

$$\text{Lower Bound} = \text{Point Forecast} - z_\alpha \cdot \text{Forecast Standard Error}$$
$$\text{Upper Bound} = \text{Point Forecast} + z_\alpha \cdot \text{Forecast Standard Error}$$

Here, z_α is the critical value from the standard normal distribution corresponding to the desired significance level (α).

For the Random Forest model, the confidence intervals are constructed differently. The lower and upper bounds are computed as the 2.5th and 97.5th percentiles of the predictions from individual trees, assuming a significance level of 5%. This approach directly captures the variability across the ensemble of tree-based predictions, is computationally not costly, and has favorable statistical properties, such as confidence intervals being automatically invariant under monotonic transformations of the relevant amount [73], which is useful for regression problems when power transformations are applied.

For the ensemble models, the variance is calculated as:

$$\text{Var}(y_{forecast}) = \sum_{i=1}^{k} w_i^2 \cdot \sigma_i^2 + \sigma_{\text{meta}}^2$$

where:

- w_i represents the weight assigned to the i-th member model,
- σ_i^2 is the variance of the i-th member model,
- σ_{meta}^2 denotes the variance of the meta-model.
- k is the number of ensembling models

This formula accounts for the contribution of each individual model's variance, weighted by the square of their blending weights, as well as the variance of the meta-model.

4 Results

4.1 SARIMA

As described in previous sections, different configurations of SARIMA models were tested using different combinations of power transformations and scalers to forecast the time series data for the Cash operations. The best-performing SARIMA model was identified with no transformation and Min-Max scaling, achieving the lowest MAPE (0.1747), the highest R^2 (0.7774), and the lowest Theil's U value (0.0903). This combination also resulted in the lowest composite score, making it the optimal choice for forecasting (Table 3).

The selected SARIMA model was fitted with the order $(2, 1, 3)(1, 0, 1, 36)$ and excluded the trend component. This configuration successfully captured the seasonal and non-seasonal dynamics of the series, as can be seen from its forecasting accuracy. The forecast plot (Figure 22) highlights the model's ability to align closely with the actual data while providing 95% confidence intervals around the forecast values.

[2]The comparison of power transformations is provided only for no transformation applied (none) and the Yeo-Johnson transformation, as the latter is the most versatile and is chosen for the sake of conciseness.

Transformation[2]	Scaler	MAPE	R^2	Theils U	Composite Score
none	minmax	0.1747	0.7774	0.0903	0.0000
none	robust	0.1751	0.7775	0.0902	0.0003
none	standard	0.1752	0.7764	0.0905	0.0005
yeo-johnson	standard	0.1893	0.6694	0.1155	0.0211
yeo-johnson	robust	0.1904	0.6596	0.1176	0.0227
yeo-johnson	minmax	0.1881	0.6047	0.1270	0.0242
none	none	0.2089	0.7587	0.0921	0.0309
yeo-johnson	none	0.9343	-5.2088	0.8899	1.0000

Table 3: Forecast performance metrics comparison

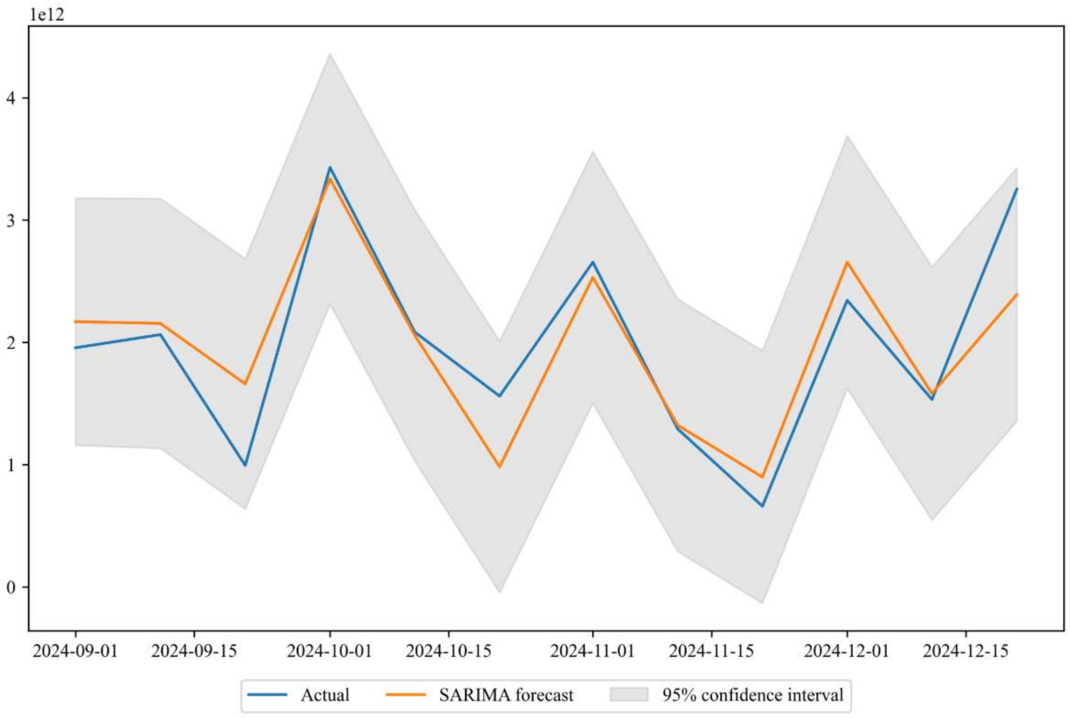


Figure 22: SARIMA best model forecast and confidence intervals

4.2 Prophet

Prophet, a robust time series forecasting framework, was configured to incorporate monthly seasonality and Uzbekistan-specific holidays with a linear trend. The results indicate that Prophet demonstrates moderate forecasting accuracy, with varying degrees of effectiveness depending on the preprocessing applied.

The best configuration for Prophet involved Yeo-Johnson transformation and Min-Max scaling (Table 4). It is noteworthy that no transformation with no scaling produced slightly worse results, with a MAPE of 0.1949 and a higher R^2 of 0.6378, indicating that for Prophet, scaling does not uniformly improve performance.

The results emphasize that data preprocessing significantly influences Prophet's performance. The inclusion of monthly seasonality and holiday effects appears to have improved the model's ability to capture key trends in the time series. The 95% confidence intervals in the forecast plot (Figure 23) highlight Prophet's ability to quantify uncer-

Transformation	Scaler	MAPE	R^2	Theils U	Composite Score
yeo-johnson	minmax	0.1791	0.5459	0.1365	0.0104
none	none	0.1949	0.6378	0.1159	0.0108
none	minmax	0.1947	0.6351	0.1164	0.0109
none	robust	0.2157	0.6336	0.1150	0.0245
none	standard	0.2202	0.6361	0.1143	0.0272
yeo-johnson	robust	0.2492	0.5927	0.1192	0.0499
yeo-johnson	standard	0.2574	0.5805	0.1199	0.0560
yeo-johnson	none	0.9365	-5.2210	0.8913	1.0000

Table 4: Forecast performance metrics comparison

tainty effectively, even though the intervals tend to widen during high-variance periods.

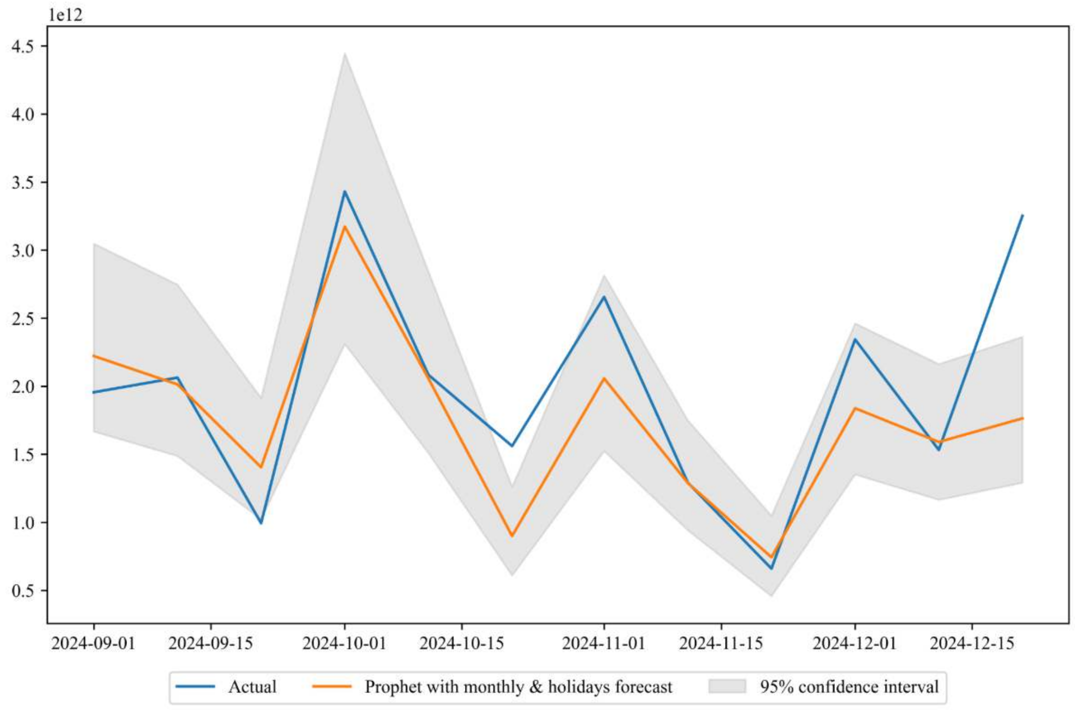


Figure 23: Prophet model forecast and confidence intervals

While Prophet does not outperform SARIMA in terms of R^2 and MAPE for the tested configurations, its robustness and flexibility make it a valuable tool for time series forecasting.

4.3 Exponential Smoothing

Exponential Smoothing with seasonality and linear trend (Holt-Winter) model resulted in less competitive compared to SARIMA and Prophet models (Table 5). No transformation with standard, robust, or Min-Max scaling produced almost the same results with slightly higher MAPE values and consistent R^2 values around 0.3541.

The confidence intervals provided by the model reflect its uncertainty, as depicted in the forecast plot (Figure 24).

Transformation	Scaler	MAPE	R^2	Theils U	Composite Score
yeo-johnson	minmax	0.3254	0.3634	0.1445	0.0005
none	standard	0.3524	0.3540	0.1421	0.0307
none	robust	0.3524	0.3541	0.1421	0.0307
none	minmax	0.3524	0.3541	0.1421	0.0307
none	none	0.3609	0.3222	0.1447	0.0419
yeo-johnson	robust	0.3809	0.1765	0.1586	0.0722
yeo-johnson	standard	0.3840	0.1700	0.1590	0.0761
yeo-johnson	none	0.9164	-4.9996	0.8602	1.0000

Table 5: Forecast performance metrics comparison

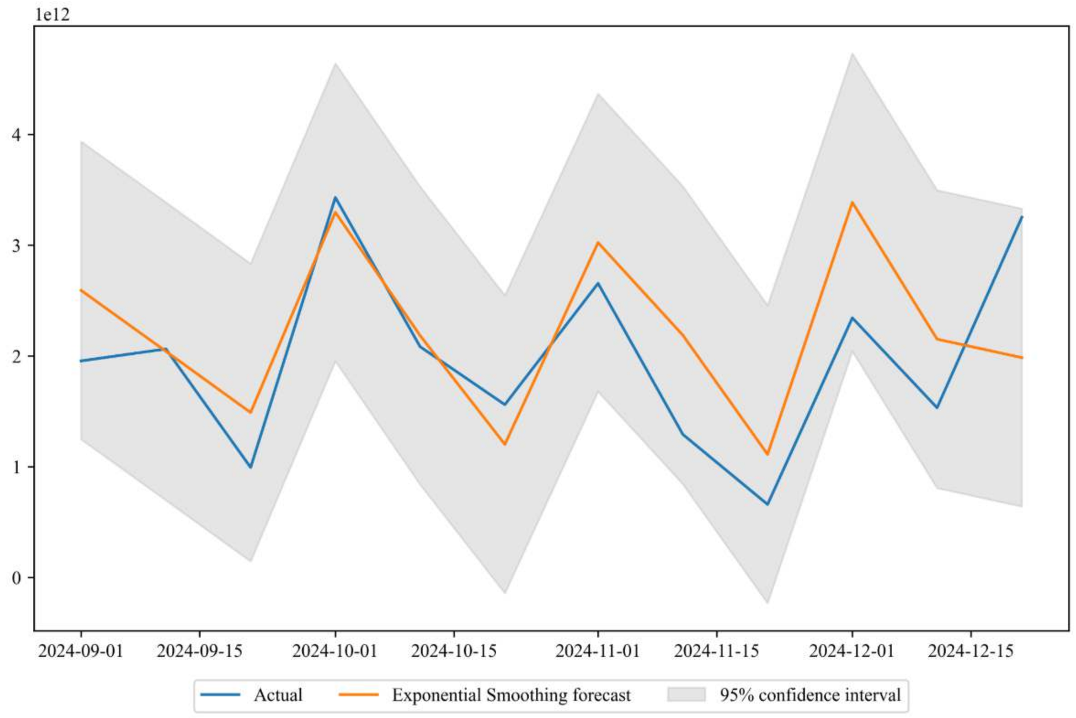


Figure 24: Exponential Smoothing model forecast and confidence intervals

While Exponential Smoothing offers a straightforward approach to forecasting, it underperforms compared to SARIMA and Prophet models in terms of R^2 and MAPE. Nevertheless, the method's simplicity and robustness make it a useful benchmark in forecasting experiments.

4.4 Random Forest

Random forest was selected as one of the forecasting models due to its robustness in capturing nonlinear relationships and its ensemble learning approach, which reduces overfitting. The model's hyperparameters were optimized using a two-step process involving GridSearchCV and Optuna. In the first step, GridSearchCV was employed to conduct an exhaustive search over a predefined parameter grid using TimeSeriesSplit for cross-validation. This provided a strong initial set of hyperparameters. The second step refined these hyperparameters further using Optuna, which explores a broader parameter space in a trial-based optimization framework.

To ensure stability and reduce variability in the forecast, an ensemble of the top-performing models was created by averaging their predictions. The composite score plot (Figure 25) highlights how the inclusion of additional models affects the overall metrics, showing the $1 - R^2$ metric so that it is consistent with other error measures (e.g., MAPE and Theil's U) where lower values are better. This consistency underscores the reliability of the composite score as a unifying metric for model selection.

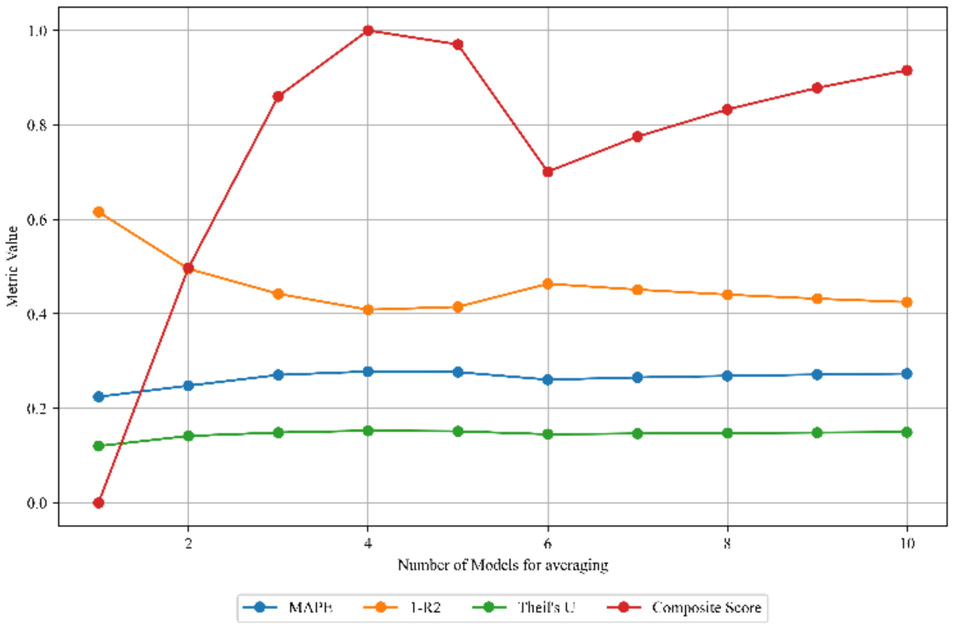


Figure 25: Composite score and performance metrics of averaged models

The performance metrics for the Random Forest model, as shown in Table 6, reveal the following:

- *Best Transformation and Scaler Combination:* The combination of the Yeo-Johnson transformation and the Min-Max scaler achieved the best composite score (0.0114), reflecting the most balanced performance across MAPE, R^2, and Theil's U.
- *Grid Search vs. Optuna:* In this specific case, the best-performing model came from GridSearchCV rather than Optuna. However, this is not a general result and might differ depending on the data and parameter space.

Transformation	Scaler	MAPE	R^2	Theils U	Composite Score
yeo-johnson	robust	0.2235	0.6108	0.1205	0.0129
yeo-johnson	minmax	0.2244	0.6156	0.1199	0.0153
yeo-johnson	standard	0.2264	0.6046	0.1214	0.0821
none	standard	0.2621	0.4988	0.1350	0.9998

Table 6: Forecast performance metrics comparison

The forecast plot (Figure 26) demonstrates the actual values, mean predictions, and 95% prediction intervals for the Random Forest model. The prediction intervals were computed as the 2.5th and 97.5th percentiles of tree predictions, providing a measure of uncertainty around the forecasts. The Random Forest model's flexibility in combining models with diverse parameter sets ensures that its predictions are robust and less sensitive to individual model biases.

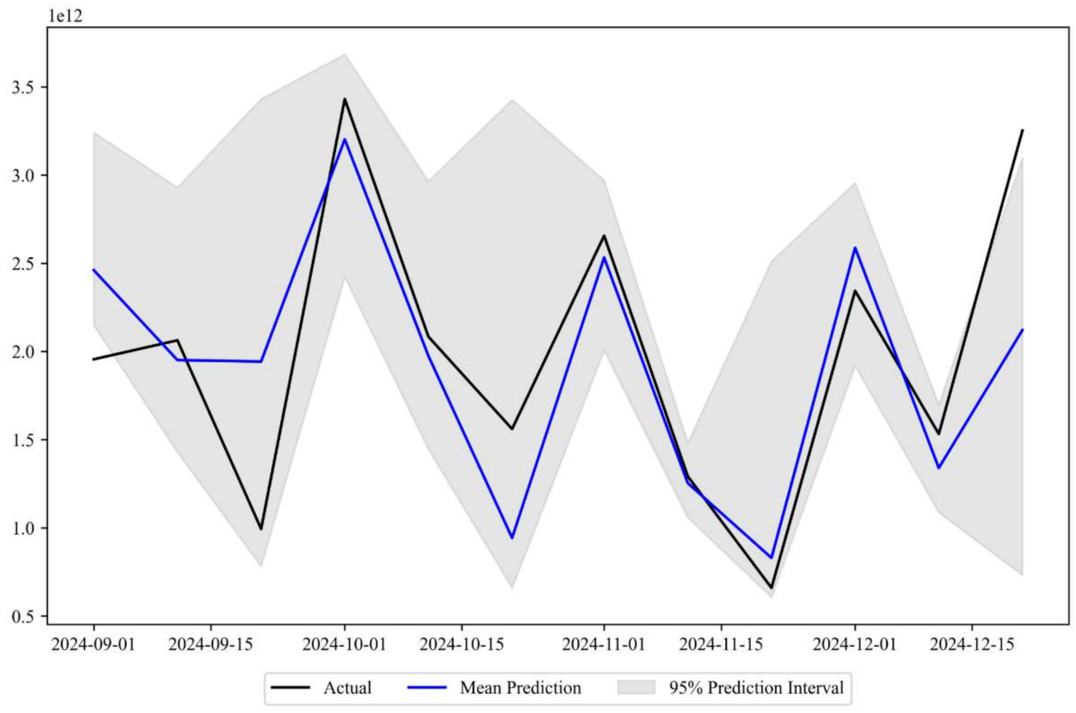


Figure 26: Random Forest Regressor model forecast and confidence intervals

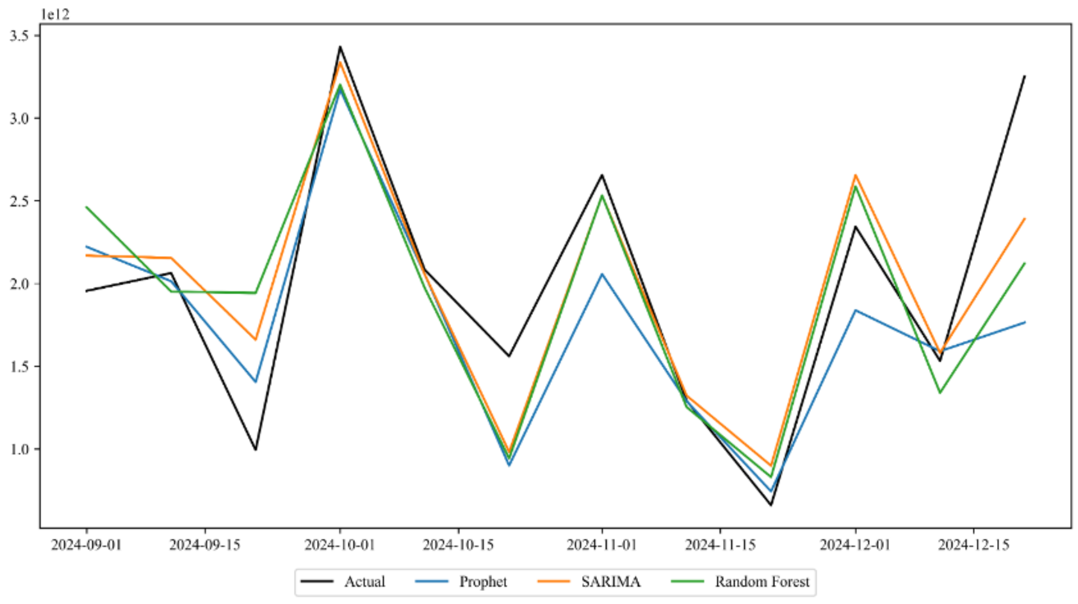


Figure 27: Multiple models forecasts comparison

In Figure 27, the comparison of better-performing models is presented. The plot illustrates the actual values alongside the forecasts generated by the SARIMA, Prophet, and Random Forest models. This visual comparison provides insights into how well each model captures the patterns and variability in the data.

4.5 Neural Networks

Although neural network-based models are theoretically and practically promising for time series forecasting, our experiments with GRU and LSTM units applied after wavelet transforms showed relatively poorer performance compared to other models. Specifically, comparatively better results were achieved for GRUs when no power transformation was applied with a standard scaler, resulting in a MAPE of 0.357, $R^2 = -0.2221$, and Theil's U = 0.2536, with a lookback window of 5 (Figure 28). For the wavelet-transformed data with LSTM (WNN), the best configuration included a robust scaler, sym8 wavelet, 5 levels of denoising, soft threshold rule, 3 levels of lookback, square-root transformation, and no inclusion of technical indicators. This setup produced a MAPE of 0.4411, $R^2 = -0.3868$, and Theil's U = 0.2392 (Figure 29). One of the key reasons for their unacceptable performance is the negative R^2, which indicates that the models' predictions on average were worse than simply guessing the mean of the series.

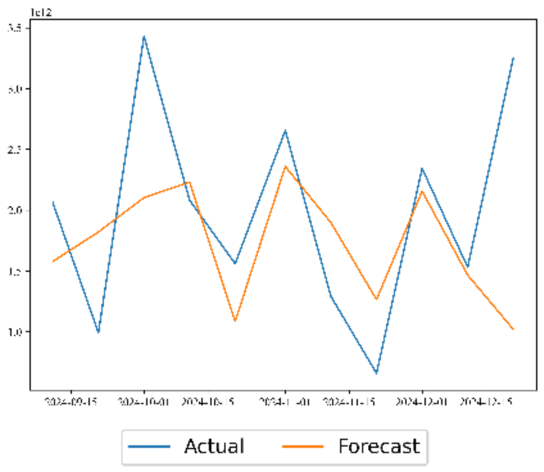


Figure 28: GRU model forecast

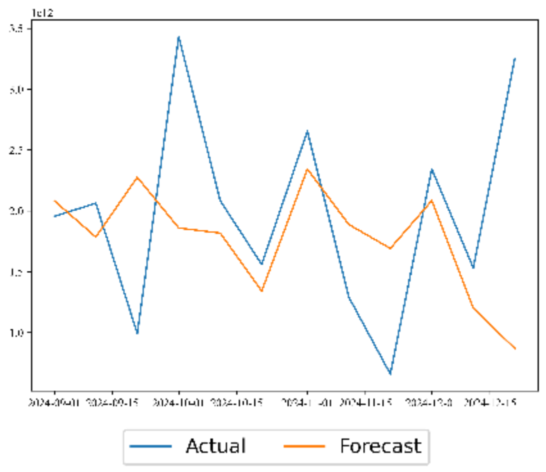


Figure 29: WNN model forecast

This may be attributed to the complexity involved in configuring the architecture of neural networks. Determining the optimal set-up for the model specs, such as the number of hidden layers (depth), the number of nodes (width), activation functions, batch sizes, loss functions, and combinations of various neural network types (e.g., multilayer, convolutional, recurrent, gated, long short-term memory, and their combinations) lacks a universally accepted rule of thumb [63, 74] or predefined procedure. Instead, better calibration of these configurations typically rely on experience and trial-and-error experimentation [75], and often, deeper research is needed to understand the optimal parameter space and training dynamics [76], which can be time-intensive and challenging to optimize effectively. Further experimenting with different architectures of neural networks is planned for the next steps of the research.

4.6 Ensemble learning

The ensemble learning approach implemented in this study illustrates the effectiveness of combining forecasts from multiple models, each employing different underlying methodologies, to improve the accuracy and robustness of predictions. The ensemble methods tested include weighted blending, simple blending (simple average of the model forecasts), stacking, and inverse expected variance, each utilizing outputs from SARIMA, Prophet, and Random Forest models because of their relatively good performances.

Weighted Blending. The weights for combining individual model predictions are determined based on their train MAE (Mean Absolute Error), where lower MAE values correspond to higher weights. The blended forecast formula is given as:

$$\text{Blended Forecast} = w_1 \cdot \text{SARIMA} + w_2 \cdot \text{Prophet} + w_3 \cdot \text{Random Forest}.$$

Stacking combines the forecasts from individual models (SARIMA, Prophet, Random Forest) as input features and uses Linear Regression as a meta-model to learn the optimal combination of these forecasts. While stacking offers an alternate approach to blending, the results indicate higher MAPE and lower R^2 compared to weighted blending, emphasizing the importance of the choice of the ensemble strategy.

The inverse expected variance method assigns ensemble weights using the normalized inverses of the forecasted model variances. The advantage of this approach is that the weights are no longer constant; instead, when a member model exhibits higher uncertainty in its prediction for a particular period — reflected by a higher variance — it receives a lower weight in the averaging process:

$$\text{Ensemble Forecast}_i = \frac{1}{\sigma_{1,i}^2} \cdot \text{SARIMA}_i + \frac{1}{\sigma_{2,i}^2} \cdot \text{Prophet}_i + \frac{1}{\sigma_{3,i}^2} \cdot \text{Random Forest}_i.$$

where $\frac{1}{\sigma_{k,i}^2}$ is the normalized inverse variance of kth model in the ith period of forecasting.

This method ensures that models with higher confidence (lower variance) have a stronger influence on the final prediction, dynamically adjusting model contributions based on forecast uncertainty.

Ensemble method	MAPE	R^2	Theils U	Composite Score
Inverse variance	0.1658	0.6624	0.2592	0.1904
Simple blending	0.1731	0.6908	0.2524	0.2020
Weighted blending	0.1818	0.6753	0.2527	0.2685
Stacking	0.2753	0.4912	0.1346	0.8333

Table 7: Forecast performance metrics comparison

From the analysis (Table 7), **inverse variance** emerges as the most effective ensemble method (Figure 30), outperforming even the SARIMA in terms of MAPE, achieving the lowest MAPE among other ensemble models. This indicates its superior capability to combine the strengths of SARIMA, Prophet, and Random Forest models for enhanced forecasting accuracy. Moreover, the Diebold-Mariano test for predictive accuracy [77], based on the mean absolute error on the out-of-sample forecast, indicates that this ensemble method demonstrates a statistically significant difference compared to the individual best-performing SARIMA model.

The stacking method, while theoretically robust, showed relatively weaker performance in this context, highlighting the challenges of optimal meta-model selection for this ensemble framework.

The visual comparisons in the provided plots (Figure 31) further supports these findings, demonstrating the practical advantages of the ensemble approaches in reducing forecasting errors and achieving more robust predictions.

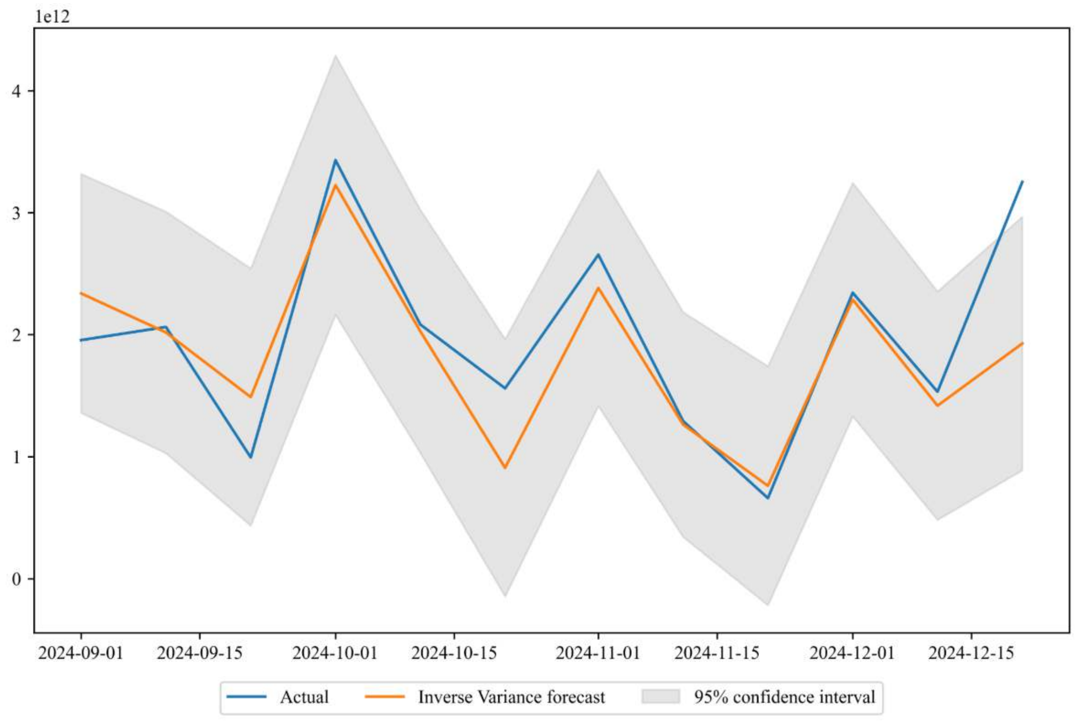


Figure 30: Inverse of expected variance ensemble forecast and confidence intervals

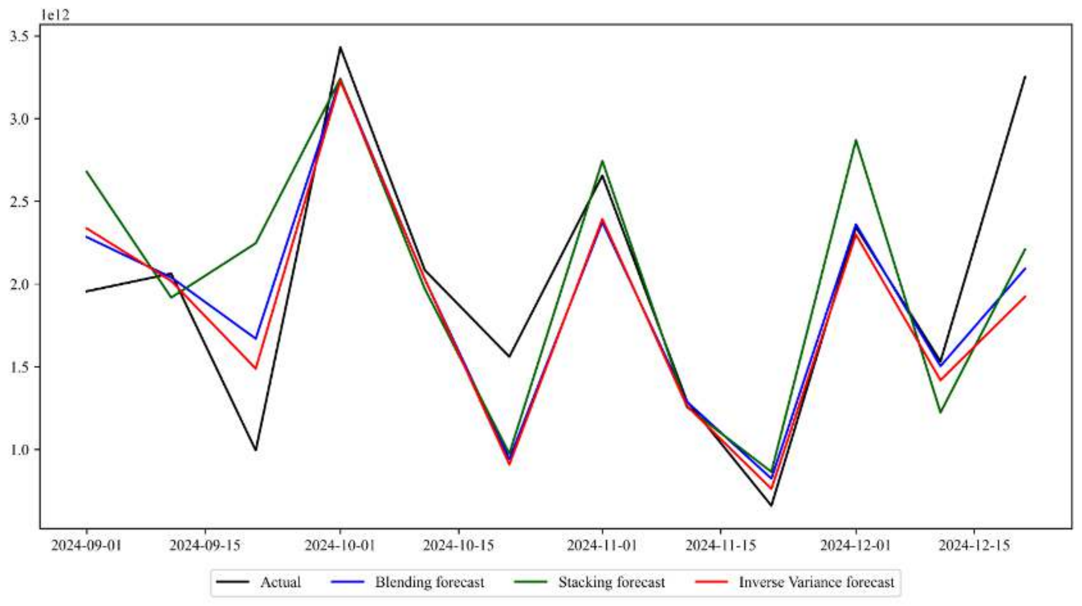


Figure 31: Ensemble methods comparison

5 Conclusion

This study conducted a "model horse racing" on some of the most widely used econometric models for time series forecasting and machine learning models for regression and sequential prediction problems to tackle the banking system liquidity forecasting problem, utilizing different data preprocessing techniques. The results of the experiments were demonstrated only through one component of the autonomous factors of liquidity, namely the cash outside the central bank, for the sake of conciseness and due to sensitive data issues. The best-performing individual model was SARIMA, which was outperformed by the ensemble model employing inverse of expected variance of SARIMA, Prophet, and Random Forest.

Despite the current relatively poor performance of neural network models, we maintain confidence that they hold significant potential for time series forecasting, which was not fully realized in this research owing to challenges in tuning their architectures and constraints regarding to experimenting with hyperparameter optimization. Moreover, the study focused mainly on local univariate models, while not utilizing the advantages offered by multivariate models and economic structural elements.

Future research will address these limitations by conducting experiments with more complex neural network architectures, and optimizing their configurations with more advanced search techniques, and extending the model base to include multivariate models and incorporating structural elements, finally, exploring other promising ensemble methods to tackle the bias-variance trade-off. By addressing the aforementioned limitations and proceeding with the proposed future directions, this continuation of research holds promise for contributing further to the field of monetary policy and financial forecasting.

The implementation of the project in Python with the **source code** is available in this *GitHub* repository: `https://github.com/Shahzod-CBU/BCC_liquidity`

References

[1] C. E. V. Borio, *The implementation of monetary policy in industrial countries: a survey*, ser. BIS economic papers. Basel: Bank for International Settlements, Monetary and Economic Dep, 1997, no. 47.

[2] U. Bindseil, *Monetary policy implementation: theory, past, present.* Oxford: Oxford University Press, 2009.

[3] A. K. Kashyap, R. G. Rajan, and J. C. Stein, "Banks as Liquidity Providers: An Explanation for the Co-Existence of Lending and Deposit-Taking," *SSRN Electronic Journal*, 1999. [Online]. Available: http://www.ssrn.com/abstract=156748

[4] A. M. Markus Burger, Bernhard Klar and G. Schindlmayr, "A spot market model for pricing derivatives in electricity markets," *Quantitative Finance*, vol. 4, no. 1, pp. 109–122, 2004. [Online]. Available: https://doi.org/10.1088/1469-7688/4/1/010

[5] S. Smyl, "A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting," *International Journal of Forecasting*, vol. 36, no. 1, pp. 75–85, 2020, m4 Competition. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207019301153

[6] C. A. Sims, "Macroeconomics and Reality," *Econometrica*, vol. 48, no. 1, p. 1, Jan. 1980. [Online]. Available: https://www.jstor.org/stable/1912017

[7] H. Lütkepohl, *New Introduction to Multiple Time Series Analysis.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. [Online]. Available: http://link.springer.com/10.1007/978-3-540-27752-1

[8] J. H. Stock and M. W. Watson, "Forecasting Using Principal Components From a Large Number of Predictors," *Journal of the American Statistical Association*, vol. 97, no. 460, pp. 1167–1179, Dec. 2002. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1198/016214502388618960

[9] M. Bańbura, D. Giannone, and L. Reichlin, "Large Bayesian vector auto regressions," *Journal of Applied Econometrics*, vol. 25, no. 1, pp. 71–92, Jan. 2010. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/jae.1137

[10] W. Bao, J. Yue, and Y. Rao, "A deep learning framework for financial time series using stacked autoencoders and long-short term memory," *PLOS ONE*, vol. 12, no. 7, p. e0180944, Jul. 2017. [Online]. Available: https://dx.plos.org/10.1371/journal.pone.0180944

[11] H. R. Varian, "Big Data: New Tricks for Econometrics," *Journal of Economic Perspectives*, vol. 28, no. 2, pp. 3–28, May 2014. [Online]. Available: https://pubs.aeaweb.org/doi/10.1257/jep.28.2.3

[12] O. Sagi and L. Rokach, "Ensemble learning: A survey," *Wiley interdisciplinary reviews: data mining and knowledge discovery*, vol. 8, no. 4, p. e1249, 2018.

[13] M. L. Bech and R. Garratt, "The intraday liquidity management game," *Journal of Economic Theory*, vol. 109, no. 2, pp. 198–219, Apr. 2003. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0022053103000164

[14] A. N. Berger and C. H. S. Bouwman, "Bank Liquidity Creation," *Review of Financial Studies*, vol. 22, no. 9, pp. 3779–3837, Sep. 2009. [Online]. Available: https://academic.oup.com/rfs/article-lookup/doi/10.1093/rfs/hhn104

[15] J. D. Hamilton, *Time series analysis.* Princeton, N.J: Princeton University Press, 1994.

[16] D. A. Dickey and W. A. Fuller, "Distribution of the Estimators for Autoregressive Time Series With a Unit Root," *Journal of the American Statistical Association*, vol. 74, no. 366, p. 427, Jun. 1979. [Online]. Available: https://www.jstor.org/stable/2286348

[17] R. F. Engle, "Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation," *Econometrica*, vol. 50, no. 4, pp. 987–1007, 1982. [Online]. Available: https://www.jstor.org/stable/1912773

[18] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*, 2nd ed. Melbourne: OTexts, 2018.

[19] G. E. P. Box and D. R. Cox, "An Analysis of Transformations," *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 26, no. 2, pp. 211–243, Jul. 1964. [Online]. Available: https://academic.oup.com/jrsssb/article/26/2/211/7028064

[20] I. Yeo and R. A. Johnson, "A new family of power transformations to improve normality or symmetry," *Biometrika*, vol. 87, no. 4, pp. 954–959, Dec. 2000. [Online]. Available: https://doi.org/10.1093/biomet/87.4.954

[21] J. Brownlee, *Better deep learning: train faster, reduce overfitting, and make better predictions.* Machine learning mastery, 2018.

[22] R. J. Hodrick and E. C. Prescott, "Postwar U.S. Business Cycles: An Empirical Investigation," *Journal of Money, Credit and Banking*, vol. 29, no. 1, p. 1, Feb. 1997. [Online]. Available: https://www.jstor.org/stable/2953682

[23] G. Elliott, T. J. Rothenberg, and J. H. Stock, "Efficient Tests for an Autoregressive Unit Root," *Econometrica*, vol. 64, no. 4, p. 813, Jul. 1996. [Online]. Available: https://www.jstor.org/stable/2171846

[24] K. Benidis, S. S. Rangapuram, V. Flunkert, Y. Wang, D. Maddix, C. Turkmen, J. Gasthaus, M. Bohlke-Schneider, D. Salinas, L. Stella, F.-X. Aubet, L. Callot, and T. Januschowski, "Deep Learning for Time Series Forecasting: Tutorial and Literature Survey," *ACM Computing Surveys*, vol. 55, no. 6, pp. 1–36, Jul. 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/3533382

[25] G. E. P. Box and G. M. Jenkins, *Time series analysis: forecasting and control*, ser. Holden-Day series in time series analysis and digital processing. Oakland: Holden-Day, 1979.

[26] P. H. Franses and D. Van Dijk, "The forecasting performance of various models for seasonality and nonlinearity for quarterly industrial production," *International Journal of Forecasting*, vol. 21, no. 1, pp. 87–102, Jan. 2005. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0169207004000524

[27] M. G. Garcia, M. C. Medeiros, and G. F. Vasconcelos, "Real-time inflation forecasting with high-dimensional models: The case of Brazil," *International Journal of Forecasting*, vol. 33, no. 3, pp. 679–693, Jul. 2017. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0169207017300262

[28] P. C. Besse, H. Cardot, and D. B. Stephenson, "Autoregressive Forecasting of Some Functional Climatic Variations," *Scandinavian Journal of Statistics*, vol. 27, no. 4, pp. 673–687, Dec. 2000. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1111/1467-9469.00215

[29] Y. Lai and D. A. Dzombak, "Use of the Autoregressive Integrated Moving Average (ARIMA) Model to Forecast Near-Term Regional Temperature and Precipitation," *Weather and Forecasting*, vol. 35, no. 3, pp. 959–976, Jun. 2020. [Online]. Available: https://journals.ametsoc.org/doi/10.1175/WAF-D-19-0158.1

[30] S. Shen, G. Li, and H. Song, "Effect of Seasonality Treatment on the Forecasting Performance of Tourism Demand Models," *Tourism Economics*, vol. 15, no. 4, pp. 693–708, Dec. 2009. [Online]. Available: https://journals.sagepub.com/doi/10.5367/000000009789955116

[31] P. Sudharma, R. Hafidh, M. S. Sharif, and W. Elmedany, "Neoj4 and SARIMAX Model for Optimizing Product Placement and Predicting the Shortest Shopping Path," in *2023 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT).* Sakheer, Bahrain: IEEE, Nov. 2023, pp. 433–439. [Online]. Available: https://ieeexplore.ieee.org/document/10391451/

[32] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time series analysis: forecasting and control*, 4th ed., ser. Wiley series in probability and statistics. Hoboken, N.J: John Wiley, 2008, oCLC: ocn176895531.

[33] J. D. Cryer and K. Chan, *Time series analysis: with applications in R*, 2nd ed., ser. Springer texts in statistics. New York: Springer, 2008, oCLC: ocn191760003.

[34] F. Canova and B. E. Hansen, "Are Seasonal Patterns Constant Over Time? A Test for Seasonal Stability," *Journal of Business & Economic Statistics*, vol. 13, no. 3, pp. 237–252, Jul. 1995. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/07350015.1995.10524598

[35] R. J. Hyndman and Y. Khandakar, "Automatic Time Series Forecasting: The **forecast** Package for *R*," *Journal of Statistical Software*, vol. 27, no. 3, 2008. [Online]. Available: http://www.jstatsoft.org/v27/i03/

[36] P. Chaussé, "Computing Generalized Method of Moments and Generalized Empirical Likelihood with *R*," *Journal of Statistical Software*, vol. 34, no. 11, 2010. [Online]. Available: http://www.jstatsoft.org/v34/i11/

[37] B. Ghosh, B. Basu, and M. O'Mahony, "Bayesian Time-Series Model for Short-Term Traffic Flow Forecasting," *Journal of Transportation Engineering*, vol. 133, no. 3, pp. 180–189, Mar. 2007.

[38] S. J. Taylor and B. Letham, "Forecasting at scale," Sep. 2017. [Online]. Available: https://peerj.com/preprints/3190v2

[39] S. Servadio, R. Zanetti, and R. Armellin, "Maximum a posteriori estimation of hamiltonian systems with high order series expansions," in *Proceedings of the AAS/AIAA Astrodynamics Specialist Conference held August 11–15, 2019, Portland, Maine, USA*, vol. 171, 2019, pp. 2843–2858.

[40] H. Weytjens, E. Lohmann, and M. Kleinsteuber, "Cash flow prediction: MLP and LSTM compared to ARIMA and Prophet," *Electronic Commerce Research*, vol. 21, no. 2, pp. 371–391, Jun. 2021. [Online]. Available: https://link.springer.com/10.1007/s10660-019-09362-7

[41] L. Menculini, A. Marini, M. Proietti, A. Garinei, A. Bozza, C. Moretti, and M. Marconi, "Comparing Prophet and Deep Learning to ARIMA in Forecasting Wholesale Food Prices," *Forecasting*, vol. 3, no. 3, pp. 644–662, Sep. 2021. [Online]. Available: https://www.mdpi.com/2571-9394/3/3/40

[42] M. Ibrahim, A. Alsheikh, F. Awaysheh, and M. Alshehri, "Machine Learning Schemes for Anomaly Detection in Solar Power Plants," *Energies*, vol. 15, no. 3, p. 1082, Feb. 2022. [Online]. Available: https://www.mdpi.com/1996-1073/15/3/1082

[43] C. C. Holt, "Planning production, inventories, and work force." 1960.

[44] P. R. Winters, "Forecasting sales by exponentially weighted moving averages," *Management science*, vol. 6, no. 3, pp. 324–342, 1960.

[45] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 2nd ed. OTexts, 2018. [Online]. Available: https://otexts.com/fpp2/

[46] T. G. Dietterich and E. B. Kong, “Machine learning bias, statistical bias, and statistical variance of decision tree algorithms,” 1995.

[47] F. Esposito, D. Malerba, G. Semeraro, and J. Kay, “A comparative analysis of methods for pruning decision trees,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 5, pp. 476–491, 1997.

[48] L. Breiman, “Bagging predictors,” *Machine learning*, vol. 24, pp. 123–140, 1996.

[49] L. Rokach, “Decision forest: Twenty years of research,” *Information Fusion*, vol. 27, pp. 111–125, 2016.

[50] L. Breiman, “Random forests,” *Machine learning*, vol. 45, pp. 5–32, 2001.

[51] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A next-generation hyperparameter optimization framework,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 2623–2631. [Online]. Available: https://doi.org/10.1145/3292500.3330701

[52] F. Chollet, *Deep learning with Python.* Shelter Island, New York: Manning Publications Co, 2018.

[53] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[54] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, “LSTM: A search space odyssey,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2017.

[55] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, “State-of-the-art in artificial neural network applications: A survey,” *Heliyon*, vol. 4, no. 11, 2018.

[56] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, M. Hasan, B. C. Van Essen, A. A. Awwal, and V. K. Asari, “A state-of-the-art survey on deep learning theory and architectures,” *Electronics*, vol. 8, no. 3, p. 292, 2019.

[57] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[58] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, “Advances in optimizing recurrent networks,” in *2013 IEEE International conference on acoustics, speech and signal processing.* IEEE, 2013, pp. 8624–8628.

[59] J. Martens and I. Sutskever, “Learning recurrent neural networks with hessian-free optimization,” in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 1033–1040.

[60] M. Nielsen, *Neural networks and deep learning.* Determination Press, 2015.

[61] K. Cho, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[62] C. Peng, Y. Li, Y. Yu, Y. Zhou, and S. Du, "Multi-step-ahead host load prediction with gru based encoder-decoder in cloud computing," in *2018 10th International Conference on Knowledge and Smart Technology (KST)*, 2018, pp. 186–191.

[63] J. Chan Phooi M'ng and M. Mehralizadeh, "Forecasting east asian indices futures via a novel hybrid of wavelet-pca denoising and artificial neural network models," *PLOS ONE*, vol. 11, no. 6, pp. 1–29, 06 2016. [Online]. Available: https://doi.org/10.1371/journal.pone.0156338

[64] F. O'Donncha, Y. Zhang, B. Chen, and S. C. James, "An integrated framework that combines machine learning and numerical models to improve wave-condition forecasts," *Journal of Marine Systems*, vol. 186, pp. 29–36, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924796317304827

[65] Z. Wang, Y. Wang, and R. S. Srinivasan, "A novel ensemble learning approach to support building energy use prediction," *Energy and Buildings*, vol. 159, pp. 109–122, 2018.

[66] X. Sun, J. Yin, and Y. Zhao, "Using the inverse of expected error variance to determine weights of individual ensemble members: application to temperature prediction," *Journal of Meteorological Research*, vol. 31, no. 3, pp. 502–513, 2017.

[67] E. Altay and M. H. Satman, "Stock market forecasting: artificial neural network and linear regression comparison in an emerging market," *Journal of Financial Management & Analysis*, vol. 18, no. 2, p. 18, 2005.

[68] T. J. Hsieh, H. F. Hsiao, and W. C. Yeh, "Forecasting stock markets using wavelet transforms and recurrent neural networks: An integrated system based on artificial bee colony algorithm," *Applied soft computing*, vol. 11, no. 2, pp. 2510–2525, 2011.

[69] Z. Guo, H. Wang, Q. Liu, and J. Yang, "A feature fusion based forecasting model for financial time series," *PloS one*, vol. 9, no. 6, p. e101113, 2014.

[70] M. Kane and S. M. Case, "The reliability and validity of weighted composite scores," *Applied Measurement in Education*, vol. 17, no. 3, pp. 221–240, 2004.

[71] J. Perktold, S. Seabold, and J. Taylor, "Construct confidence interval for the fitted parameters," https://www.statsmodels.org/dev/generated/statsmodels.tsa.statespace.sarimax.SARIMAXResults.conf_int.html.

[72] Facebook's Core Data Science team, "Uncertainty intervals," https://facebook.github.io/prophet/docs/uncertainty_intervals.html.

[73] P. C. M. Filho, "Confidence intervals for the random forest generalization error," *Pattern Recognition Letters*, vol. 158, pp. 171–175, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167865522001416

[74] J. M. Zurada, *Introduction to artificial neural systems.* St. Paul: West, 1992.

[75] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi *et al.*, “DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning,” *arXiv preprint arXiv:2501.12948*, 2025.

[76] X. Bi, D. Chen, G. Chen, S. Chen, D. Dai, C. Deng, H. Ding, K. Dong, Q. Du, Z. Fu *et al.*, “DeepSeek LLM: Scaling open-source language models with Longtermism,” *arXiv preprint arXiv:2401.02954*, 2024.

[77] F. X. Diebold and R. S. Mariano, “Comparing predictive accuracy,” *Journal of Business & economic statistics*, vol. 20, no. 1, pp. 134–144, 2002.