**Richard Lesko**

# Comparing Machine Learning and Traditional Statistical Methods for Gold price Prediction

# Executive Summary

First model - ARIMA, which is based solely on statistical theory yielded RMSE and MAE of 14.86 and 9.96 respectively. The increase from test to train error suggests, that algorithm successfully generalizes on unseen data. The ARIMA model assumes linear relationship between data points, but seems to generalize relatively well, since average daily variation between gold prices is around 10. There are many external macroeconomic factors that influence gold, which might often be unable to be projected in numbers. However, ARIMA model seems to perform well taking into consideration only past gold prices and errors. We might assume, that the best factor to predict value of financial instrument is its yesterday's value.

Out of all models LASSO yielded in highest test and training error. LASSO also assumes linear relationship and applies penalty on coefficients, that have lowest predictive power, setting them to exact 0. Compared to regular regression it performs better in high-dimensional space, that this thesis is dealing with. However, it might struggle in non-linear space. LASSO yielded test RMSE of 21.6, which is almost double the value of daily gold price variations. Big gap between training and testing error also indicates, that model is struggling with generalizing on unseen data. Using LASSO method might in this case lead to oversimplification of real patterns.

Random Forests is one of the two models in this thesis, that can capture non-linear relationships. It yielded RMSE of 18.94. The function that I used to display most used predictors in decision split indicates, that only few variables are carrying predictive power. Due to this observation, we might assume that model was unnecessary complex, and thus could have led to better predictive performance, if further simplified.

The last model – XGBoost resulted in RMSE of 16.68 and MAE of 11.53, which might be considered decent performance. The model complexity is here controlled by imposing penalties for being too complex. The model is of course able to capture non-linear relationships and maintains high predictive performance in high-dimensional data. Both error metrics are considered decent, since average daily variation is only 10.

In conclusion this thesis compared four different predictive models – ARIMA, LASSO, Random Forests and XGboost. First two assumes linear relationship linear relationships between predictors and target variable, while Random Forests and XGboost also take non-linearity into consideration. Analysis of the models showed that the worst performing is

LASSO. This might happen, because model assumes linearity, and since 28 predictors were part of the dataset, it resulted in poor predictive performance. The second worst performing model was Random Forests, however here we might assume, that if the input data were simplified, model could have yielded in lower testing error, but RMSE of 18.94 still might be considered decent performance. The next model based on performance is XGBoost, which yielded very decent RMSE of 16.68. The model generalizes well on unseen data, resulting in relatively lower error, taking average variation in gold price into consideration. The best performing model according to this thesis is ARIMA. ARIMA yielded in decent RMSE of 14.86. The fact, that ARIMA model is the best performing might be surprising, but also might be considered as evidence, that best predictor for tomorrow's gold price is today's price. Gold price has increasing trend over long period of time, and even though there are sudden spikes, they do not happen very often, which supports the fact that lagged price is best predictor.

# Abstract

Gold is as a chemical element with the symbol AU, which belongs to the metal chemical element group. However, this thesis examines gold from a financial rather than a chemical perspective. We can assume that gold was the first instrument used to store the value and for other exchanging purposes. Ever since, gold is an instrument that is still widely used to store value, its demand shifts when economic recessions or instability is expected in the future, which affects the price. It is still considered financial instrument that relatively stable and not volatile. Countries might use this instrument to prepare for upcoming recession or to simply store its cash resources in gold, hence its price plays important role when it comes to countries' political or economic decisions. Knowing gold's exact future price therefore plays crucial role for investment decisions. Knowing the exact future price is of course impossible, but having a reliable estimate may be very useful for policymakers or investors. Machine Learning and Traditional Statistical methods are well-suited for estimating future gold price. Traditional Statistical methods are grounded by statistical and mathematical theory, machine learning models can identify non-linear patterns. By conducting predictive analysis using both type of models, this thesis comprehensively explains selected predictive models, identifies their pros and cons and compares performance of selected models

# Problem statement

Gold is considered as save-heaven resource during economic instability, its price is influenced by various macro-economic factors including exchange rates, interest rates, performance of stock market or economic state of greatest world's economies. Having reliable estimates of gold's future price allows investors, policymakers or financial institutions to be well prepared for the future economic development. However, having reliable estimates remains complex task due to political instability, non-linear behavior of predictors, unforeseeable economic events or having all factors influencing gold price included in the analysis. Several models will be used to conduct predictive modeling – ARIMA, LASSO regression, Random Forests and XGBoost. First two models mentioned assumes linearity, while Random Forests and XGBoost can capture complex non-linear patterns. The aim of this thesis is to identify most efficient and best performing models for gold price prediction, while providing comprehensive understanding of all selected methods.

# Literature Review

To further support credibility of this thesis as a source of information I used various sources including research papers, articles or books.

As primary data collection method I used various official internet websites such as *Investing.com, Macrotrends* or *Statista.*

First step in preparing the data was identifying which predictors are relevant i.e. are carrying predictive power. To decide what predictors, carry predictive power I used Rolling Cross Validation method. The book *Forecasting: Principles and Practice* by *Hyndman* and *Anthanasopoulos* laid theoretical foundation for Rolling Cross Validation. To evaluate and observe how this method was applied in practice, I used research paper *Volatility forecasting using deep neural network with time-series embedding* written by *Chen, Wei-Jie & Yao, Jing-Jing & Shao, Yuanhai.* Although Rolling Cross Validation is a method based on which was decided which predictors will be kept, machine learning model had to determine which predictors will be used further. To select important predictors, elastic net method was used, which combines LASSO and Ridge regression. Theoretical foundations for both methods were described in research article *In Layman's terms, what is lasso and ridge regression?* by *Jaideep Negi.* To fully understand Elastic Net method, it was crucial to first gain knowledge about ridge and LASSO regressions.

To fully understand how Elastic Net method for feature selection works I have used several articles at *Medium* internet space, where research articles are published. Crucial for understanding Elastic Net were articles *Elastic Net Regression* by *Abishek Jain* and *Elastic Net Regression Guide* by *Shruti Dhumme.* To support the ideas and observation regarding Elastic Net throughout explanation was provided highly praised book - *Elements of Statistical Learning* written by *Trevor Hastie, Robert Tibshirani* and *Jerome Friedman.*

For in-depth understanding of statistical method ARIMA the book written by *Hyndman* and *Anthanasopoulos – Forecasting: Principles and Practice* has proven to be very useful explaining foundations of this method. For parameters comprehensive understanding and their tuning articles *A Guide to Parameter Tuning for Time Series Forecasting* by *Seyma Demir* and *Understand Arima and tune P,D,Q* by *Xiaoyu Sun* were utilized.

Main resources used to explain and conduct LASSO regression were lecture presentations, more specifically Lecture 3, course *Introduction to Machine Learning* and the book already mentioned; *Elements of Statistical Learning.*

Even though the book *Elements of Statistical Learning* mentions both Random Forests and XGBoost it does so only slightly, hence several articles and research papers had to be used. Firstly to gain understanding of ensemble methods and decision tree methods articles *Ensemble methods in Machine Learning* by *Dietterich* and *Time Series forecasting using Tree Based methods* written by *Houssainy A El, Amal Mohammed* and *Haitham Fawzy* provided full understanding of both topics.

Medium Article *Can Random Forests overfit* by *Alex Molas* and *Hyperparameters and tuning strategies for random forest* by *Probst, Wright* and *Boulsteix* provided further understanding of Random Forests method and how hyperparameters are tuned.

To gain in depth understanding of how XGBoost method works, and how it generates prediction several articles including *Tree Boosting With XGboost* by *Didrik Nielsen, XGBoost : A Scalable Tree Boosting System* by *Tianqi Chen* and *Carlos Guestrin* and *Understanding Boosting in Machine Learning* by *Brijesh Soni* employed.

# Methodology

This section will briefly explain what my approach to goal of the thesis is – comparing between various methods and their performance when it comes to predictive analysis.

**Data Collection**

The data were collected from various internet websites dealing with financial data including *Fred, Investing,* or *Macrotrends*. In the *Section,* I will reflect on importance of each dataset that I used as predictor in my analysis.

**Data Processing and Pre-Processing**

After all the data needed were obtained it was obvious that not all datasets have values correctly interpreted and presented. All datasets needed to be checked for missing values, outliers or inconsistent values. The main goal here was to have all the data combined into one merged dataset. I will touch upon data processing in *Section 2.*

**Model Selection**

Predictive data modeling must be conducted to identify overall performance and difference between various models. Firstly, in *Section 4.1* I have predicted gold price using ARIMA model, which is considered traditional statistical model, as it was developed long ago first machine learning models and it relies on time series analysis. In *Section 4.1* LASSO regression was used to predict gold price. LASSO regression could be considered as a bridge between traditional statistical models and machine learning, as it has foundations in statistics, however when used for predictive modeling it might also be considered machine learning technique. *Section 4.3* will introduce purely Machine Learning algorithm - Random Forest and *Section 4.4* will reflect on Extreme Gradient Boosting.

**Evaluation**

To assess which out of the four methods performs the best, outcome of each method will be compared based on error evaluation metrics. This section will identify which model provides us with the lowest prediction errors and highlight key factors contributing to model performance.

# Predicting the price

**Machine Learning Methods**

Machine Learning can be characterized as a tool, that deals with developing algorithms that enable computer to learn from data. Learning from data in this case means identifying some patterns. Algorithm draws patterns from the data that it is trained on, the data are used to train

the model are features, or in case of predictive analysis, ***predictors (X variables).*** Predictors are ***independent variables***, that have influence on ***dependent variable (Y variable)***, future gold price in this case. Simply put algorithm I will try to make predictions of Y-variable (gold price) based on X-variables (Silver daily prices, Inflation rates etc.)

If the algorithm is trained well using suitable method, it can come up with reliable outcomes, in this case future price of gold. Machine Learning models usually perform well in handling ***high dimensional data.*** High dimensional data are characterized by having large number of features, i.e. predictors (data, which algorithm uses to identify patterns). ***Non-linear data*** might be another issue when it comes to predictive analysis, relationship between this type of data cannot be accurately as some predictive models assume linear relationship. Another advantage of various machine learning algorithms is, that they can capture non-linear relationships. Many Machine learning models can perform ***variable selection***, meaning, that algorithm can determine importance of predictors. ***Multicollinearity*** is another challenge that predictive analysis might face, multicollinearity is characterized as interrelation between two or more predictors (X variables), affecting interpretability of the model. Based on mentioned challenges I chose to use algorithms that deal with these challenges well – **Random Forests** and **XGBoost,** as well as **LASSO** model.

**Traditional Statistical Methods**

Traditional Statistical Methods have been around for decades and are important foundation for data science in general. Its methods rely heavily on interpreting the data mathematically. Most of traditional statistical methods assume linearity between X and Y variables, thus they face many challenges when it comes non-linear or high-dimensional data. Traditional Statistical Methods are also sensitive to multicollinearity, which affects the outcome using these models. When trying to analyze patterns from the data, these methods rely on some kind of functional form, which can cause inaccuracy or high bias for real-world data structures. These methods use models that assume data structure and distribution. Despite facing many challenges, they are still regarded as valuable, especially with connection to machine learning methods. One of the traditional statistical methods is so called **ARIMA – Auto Regressive Integrated Moving Average**, which I will use as one of the models in this thesis. Another method that is based on statistical assumptions is LASSO regression, which will also be reflected upon in this thesis.

# 1. Choice of data

To be able to predict future gold price, we need to have some data, based on which algorithm can draw patterns that have influence on gold's spot price to some extent. Even though it is impossible to obtain all the data that may have some impact on the gold price, the more relevant data there is available for algorithm to learn from, the more accurate algorithm's predictions will be. In following section, I will reflect on each data i.e. predictors, that were used for predicting future gold price

**Commodities**

***Silver***

As *Finance Magnates* mentions in many of their articles, gold and silver are rare metals, which are considered as close substitutes. Although their relationship has experienced negative values in the past, gold and silver in general are known to have strong positive relationship over the course of past 40 years.

***Aluminum***

Aluminum is commonly used high-value industries, such as transportation or construction. It usually increases when economies are experiencing economic growth, mainly due to higher demand. Price of both gold and aluminum is determined by shared macroeconomic factors hence it can be assumed that there is some correlation between the two metals.

***Platinum***

Platinum is mostly used in automotive, electronic or aerospace sectors. Gold's price usually outperform platinum when economic stability and value of fiat money is experiencing downturn.

***Oil and Gasoline***

Commodities are sensitive to inflation in general, for instance, if transportation or mining costs rises (increased price of oil/gasoline), gold becomes more expensive as oil or gasoline are necessary for its transport and mining. Higher oil or gasoline prices may lead to higher inflation, which automatically makes gold more lucrative financial instrument to invest in. While these two examples are contradictory, they are pointing out how interdependent are these commodities and gold.

***Corn and Wheat***

As stated on the article by *CME Group* from 2022 inflation is measured by CPI (Consumer Price Index) i.e. average price over time. Two mostly used commodities in agricultural sector are corn and wheat, which makes them impactful on CPI. When it comes to transportation and production of these commodities, oil and gasoline are necessary, which means that both types of commodities are interrelated.

***Lumber***

Based on analysis by *FFR Trading* from 2023 lumber is essential commodity for construction industry, meaning that its price can be a reliable indicator of economic growth and stability. Lumber – Gold ratio is considered as important factor when it comes to evaluating health of economy. Here we might deal with an inverse relationship i.e. when lumber price decreases, leading to economic instability or recession, demand for gold is potentially increasing.

**Exchange Rates**

***USD/EUR Exchange rate***

As Eurozone is considered important player in the global gold market, I decided to include exchange rate between US Dollar and Euro as variable in predictive analysis. Generally stronger Euro compared to USD makes gold less expensive to European investors.

***USD/CHY and USD/INR***

Based on the *Statista,* I decided to include two exchange rates: USD to Chinese Yuan and USD to Indian Rupee because China and India are two world's biggest gold consumers. Both exchange rates may shift demand in the regiones, which might impact the gold spot price. The central banks of each region mentioned also influences the large volume gold purchases, which may shift the demand.

**Stock markets**

Even though stocks as a financial instrument have been outperforming gold over a long-horizon investment period, there have been some short-term periods where inverse relationship could be observed. Since 2020 gold has been outperforming stocks, the reason behind it might be that since 2020 world have been dealing with many economic instabilities or geopolitical tensions. Hence, performance of the stock market may be good indicator when it comes to forecasting gold price. Gold as a financial instrument tends to have increased demand in the periods of

economic instability, what affects its price, which article by *Investopedia* from 2024 also supports. Based on these facts, I decided to include some of the major stock markets.

***London Stock Exchange (LSE)***

Stock Exchange with headquarters in London, which is currently listing companies worth more than 3 trillion USD.

***New York Stock Exchange (NYSE)***

New York Stock Exchange is the largest stock exchange in the world with market capitalization over 20 trillion USD.

***Shanghai Stock Exchange (SSE)***

Shanghai Stock Exchange is considered largest Asian stock exchange, whose value sits at approximately 6 trillion USD.

***S&P 500***

Even though S&P 500 isn't stock exchange I decided to include data from its performance in gold price prediction. S&P 500 is the exchange fund where 500 best performing US companies have their stock included. This might well reflect on the performance and overall stability of American economy.

**Inflation**

According to *Investopedia* inflation goes hand in hand with currency valuation. Even though there are several factors influencing value of currencies, in general higher inflation results in lower value of the currencies. In the periods of high inflation or economic instability, gold tends to be investors' safe haven.

***US inflation***

I decide to include inflation in United States of America, because it is one of the biggest economies in the world, which might be saying a lot about overall economic situation worldwide and might also be considered as a reliable indicator of future gold price.

**Bond**

***Yield on 10 Year US Bond***

Bond yield states what return investors get from their initial investment, if they hold bond until its maturity. By investing in bond, investor invests in country's debt, and government guarantees the yield on this investment. Since prices of bond depends on a country's economic stability and growth, I assumed that it might also be related to gold price.

**Cryptocurrencies**

***Bitcoin***

Cryptocurrencies, in the times of economic instability also serve investors as safer asset. Their value generally increases with decreasing value of biggest world currencies like Euro or US Dollar. It can be assumed that Bitcoin has positive relationship with gold, since demand after it increases, in the times of higher inflation.

**Final Reflections on data used to predict gold price**

Gold price is influenced by various factors such as price of other commodities, financial markets, currency exchange rates, inflation, government bonds or alternative assets such as Bitcoin. In predictive analysis it is crucial to gather as much relevant data as possible. In this analysis I included mentioned data (*predictors).* ***Omitted Variable Bias*** i.e. not including one or more relevant predictors in the analysis might be present in this case and can affect overall accuracy and performance of the model. However, It is impossible to gather all possible factors and project them in data so I considered mentioned predictors as crucial in determining gold price. Omitted Variable Bias should thus have minimal impact on accuracy of performance of the models. It should be noted that vast majority of the predictors are classified as daily data – there is value assigned to financial instrument for every day. For some predictors values are listed for every day of the year (*currency rates*) and for some only on market days (*Monday – Friday*), meaning that multiple missing values simply must be present. When explaining each macroeconomic factor relevance, a general relationship with gold was assumed. However, it needs to be pointed out, that even though relationship between these factors and gold's price is in many cases unpredictable and does not always follow mentioned patterns. After gathering all the data the next step is to check for any irregularities or mistakes within each dataset, next section will reflect on this process.

# 2. General Data Processing

To ensure that algorithm can easily read the data it must be sufficient, well-processed and relevant. The data were obtained from various sources. such as *Fred, Macro Trends or Investing,* hence, they have different time frequencies or overall format. To ensure that algorithm can process and read the data, they must have joint format and consistent time frequencies, otherwise it could lead to misinterpretation of the data or false conclusions. The primary goal of data cleaning process is to have one structured dataset, that makes sure data accessibility for predictive analysis. There are various challenges when it comes to data processing such as missing values, different format or different time frequencies.

**Standardization of Date formats**

Because data were obtained from various sources, each source used different date formats or i.e. observation name. For instance, I encountered at least four different date formats.

- 2023/04/15
- 2023-04-15
- 15-04-2023
- 2023 Apr 15.

Clearly first three dates are represented numerically, but the last contains text-based month abbreviation. For algorithm to clearly read the name of the observations (dates) they need to be standardized into numeric format. After all dates were transformed into a numeric form, they were further standardized into Year-Month-Day (*YYYY-MM-DD*) format. After transforming the dates to joint format across all datasets, I ensured, that algorithm will read and process data efficiently. Another point to mention when it comes to dates format is that some datasets had descending and some ascending order of the dates. This was solved by setting all dates order to ascending.

**Selection of relevant predictor variables**

Some datasets contained various variables such as day high, day low, % change or open value. In the predictive analysis only include the “Price” column i.e. variable, which states the price of instrument at day close will be included. It is not necessary to include variables such as day low, day high or volume traded at the specific day. After all the data are processed and cleaned,

they are ready to be merged into one dataset. To maintain clarity column names were standardized by renaming columns in each dataset from "Price" to "Name of instrument" (e.g. *WHEAT_DAILY or SILVER_DAILY*).

**Merging**

After completing data cleaning and pre-processing, datasets are ready to be merged into one single dataset. When using algorithm to conduct predictive modelling there must be one single structured dataset, where all variables are included. To achieve this, datasets was merged on a basis of common date index, which is the name for each observation where each column corresponds to specific variable (*SILVER_DAILY, WHEAT_DAILY*). Most of the predictors are classified as daily data – there is value assigned to financial instrument for every day.

**Selecting observations with complete data**

After final merged dataset is ready, I encountered many empty observation (*figure 1.).* To conduct predictive modeling, these missing values must be either filled or omitted. The goal is to have as much full consecutive observations as possible, which would enhance model's accuracy and interpretability. It should be noted that in this section I deal with NA values that occur due to lack of data entry or simply could not be obtained from available sources. While NA values could still be present in the dataset, they occur rather occasionally. To avoid going through each predictor manually, I used a code, which directed me to the first and last complete observation. Observations occurring before first complete observation (*observation with all values in predictors*) and after most recent complete observation were omitted. The number of observations decreased drastically from 17 366 down to 2601, however this significantly improves model performance and reliability. Figure 1. And figure 2 in Appendix show dataset before and after omitting most of NA values.

# 3. Preprocessing data for modeling

After I generally processed data, which was described in the previous section, the next step in predictive analysis is to prepare the data for the actual predictive algorithms. It is essential that steps of data processing are in correct order, otherwise it could result in data leakage and unreliable prediction.

1. **Applying Lags**

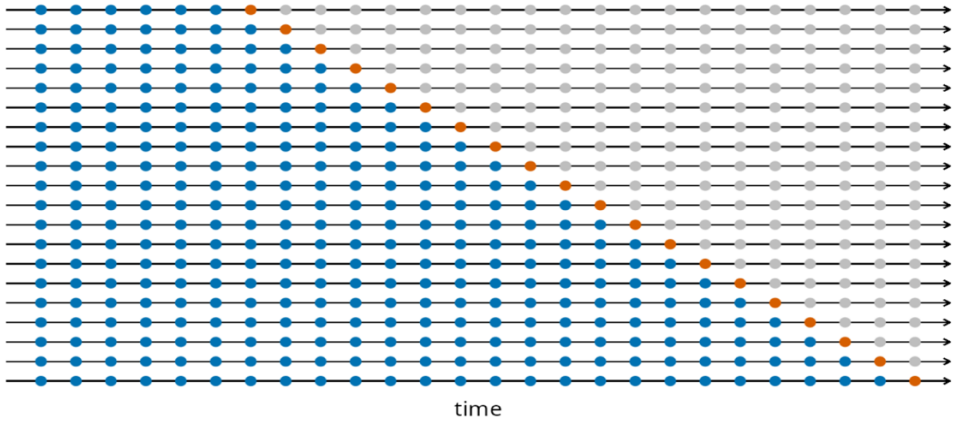
When conducting predictive analysis with time-series type of data, including lags as predictors might help increase model's accuracy and interpretability. In simple words lags are basically past value of predictor that might have some impact on present value of dependent variable Y i.e. yesterday's price of oil might affect gold price today. Adding lags improve performance and interpretability of the model but must be done thoughtfully to avoid unnecessary complexity and noise, which can arise when irrelevant lags are included. In this analysis I decided to include 1-day and 2-day lags to enhance model's performance. For each predictor I created 1 and 2-day lags, which increased the number of predictors to 51. Now the goal is to eliminate the irrelevant lags

In the next section I will explain why and how Rolling Cross Validation method helped me determine which lags to include in the dataset.

## 2. Rolling Cross Validation – Feature Selection Using Elastic Net

For the time-series data, the traditional cross-validation model cannot be used, in other words we cannot split training and validating sets randomly for time-series forecasting, because when we deal with time-series data, observations are bounded by certain date within the time-series. Traditional Cross-Validation would break time order between observations, i.e. we cannot predict the past data using present data. Hence, we must avoid breaking time-order between the observations and preserve time dependency within the training and testing dataset. To handle this issue, I will use Rolling Cross Validation. In simple words Rolling CV basically "rolls" training and testing datasets forward in time. Based on the research paper *Volatility forecasting using deep neural network with time-series feature eembedding* by *Chen,Wei-Jie & Yao, Jinj-Jinj & Shao, Yaunhai* from 2022 the algorithm begins with small subset of time-series (*t)* observations and predicts the follow-up data with length of $t + n$. Than it is evaluated based on mentioned criteria. Every step model undergoes is referred to as iteration. This allows model to be evaluated multiple times on different subsets of the data, this process is repeated until the model hits the end of time-series.

To provide better understanding of Rolling Cross Validation Concept I used following illustration where blue dots represent training observation $t$ , orange dots represent one added data point $n$. Each row represents one iteration in the process.




*(Figure 3. source: Hyndman R., Athanasopoulos, Forecasting: Principle and Practice 3rd Edition, 2021)*

**Elastic Net for Feature Selection**

To understand how Elastic Net works, Linear, Ridge and Lasso regression terms need to be briefly introduced first.

**Linear Regression**

Linear regression is a model, that estimates relationship between *Y* (dependent variable e.g. Gold Price) and independent variables *X* (Predictors), it assumes a linear relationship between *Y* and *X*. The Formula is as follows.

$$\widehat{Y}_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_n X_{in} + \widehat{\varepsilon}_i$$

- $\widehat{Y}_i$ represents estimate for dependent variable.
- $\beta_0$ represents estimate for intercept, i.e. predicted values of *Y* when all predictors *X* are equal to zero.
- $\beta_1, \beta_2 \ldots . \beta_n$ are coefficients. They determine value of change in *Y* when one-unit change in predictor **X** occurs, holding other predictors constant.
- $X_{i1}, X_{i2} \ldots . X_{in}$ are predictors or features of observation.
- $\widehat{\varepsilon}_i$ stands for noise or error.

This thesis deals with time series data, so equation for Linear Regression would need to be changed to:

$$\widehat{Y_t} = \beta_0 + \beta_1 x_{1t} + \beta_2 x_{2t} + \beta_3 x_{3t} + \cdots + \beta_n x_{nt} + \widehat{u_t} \quad for\ t = 1, \ldots, T$$

This is similar equation as before, but since we have time series data, we must denote small *t* index, which stands for time.

For clarification variables from actual dataset were plugged in the formula.

$GOLD\_SPOT\_PRICE_t = \beta_0 + \beta_1\ GOLD\_SPOT\_PRICE_{\ t-1} + \beta_2 INTEREST\_RATE\_US_{t-1} + \varepsilon_t$

The following formula known as ***Ordinary Least Squares*** (OLS) defines how coefficient *Beta* is calculated. The formula minimizes sum of squared differences between real and predicted values.

$$\widehat{\boldsymbol{\beta}} = \frac{\sum(x_t - \overline{x})(y_t - \overline{y})}{\sum(x_t - \overline{x})^2}$$

- $\widehat{\boldsymbol{\beta}}$ is estimated coefficient, that determines relationship between *Y* and *X.*
- $\overline{x}$ **and** $\overline{x}$ are averages of predictors and variables respectively at the time *t.*
- $x_t$ **and** $y_t$ are predictor and dependent variable respectively at the time *t.*
- $\sum$ sums up the entire time period (t = 1,2,3…. T).

In case Linear Regression was used in predictive modelling, calculating coefficients (*Betas)* would not be point of interest because the goal is to predict, not to assess magnitude of coefficients. However, since this section is dedicated to selection of variables that impact *Y* variable, the values of coefficients are crucial. Since, the goal is to select impactful variables for further predictive analysis, Linear Regression can provide coefficient value, but this alone is not enough to select predictors for further analysis.

**Ridge Regression**

$$\widehat{\boldsymbol{\beta}^{\mathbf{ridge}}} = \boldsymbol{\beta}\mathbf{argmin}\left\{\sum_{t=n+1}^{T}\left(y_t - \beta_0 - \sum_{j=1}^{p}\beta_j\, y_{t-j}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2\right\}$$

As stated in the book *Elements of Statistical Learning* Ridge Regression is extension of OLS, it shrinks predictors' coefficients by adding penalty, based on their value. In previous equation,

all terms were already mentioned, but new term L2 regularization penalty $\boldsymbol{\lambda}$ is introduced. "Lambda" is penalty, that is being imposed on coefficients and shrink them towards zero, but never to exact zero. $\lambda > 0$ is a parameter, that controls by how much coefficient are shrunk. In other words, the greater value of lambda, coefficients are likely to be close towards zero.

This method still does not help us with selecting important predictors, but it is crucial step in Elastic Net model, since it shrinks the value of coefficients towards zero.

**LASSO Regression**

$$\widehat{\boldsymbol{\beta}^{\boldsymbol{L}}} = \boldsymbol{\beta}\mathbf{argmin}\left\{\sum_{t=p+1}^{T}\left(y_t - \beta_0 - \sum_{j=1}^{p}\beta_j y_{t-j}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j|\right\}$$

Abbreviation LASSO stands for Least Absolute Shrinkage and Selection Operator, as the name implies, this method can perform variable selection introducing L1 penalty $\boldsymbol{\lambda}$**.** L1 penalty term in Ridge Regression shrinks coefficients towards zero, but in this case L1 penalty can set coefficients estimates to exact zero, because penalty is imposed on absolute value of coefficients. Hence, effectively removing unimportant variables. In this case L1 penalty is subject to $\boldsymbol{\lambda}$ >= 0. Choosing right penalty will not be discussed as algorithm does it automatically by trying different penalty parameters and choosing the best performing one.

To better understand the difference between LASSO and Ridge Regression following figure 4 in Appendix clarifies differences between the two methods graphically.

**Elastic Net**

Elastic Net could be described as a method that incorporates both L1 and L2 i.e. Ridge and LASSO penalties.

$$\widehat{\beta_e} = \frac{1}{2n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2 + \lambda\left(\alpha\sum_{j=1}^{p}|\beta_j| + (1-\alpha)\sum_{j=1}^{p}\beta_j^2\right)$$

- The first part of the term is again prediction error term
- $\boldsymbol{\lambda}$ Is the penalty parameter
- $\boldsymbol{\alpha}$ is a parameter, that defines balance between Ridge and LASSO regression.

- If $a$ = 1; LASSO (stronger variable selection)
- If $a$ = 0; Ridge (just shrinkage of variables' coefficients)
- If 0 < $a$ < 1; *(balance of both i.e. Elastic Net)*

- $\boldsymbol{\alpha} \sum_{j=1}^{p} |\boldsymbol{\beta}_j|$ is a term for LASSO L1 penalty.
- $(\mathbf{1} - \boldsymbol{\alpha}) \sum_{j=1}^{p} \boldsymbol{\beta}_j^2$ is a term for Ridge L2 penalty.

Elastic Net Regression is a powerful tool when it comes to multicollinearity and feature selection. This thesis deals with time-series data, which in most cases have highly correlated predictors. L1 penalty can set predictors' coefficient to exact 0, hence, remove it to improve model interpretability and reduce overfitting. To use Elastic Net fully balanced between L1 and L2 penalty, alpha parameter value should be set exactly to 0,5. Values ranging from 0,1 – 0,9 were used when running the model, but value of 0,5 provided best results.

**Criteria used to evaluate different parameter subsets**

***MAE (Mean Absolute Error)***

$$= \frac{\mathbf{1}}{\boldsymbol{n}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{n\rho}} |\boldsymbol{e_i}|$$

- $\boldsymbol{n}$ represents total number of observations.
- $|\boldsymbol{e_i}|$represents error, which in absolute value, so both over and under-predictions contribute equally. It is the difference between real and predicted value.
- $\frac{\mathbf{1}}{\boldsymbol{n}}$ averages the absolute errors
- $\sum_{\mathbf{i=1}}^{\mathbf{n\rho}}$ sums up all the errors across the dataset.

MAE is widely used when determining by how much predictions differ from real values, not taking into consideration whether the difference is negative or positive.

***RMSE (Root Mean Squared Error)***

$$= \sqrt{\frac{\mathbf{1}}{\boldsymbol{n}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{np}} \boldsymbol{e_i^2}}$$

- $\boldsymbol{n}$ represents total number of observations.
- $\boldsymbol{e_i^2}$ represents error, which in absolute value, so both over and under-predictions contribute equally. It is the difference between real and predicted value but **is squared.**
- $\frac{1}{n}$ averages the sum of squared errors.
- $\sum$ sums up all the errors across the dataset.
- $\sqrt{}$ squares the sum of squared errors, making it easier to interpret, by changing errors to original units.

RMSE also measures difference between real and predicted values, however the larger deviations count more, so RMSE would typically be greater than MAE, but not by much.

**Parameter tuning:**

***Choosing t (initial subset of observations used for training the model i.e. rolling window) and n (number of observations used to test the model)***

There is no correct answer to what values of parameters should be used when running a model. It depends on number of observations, predictors or overall data. The algorithm was ran on different parameter values, and reasonable parameters were chosen. The results for every possible combination of parameters can be seen in the screenshot of R interface in Appendix, Rolling Cross Validation Section

Considering prediction errors and the number of selected predictors I decided to exclude all parameters where horizon of prediction was 1 day. Using parameters with horizon of 1 yield in same value in error metrics. When taking number of predictors into consideration, the goal is to have small number of important predictors. Looking at window size of 300, the RMSE and MAE are relatively consistent without huge variations across all horizons *(1-20)*. I therefore decided to choose window size 300 and horizon 10 as tuning parameters.

Rolling Cross Validation Using Elastic Net for feature selection was run on parameters *(300;10)*. Function summary() of the model provides better understanding of each predictors. To further support chosen values for parameters (300;10), which resulted in MAE of 10,29, MAE was compared to average variations in daily price, which resulted in 10,07.

Figure in Rolling Cross Validation part in Appendix provides overview of coefficients that Rolling Cross Validation using Elastic Net for Feature Selection computed. Out of 54

predictors, 28 were determined as impactful, least impactful predictor's coefficients were set to exact 0, and thus removed.

| | Window_Size | Horizon | Mean_RMSE | Mean_MAE | Num_Selected_Features |
|---|---|---|---|---|---|
| 1 | 100 | 1 | 10.079773 | 10.079773 | 24 |
| 2 | 100 | 3 | 12.860755 | 11.386288 | 31 |
| 3 | 100 | 5 | 14.482042 | 12.487067 | 31 |
| 4 | 100 | 10 | 17.520840 | 14.946902 | 21 |
| 5 | 100 | 20 | 22.647081 | 19.293930 | 26 |
| 6 | 200 | 1 | 9.555695 | 9.555695 | 34 |
| 7 | 200 | 3 | 11.523380 | 10.071357 | 41 |
| 8 | 200 | 5 | 12.380677 | 10.481535 | 47 |
| 9 | 200 | 10 | 14.056042 | 11.733595 | 45 |
| 10 | 200 | 20 | 16.694711 | 13.877912 | 33 |
| 11 | 300 | 1 | 9.265992 | 9.265992 | 30 |
| 12 | 300 | 3 | 11.039832 | 9.552461 | 33 |
| 13 | 300 | 5 | 11.680190 | 9.762308 | 35 |
| 14 | 300 | 10 | 12.571780 | 10.299951 | 28 |
| 15 | 300 | 20 | 14.015647 | 11.426743 | 32 |
| 16 | 400 | 1 | 9.519077 | 9.519077 | 41 |
| 17 | 400 | 3 | 11.098118 | 9.607892 | 44 |
| 18 | 400 | 5 | 11.621454 | 9.711112 | 31 |
| 19 | 400 | 10 | 12.415779 | 10.125185 | 38 |
| 20 | 400 | 20 | 13.600478 | 10.982912 | 34 |
| 21 | 500 | 1 | 9.841144 | 9.841144 | 40 |
| 22 | 500 | 3 | 11.437928 | 9.861158 | 35 |
| 23 | 500 | 5 | 11.971048 | 9.963031 | 29 |
| 24 | 500 | 10 | 12.726222 | 10.307551 | 37 |
| 25 | 500 | 20 | 13.665784 | 10.922311 | 34 |

### 3. Handling Missing Values

When there is no value stored for an observation, we encounter missing value or NA (not available) values. Missing values might be present due to various reasons such as data entry errors or differences between data collection frequency. With too many missing values algorithms would not be able to identify pattern. After selecting only impactful variables the dataset now includes 2601 observations of 28 predictors. Number of missing values in this filtered dataset is 3543, which indicates, that dataset has 4,7% of all datapoints missing. To fill in missing values imputation will be performed. Only 4,7% of missing values needs to be imputed. Imputed value will be calculated by averaging previous and the next available value in a predictor to ensure consistent dataset for further predictions. After imputation, dataset is containing 24 missing values, which account for observations of lagged predictors in the

beginning. Values for lagged predictors in the beginning of the dataset simply cannot be imputed.

**Finalized Dataset for Predictive Modeling**

After throughout preprocessing including feature selection using Elastic Net and consistent imputation of missing values, the dataset is well set for predictive modeling. The use of Rolling Cross Validation method kept the time dependencies eliminating issues related to potential bias or overfitting. With further use of Elastic Net method, 15 predictors' coefficients were set to 0, e.g. determined as not impactful on the gold price. After eliminating 15 predictors, the dataset now contains 28 predictors, which support data interpretability and further predictive performance. Completion of mentioned preprocessing steps is crucial when preparing the data for actual predictive modeling.

# 4. Predictive models

## 4.1. ARIMA

ARIMA or Autoregressive Integrated Moving Average will be the first method, that this thesis will discuss. Unlike the many machine learning methods, which typically require many variables or predictors to recognize patterns within the data and generate predictions, ARIMA is statistical method which relies solely on the past values of *Y* variable (gold price) and its past prediction errors. To get to the core idea how does ARIMA work following three components are necessary to be discussed.

1. ***AR (Autoregressive)***

As stated in the *Forecasting: Principles and Practice,* by *Hyndman Anthanasopoulos* Autoregressive part of the model can be described as model that generates prediction using linear regression. The only difference here is, that AUTOregression uses past values as predictors. Following formula represents Autoregressive part of the model.

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t$$

- $y_t$ is prediction of target variable at time *t.*
- $c$ represents intercept.

- $\phi_1$ is coefficient i.e. the value that determines the "influence" of $y_{t-1}$ (past value) on value, that is being predicted.
- $\phi_p y_{t-p}$ represents lagged (past) values of *Y* variable multiplied by coefficient
- $\varepsilon_t$ stands for error term at time *t.*
-

### *1.1. Computation of coefficients*

To determine how much "weight" each past value carries coefficients must be computed. Coefficients are computed using OLS (Ordinary Least Squares). The goal of OLS is to minimize errors between actual and predicted values. Following formula represents equation of OLS, i.e. by solving this equation coefficients are computed.

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

Where $\hat{\beta} = \begin{bmatrix} c \\ \phi_1 \end{bmatrix}$

- $\hat{\beta}$ is vector of estimated intercept and coefficients.
- $X^{\top X}$ represents matrix of intercept and predictors (past values)
- $X^{\top y}$ is matrix of actual *Y* values.

Large number of observations would be required to showcase how value of coefficient is computed in matrix notation, thus, to clearly demonstrate how autoregression formula works I assumed following:

c = [20], $\phi_1$ = [0.6], $\phi_2$ = [0.3], $y_{t-1}$ = [1865], $y_{t-2}$ = [1850]

$$\boldsymbol{y_t = 20 + 0.6 \cdot 1865 + 0.3 \cdot 1850 + \varepsilon_t = 1694}$$

Value of gold price at time *t* = 1694 was computed using assumed values for coefficients, and values from previous two observations.

### *2. I (Integrated)*

Another part of ARIMA model is Integration or so-called Differencing. According to *Hyndman* and *Anthanasopoulos* differencing means transforming non-stationary into stationary data. Transforming non-stationary data into stationary helps reduce variance as well as stabilizing the moving average of data by disregarding trends in price changes of

target variable, allowing the model to better capture patterns. In simple terms integration in this context means, not looking at actual observations' values, but rather at the value of difference between observations. Plot in appendix, ARIMA section clearly shows the difference between stationary and non-stationary data.

$$\boldsymbol{y'_t = y_t - y_{t-1}}$$

- $y_t$ is the current value
- $y_{t-1}$ represents previous value

$$\boldsymbol{y'_t = 1865 - 1850 = 15}$$

### *3. MA (Moving Average)*

As explained by *Hyndman* and *Anthanasopoulos* moving average helps generate predictions by taking past prediction errors into consideration. Moving average adjusts the prediction based on past errors, simply put if target variable prediction was overpredicted today, model should adjust today's prediction accordingly. Following formula will clarify how Moving Average is computed.

$$\boldsymbol{y_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}}$$

- μ is the mean of all observations.
- $\varepsilon_t$ term represents error from today.
- $\theta_1$ represents the "weight" of previous error, which is estimated by minimizing function of sum of squared errors.
- $\varepsilon_{t-1}$ previous error

To compute today's price using moving average formula I assumed following values;

μ = [1800]

$\varepsilon_t$ = [5]

$\theta_1$ = [0.6]

$\varepsilon_{t-1}$ = [-10]

$$\boldsymbol{y_t = 1800 + 5 + 0.6\,x\,(-10) = 1799}$$

Computed equation using assumed values would generate prediction of gold price at 1799.

**Autoregressive Integrated Moving Average**

Arima combines mentioned three components into one final model, that can be represented using following equation, on already differenced data.

$$\boldsymbol{y'_t = c + \phi_1 y'_{t-1} + \theta_1 \varepsilon_{t-1} + \varepsilon_t}$$

Given c = 20, $\phi_1 = 0.6$, $\theta_1 = 0.5$, $y'_{t-1}$ = 15, $\varepsilon_{t-1}$ = -10 and $\varepsilon_t = 5$, I computed formula, which resulted in:

$$\boldsymbol{y'_t = 20 + 0.6 \cdot 15 + 0.5 \cdot (-10) + 5 = 20 + 9 - 5 + 5 = \ 29}$$

Using last known actual value $y_{t-1}$ = 1865 we have:

$$\boldsymbol{y_{t-1} + y'_t = 1865 + 29 = 1894}$$

***Pros and Cons of the model***

As discussed in the book *Forecasting : Principles and Practice* ARIMA is highly effective when it comes to short-term predictions, since it is very dependent on most recent observations as discussed in sections dealing with autoregression and moving average. The average daily variation of gold price is relatively low ~ 10.13 $, which should enable model to be effective in predicting.

Another advantage of ARIMA model could be fact, that target variable prediction is relying only on its past data, meaning that there is no need to include other external predictors such as exchange rates or other financial instruments.

The whole model is considered based solely on the statistics using established statistical theory.

The fact, that ARIMA model is relying solely on past observations of target variable, might on the other hand be considered as disadvantage. This is mainly due to potential sudden spikes of gold price due to external factors such as political tensions or other macroeconomic factors. Model might struggle to adjust its predictions accurately in the periods of high volatility and instability, since it cannot be explained by external factors.

ARIMA is very sensitive to hyperparameter values. Testing multiple combinations of parameters might be time expensive.

1. ***Data preparation***

   ARIMA model only requires target variable to generate predictions (gold price), hence new dataset was created, using single column with daily gold prices. Time – dependencies will be kept by using rolling window - fixed number of observations for training and predicting one step ahead. After prediction is generated, algorithm moves rolling window by one observation until it reaches the end of the dataset.

2. ***Parameter Tuning***

   ARIMA model consists of three main components, where each of them is treated as parameter.

### *2.1. Autoregressive Parameter – p*

Autoregression component of the ARIMA is taking past values into consideration, thus *p* parameter determines how many past observations does model uses to generate prediction. High *p* value might lead to overfitting, fitting training data very accurately, but might not recognize real patterns when it comes to generating predictions for unseen data. The book *Forecasting : Principles and Practice* suggests that low values, typically 1 or 2 should be sufficient.

### *2.2. Differentiating parameter - d*

*D* parameter determines order of differencing the data to achieve non-stationarity. According to *Hyndman and Anthanasopoulos* most time-series use 1st order differencing to ensure non-stationarity. Higher order might introduce unnecessary complexity for the model; thus this value was fixed to 1.

### *2.3. Moving Average parameter – q*

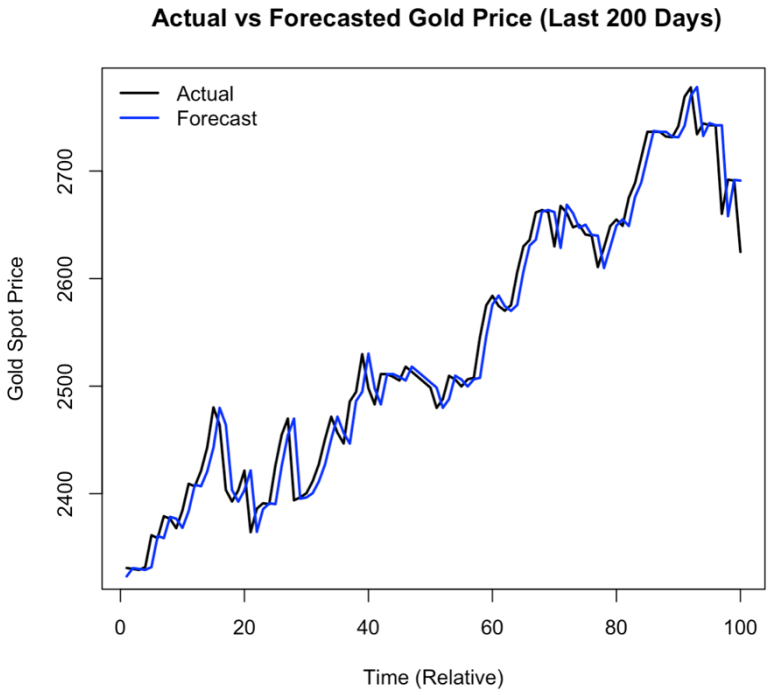
Moving average parameter - *q* controls how many past errors are used for generating predictions. Based on this parameter algorithm adjusts its predictions and captures short-term volatility or sudden spikes in the price. The book *Forecasting: Principles and Practice* discusses that values of 1 and 2 should be sufficient for real world financial data modeling.

## *3. Model Evaluation*

To determine optimal combination of parameters I conducted Cross-Validation method using range [1:3] for *p* parameter, [0:3] for *q* parameter and *d* parameter stayed fixed at 1. In total ARIMA was evaluated using 12 possible configurations of parameter values, as evaluation metric RMSE was used. The differences between configurations were negligible ranging [14.51, 14.62]. The lowest RMSE value yielded configuration (1, 1, 0) for *p, q* and *d* respectively.

The model was run using optimal parameter configuration and yielded in training error of 13.42 and 9.52 for RMSE and MAE respectively. Both values seem reasonable as average daily variance of gold price is around 10 USD. Higher RMSE is expected as larger errors are penalized more. When we take testing errors into consideration the values are 14.86 and 9.96 for RMSE and MAE respectively. Both values are slightly higher compared to testing error, which suggests that model can recognize patterns.

Following plot is comparing actual and forecasted gold price for last 200 observations. While it is not possible to determine whether model is over or underpredicting actual values, mean error was computed, what resulted in positive value (0.4). Since optimal ARIMA configuration was computed (1,1,0), the model simply assume that predicted value is just 1-day lag of actual value.




*(Figure 5; source : R Interface)*

Figure 6 in *ARIMA Appendix* illustrates distribution of errors. The mean is centered around zero, which suggest that model's predictions are not systematically biased. The graph supports observation from previous section that mean error is close to 0. It is also possible to observe that most errors are 25 units off. There are few outliers, but it is considered usual when modeling financial time series data due to sudden spikes of price.

Figure 7 in ARIMA Appendix displays two plots, that illustrate price and absolute errors over time. It is possible to observe upward trend in gold price. The model is also experiencing larger errors when gold is experiencing sudden spikes in the price. This observation is expected, because ARIMA takes only past values of variable and error into consideration, so it is unable to react to sudden spikes in price.

## 4.2. LASSO Regression

One of the models used for gold price prediction will be LASSO Regression. Least Absolute Shrinkage and Selection Operator is a regression model, that in addition to feature selection is also effective when it comes to predictive modeling. The formula is as follows.

$$\widehat{\boldsymbol{y_t}} = \boldsymbol{\beta_0} + \sum_{\boldsymbol{i=1}}^{\boldsymbol{p}} \boldsymbol{\beta_i X_{t-i}} + \boldsymbol{\lambda} \sum_{\boldsymbol{=1}}^{\boldsymbol{p}} |\boldsymbol{\beta_i}|$$

- $y_t$ is predicted value at certain time *t.* (Gold_Price)
- $\beta_0$ is the intercept, i.e. value of *Y,* when all predictors' coefficients are 0.
- $X_{t-i}$ predictors from previous time steps *t-1*
- $\beta_i$ represents coefficient for *i* predictor
- $\lambda \sum_{=1}^{p} |\beta_i|$ is L1 penalty term, which shrinks coefficients unimportant coefficients to zero

Following formula with assumed plugged in values demonstrates how algorithm predicts the gold price. I assumed that $\beta_0$ = 1230, $\beta_1$ = 0.5 (Coefficient for gold price lagged by one day), $\beta_2$ = -45 (coefficient for USD/EUR exchange rate) and $\beta_3$ = 1.8 (coefficient of oil price). The penalty is not present in final prediction formula, it is only included while model is being trained.

$$\widehat{\mathbf{Gold\_Price}}_{\mathbf{201}} = \mathbf{1230} + (\mathbf{0.5} \times \mathbf{1905}) + (\mathbf{-45} \times \mathbf{1.09}) + (\mathbf{1.8} \times \mathbf{84.2})$$
$$= \mathbf{2285.01}$$

The formula was computed provided assumed values for coefficients and predictors and resulted in Gold Price at *t* (201st) observation of 2285.01

**Pros and Cons of LASSO used for predictive modelling**

When dealing with dataset, which includes many predictors, it is considered high-dimensional. To eliminate irrelevant predictors, and thus simplify the model interpretability and performance, LASSO uses L1 penalty to shrink coefficients to exact zero, hence, removing them from the model.

Overfitting problem may also arise when performing not only LASSO, but any other model. Overfitting occurs when model fits the training data too accurate, likely capturing the noise, which can hurt predictive performance. In this case overfitting might cause a problem, since all datapoints are time-dependent, and model may capture noise rather than real patterns. For example, if too many lags were used, the model might fit training data perfectly, leading to overfitting. By setting non-essential predictors' coefficients to exact 0, it eliminates redundant predictors, simplifies model, and thus prevents overfitting. (AU Lecture presentation, Linear Regression)

Multicollinearity issue is also addressed by setting coefficients to zero by L1 penalty. Multicollinearity is occurring especially in time-series data. High correlation might be present within lagged variables or other macroeconomic indicators, which tend to oscillate together. In simple words, if there are multiple highly correlated predictors in the dataset, the LASSO model picks only the ones which possess predictive value for the model.

Overall, after issues mentioned are being handled LASSO may be considered as model that is characterized by its simplicity and good interpretability.

Excessive simplification could be considered both strength and weakness of the model. When more predictors are highly correlated, L1 penalty might regularize coefficients too aggressively by keeping only one predictor, even though others might also carry predictive value. If many coefficients are set to zero, LASSO might create a model, that is oversimplified leading to poor predictive performance and false data interpretability.

Sensitivity to penalty value may also be a challenge, since balance between penalty and MSE is crucial. The model tries to reduce MSE, while eliminating irrelevant predictors. If penalty is too strong, model might not consider predictors carrying valuable information. When penalty is too weak, model may behave like simple linear regression, being too complex or not handling multicollinearity, introducing risk of overfitting.

**Introducing Bias – Variance tradeoff**

Introducing bias might not be necessarily disadvantage of a model, but it might be beneficial. The following formula represents Bias – Variance tradeoff

$$\boldsymbol{E\left[\left(\widehat{Y}_i - Y_i\right)^2\right] = \mathbf{Var}\left[\widehat{Y}_i\right] + \mathbf{Bias}\left[\widehat{Y}_i\right]^2 + \mathbf{Var}[\varepsilon_i].}$$

- $\mathbf{Var}\left[\widehat{Y}_i\right]$ represents variance of our prediction. Usually, errors that comes from sensitivity to fluctuations of values in training subset. Measures error that would occur if we used different training subset.
- $\mathbf{Bias}\left[\widehat{Y}_i\right]^2$ bias is basically error, that comes out of simplifying complex problem with a simpler model.
- $\mathbf{Var}[\varepsilon_i]$ last term represents irreducible noise – errors that cannot be eliminated.

Simplifying a complex model might result in overfitting, poor predictive performance and misinterpretation of the data. In this thesis dataset used contains many variables that are highly correlated, making Bias – Variance tradeoff crucial. The goal is to minimize mean squared error by minimizing variance and bias. However, this is not possible because decreased variance results in increased bias and vice-versa. To put it simply the model tries to find penalty, that would find perfect trade-off between bias and variance. Larger penalty increases bias but lowers variance and small penalty does the opposite. Finding optimally balanced penalty term is thus essential. The process of how optimal penalty parameter is computed, will be reflected upon later in this section. Illustration in *LASSO Regression Appendix* represents bias – variance tradeoff; where black line represents Bias, green line represents variance and finally purple line stands for mean squared error

### *1. Data preparation (data splitting)*

The challenge arising when it comes to predicting the dependent *Y* variable (Gold price) is mentioned overfitting. Predictive algorithm attempts to identify pattern within the data, and based which it will generate predictions. However, if the model is trained and evaluated on the same subset of data it might lead to overfitting. In such case, model generates predictions based on the patterns identified on the data it has already encountered while training, leading to minimal error rate. Minimal error in this case does not mean that the model performs well, but

rather that it simply memorized training data patterns. The solution to overcome this challenge is to divide data into three subsets.

- **Training set,** typically the largest partition. This subset of data is used for training, where algorithm tries to recognize patterns within the data.
- **Validation set,** used to assess how well was model trained on the training set on unseen data before final testing
- **Testing set,** subset used for final evaluation of model's predictive performance on unseen data.

The data split ensures, that algorithm will try to identify patterns and learn exclusively on the training data subset, tune parameters using validation set and finally generate predictions using optimal parameter value on testing subset.

There is no correct answer in what exact proportions should data be divided into, but training subset is usually largest, while validation and testing are usually similar size. Using the filtered dataset which includes $Y$ variable (gold price) and 28 $X$ variables (predictors) with 2601 observations in total data was divided as follows.

- Training set (60%); 1560 observations
- Validation set (20%); 520 observations
- Testing set (20%); 521 observations

### *2. Keeping time dependencies while splitting the data*

The way of how data are partitioned into three subsets depends on the nature of the data. When dealing with time-series i.e. time dependent data algorithm must ensure, that time dependencies are maintained. In other words, algorithm must be trained on past data and generate predictions on future data. When partitioning the data that are not time dependent, shuffling data observations is random, however in this case observations must be shuffled into three subsets based on chronological order. Ensuring the time dependent shuffling is essential to avoid data leakage as well as ensuring that future data will not be used to predict past data. Following formulas reflects mathematically on how data was partitioned while maintaining chronological order of observation dates.

- Dataset in the beginning

$$\mathcal{D} = \{(X_1, y_1), (X_2, y_2), \dots, (X_T, y_T)\}$$

- Training subset

$$\mathcal{D}train = \{(X_1, y_1), \dots, (X_{t_1}, y_{t_1})\}$$

- Validation subset

$$\mathcal{D}valid = \{(X_{t_1+1}, y_{t_1+1}), \dots, (X_{t_2}, y_{t_2})\}$$

- Testing subset

$$\mathcal{D}test = \{(X_{t_2+1}, y_{t_2+1}), \dots, (X_T, y_T)\}$$

- Defining time structure where;
  - *t1* represents end of training set at 60% of total observations

$$t_1 = \lfloor 0.6T \rfloor$$

  - *t2* represents end of validation set at 80% of total observations

$$t\_2 = \lfloor 0.8T \rfloor$$

  - *T* stands for total number of observations in the dataset

$$T = 2601$$

By defining ranges as {*t* = 1 to *t1}* ; {*t1* + 1 to *t2*} and {*t2* + 1 to *T,*}, it is ensured that observations in each subsets are not overlapping and mutually exclusive.

### *3. Parameter Tuning*

After data were partitioned maintaining time dependencies model is moving onto actual training data, where it tries to identify patterns in predictors to predict *Y* variable. The process of training data also includes choosing optimal penalty value.

***Cross-Validation*** technique is used to determine optimal penalty strength. Based on *Linear Regression, AU Lecture Presentation* Cross-Validation technique splits data into *k* partitions. It

estimates model based on k -*1* folds and evaluates it on the remaining partition. It repeats the process using different combinations of partitions. After process was repeated using all partitions, it calculates average performance over *k* partitions and chooses lambda that yields lowest error. By default, R programming language uses 100 different values in Cross-Validation for L1 penalty parameter. In this case 5 folds were set for Cross-Validation, which determined optimal lambda value.

After Cross-Validation was used to determine optimal *lambda* (penalty) value, the final LASSO model was once again trained using training subset, now with fixed value for optimal penalty. The model identified patterns and relationships between predictor values and dependent variable e.g. gold price. Once model finalizes training part, it moves onto generating predictions on unseen data using validation and testing subsets. Prediction accuracy was measured using Root Mean Error Squared (RMSE), Mean Error (ME) and MAE (Mean Absolute Error).

RMSE and MAE were already introduced in the thesis, however in the model evaluation I decided to also include Mean Error, which determines if the model is on average over or underpredicting. Since negative and positive errors cancel each other out value of this metric is expected to be around zero, but due to introducing Bias-Variance tradeoff this might not be case, because bias is increased.

$$\boldsymbol{ME = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)}$$

- $\frac{1}{n}$ represents number of observations
- $\sum_{i=1}^{n}(\hat{y}_i - y_i)$ second part of equation is summing up errors, that came from subtracting actual value from predicted value

### *4. Model evaluation and performance assessment*

Following table reflects on results from model performance evaluation.

Before assessing the errors, the fact that average price variation between two observations of gold price is 10,03 USD taken into regard.

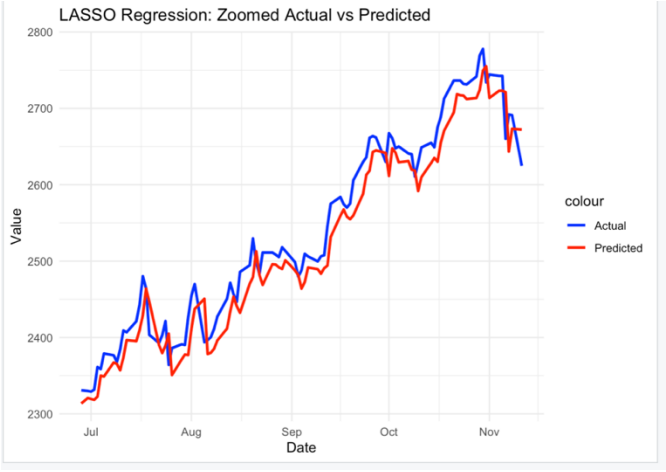
RMSE of training set ~ 12.23, is slightly above average daily variation indicating good fit on training data. Value of RMSE for validation set and test set is 17.2 and 21.6 respectively, suggesting that model loses generalization power to some extent. Increasing RMSE over sets indicates that model is not able to capture dynamics of values based on pattern learned from training set. One of possible reasons why this is occurring may be that validation or testing set contain the periods of time where price of gold was more volatile for instance Covid-19 Crisis or economic recession.

The same reasoning could be used for assessing MAE, which is across subsets slightly lower, mainly due to fact that RMSE penalizes large errors more and is more sensitive to outliers (spikes in price).

When it comes to values of Mean Error across subsets, we could say that there is almost no bias in training set, since negative and positive errors cancel each other out and value is close to zero. As mentioned after selecting optimal penalty value model is retrained, meaning that is being trained on data it has already seen (training subset), which explains error very close to zero. Looking at errors for validation and testing sets, it can be concluded that model is underpredicting on average, where bias is increased due to introduction of penalty term.

In Conclusion it can be assumed that variation of errors across all three subsets might be due to price spikes and volatility in times of economic recession or sudden changes in macroeconomic environment.

Following plot displays two curves – actual and predicted values. It can be observed that model is underpredicting the actual values. The model also seems to not react well to sudden spikes, underestimating actual price.




*(Figure 9; R Interface)*

Figure 10 in *LASSO Appendix* displays residuals over time. It is obvious, that errors fluctuate around zero, but tends to be negative at most predictions especially in second half of the plot. The magnitude of errors seems to increase towards the end of period, suggesting either more volatility in prices or decreasing model accuracy when it came across more recent data.

Histogram displayed in figure 11 in *LASSO regression Appendix* shows distribution of prediction errors. They are normally distributed around zero with left skew. As appears on the histogram, most errors fall within the range from -20 to 20, but presence of outliers can also be observed. The skewness on the left side supports earlier observation that model is underpredicting. The outliers as mentioned may represent price spikes or economic recession.

## 4.3. Random Forests

To fully gain of understanding of how Random Forest model generates prediction, it is necessary that following concepts are briefly explained before getting to actual Random Forests method.

**Decision Trees**

Decision trees is one of the simplest supervised machine learning algorithms that uses a "tree" model for decisions. Decision trees build regression problem in the form of tree structure. It operates by partitioning the data into smaller subsets containing similar values in the predictors. We can imagine these subsets as a "tree structure" consisting of ***nodes, branches*** and ***leaf nodes.***

The node corresponds to a "decision point", where data are split based on the specific criterion value of a predictor. For example, criterion for a node could be *silver daily price > 29$.* Each node makes a split that divides data in a best possible way based on a criterion.

Branches represent a path between the nodes that data are flowing through. As data flows through nodes by branches, it eventually reaches leaf node.

Based on *Time Series forecasting using Tree Based method* research paper, leaf node is final subset of partitioned data used to generate prediction of *Y* variable. This process of data flowing through nodes by branches, eventually leading to being assigned into dataset to generate prediction is conducted independently in each decision tree. The final prediction of a regression Random Forest is average of predictions resulting from all leaf nodes.

**Ensemble Methods (Bagging)**

In research paper *"Ensemble Methods in Machine Learning" Diettreich* from Oregon State University describes ensemble methods as types of techniques that combine multiple models into one, which builds upon one main idea – Law of Large Numbers. Individual models have weaknesses, but when they are combined, it can lead to high predictive performance. There are few types of ensemble methods such as ***Bagging, Boosting*** and ***Stacking***. Random Forests are method that uses bagging. Bagging constructs numerous decision trees on bootstrapped datasets with random feature selection and then averaging their outputs. Bootstrapped dataset is created by randomly sampling original training data with replacement. Replacement in this case means that model randomly picks a datapoint for a decision tree, it is put back into "data pool", so it can be selected again.

**Random Forests**

According to the *Hasti, Tibshirani* and *Friedman* in the book *Elements of Statistical Learning* Random Forests is classified as an ensemble method, that generates accurate prediction by aggregating outputs of numerous decision trees. It is based on bagging ensemble method, which in general means, that each decision tree is trained on bootstrap training subset, which is randomly drawn from original training data with replacement. Rather than evaluating all predictors, Random Forests selects random subset of predictors at each decision node. This ensures that overall variance is decreased along with low bias. For regression problem, the final prediction is the average of all outputs from the trees (leaf nodes). The formula is as follows.

$$\widehat{y_{t+1}} = \frac{1}{B}\sum_{b=1}^{B} T_b(x_t)$$

- $\widehat{y_{t+1}}$ represents final aggregated predicted value at time *t +1*, that was averaged from prediction of all trees.
- $B$ is total number of decision trees
- $T_b(x_t)$ represents prediction made by the b*th* tree based on the input vector **x*t*** (subset of predictors)
- $\sum_{b=1}^{B} T_b(x_t)$ is the sum over all *B* trees, i.e. all predictions are being gathered

- $\frac{1}{B}$ takes average of all tree predictions

Following illustration explains graphically how final predictions are computed. In the beginning algorithm takes one datapoint with different values in predictors. For example, algorithm takes one data point (observation) at time ***t*** e.g. 20.06.2018 and uses values in predictor variables (Gold_t = 1925.3; Oil_t = 83.1; Interest_t = 4.75; …etc.). This data point is passed to all decision trees at prediction time (RT 1; RT 2 and RT 3). Each of these three trees uses its own structure of ***nodes,*** which were formed during training based on random feature selection and bootstrapped data. The data point passes the nodes connected by branches until final prediction of an individual tree in a leaf node. All of these individual predictions are than averaged to final prediction.

Breakdown of steps algorithm undertakes to form final prediction.

1. Algorithm receives data point with values in predictors, Oil_t = 83.1; Gold_t = 1925.3
2. Then runs the data point through decision trees e.g. " Is the value (83.1) of data point in Oil_t predictor greater than 80 ?"
    - If yes than either go to next node (if there is) or make prediction.
    - If no go to node on the left e.g. ask "Is the value of Gold_t (1925.3) in predictor greater than 1926?" and makes the prediction.

**Note :** to avoid confusion of why Gold_t is assessed as a predictor in this decision tree is, that we use past value of gold to predict the future ones, so here it can be treated as a predictor.

3. This process is repeated with all training observations.
4. The tree predicts value by simply averaging actual ***yt + 1*** values of training examples in the same leaf. This average is computed and stored in leaf node.
5. Algorithm takes average of all predictions across the trees and forms final prediction.

$$\widehat{y_{t+1}} = \frac{1}{3}(1923.7 + 1924.5 + 1926.1) = 1924.8$$

Figure 12 in *Random Forests Appendix* displays simplified diagram of how nodes are connected through branches leading to final prediction

**Pros and Cons of the model**

By averaging outputs of many decision trees Random Forests improves ***accuracy*** significantly. Each tree makes individual prediction based on different subset training data and features. These individual predictions are than averaged and form final prediction.

As discussed in *The Elements of Statistical Learning* increasing number of trees *B* in the forest does not lead to **overfitting,** but rather to more stable and consistent predictions because of averaging predictions from each individual tree to final prediction. Averaging predictions across all trees reduces variance, which is often primary reason complex models are overfitting. Even when Random Forests deals with high-dimensional data is keeps strong predictive accuracy, because probability that impactful variable will be selected at decision node is high enough.

On the illustration number 13 in *Random Forests Appendix* it is possible to see that both test and train mean squared errors are decreasing with increasing number of trees. However, if the number of trees is sufficiently large enough, adding more trees to the model does not really increase predictive performance.

Another advantage also discussed in *The Elements of Statistical Learning is* **OOB** or **Out-Of-Bag observations** is another advantageous feature of Random Forests model. It represents internal validation to estimate predictive performance without need of validation set. For each observation, the OOB prediction is calculated using only data points, that were not included in a bootstrapped subset, allowing model to evaluate performance while training.

Random forests is also beneficial when it comes to ***handling non-linear*** interactions between data points. The model does not assume any functional relationship between outputs and inputs, so it can naturally model non-linear interactions.

When algorithm comes across high number of decision trees that must be trained, Random Forests can be ***computationally expensive,*** especially with large datasets. The algorithm becomes more complex with increasing number of trees and the depth of each tree, which essentially leads to longer training time, what I experienced when training the model. To support this statement, that empirical testing of all parameters (algorithm had to be run 36 times), took approximately 50 minutes. "*Model training and evaluation"* section will further reflect on empirical testing.

Based on the Article *Understanding Random Forests Algorithm* provided by *Data Science Dojo* if parameters of the model (number of trees) are properly tuned ***Bias-Variance Trade-off does***

not constitute a challenge, however model has to include sufficient number of decision trees in order to find optimal bias-variance tradeoff.

**Modeling the algorithm**

***1. Data preparation along with maintaining time dependencies.***

Unlike LASSO regression in case of Random Forest due to previously explained Out of Bag observations there is no need for validation set in this model. The challenge algorithm faces now is, that Random Forests model is shuffling training data completely at random. The dataset thesis deals with are time-series data. Thus, the model needs to be manually set to partition data into chronologically ordered training and testing subset, so it recognizes time dependencies an avoids predicting past value of gold price based on a future one and prevents data leakage. To maintain time dependencies rolling window method was used. Different size of rolling window was used and evaluated along with different number of trees; the best performing model was than chosen to perform predictive analysis. Stated otherwise, data will be trained on size of subset that rolling window parameter will determine e.g. (100,150,200 datapoints) and will predict *rolling window* at time $t + 1$, i.e. model will be trained on rolling window size and will predict one observation ahead. This process is repeated until the algorithm reaches the end of the dataset.

***2. Parameter Tuning***

All parameters were chosen empirically manually changing parameters in algorithm.

***2.1. Rolling Window size***

The data this thesis generates prediction from are time-dependent data, so Rolling window size determined size of training subset, while keeping time-dependencies maintained. For this parameter predictive analysis was performed using following values 100, 150 and 200). Model using this size of testing subset predicts exactly $t + 1$ e.g. immediate next value based on historical data.

***2.2. Number of trees***

To support section from pros and cons, increasing number of trees in the model decreases variance, until it reaches the point, where additional tree doesn't increase predictive performance. The research paper "*Hyperparameter and Tuning Strategies for Random Forests"* written by *P.Probst, M.Wright and A.Boulesteix* states, that most predictive performance is

gained using first 100 trees, but 500 to 1000 trees are most often used. Since 500 trees were already computationally expensive, hence 1000 trees were not used as parameter.

### *2.3. Number of random predictors*

The number of random predictors at each decision node is determined by value of *mtry* in the algorithm. Algorithm uses this parameter to determine how many predictors it considers at each decision node, for instance, at first decision node algorithm randomly selects 5 out of 28 predictors and chooses the predictor and its value for data split. This process is repeated until algorithm reaches leaf node in individual tree. The research paper "*Analysis of Random Forest Model"* by *Gerard Biau* states, that if the *mtry* parameter is too small, probability of choosing strong variables decreases. To avoid this *Gerard Biau* states following set of formulas.

***Probability of strong predictor being chose at single split***

$$p_n^\star = \frac{1}{S}\left[1 - \left(1 - \frac{S}{d}\right)^{M_n}\right]$$

- $d$ represent total number of predictors
- $S$ is number of impactful variables
- $M_n$ is *mtry* parameter

To clearly demonstrate how formula works I used following values; $M_n$ = 5, $d$ = 28 and I assumed the dataset used contains $S$ = 8 important predictors.

$$p_n^\star = \frac{1}{8}\left[1 - \left(1 - \frac{8}{28}\right)^{5}\right] = 0.1017 \sim 10.17\%.$$

The result obtained from the probability formula states, that at each split there is 10.17% chance of selecting important predictor. At first sight the probability might seem low, but it needs to be noted that this process is repeated numerous of times.

Based on first formula it states following theoretical condition

$$\left(1 - \frac{S}{d}\right)^{M_n} \cdot \log n \to 0 \quad \text{as } n \to \infty$$

This theoretical assumption ensures, that if dataset grows, probability of strong variables not used at split becomes negligible.

This assumption is satisfied only if following condition is met,

$$M_n \to \infty \quad \text{and} \quad \frac{M_n}{\log n} \to \infty \quad \text{as } n \to \infty$$

In simple words author of research paper *Gerard Biau*, argues, that as training size increases, *mtry* parameter should increase accordingly to ensure that strong variables are used in training.

Even tough theoretical grounds for this parameter were established, this thesis is dealing with time-series data, and rolling window e.g. training size is fixed number, this theorem will not apply and will serve just as theoretical foundation of this parameter. To determine optimal *mtry* parameter (*3,5,7,10*) values were empirically tested, and based on error metrics optimal value was determined.

### *2.4. Tree Depth*

The Tree Depth parameter determined how big the tree grows, i.e. bigger tree (more nodes) makes model able to capture more complex relationships. Even tough tree depth can be specified using very specific library "*party"* in R, which is able to change values of this parameter, nevertheless I chose not to specify it. Tree Depth. However, the research paper "*Analysis of Random Forest Model"* by *Gerard Biau* argues, that the depth of tree is determined by $\lceil \log_2(k_n) \rceil$ where $k_n > 2$, is the parameter, that algorithm sets automatically. If the value of the parameter is too small, it may lead to high bias, on the other hand, if the parameter's value is set too large, higher variance might occur.

### *3. Evaluation of the model*

The model was trained on using 36 different combinations of three parameter values (Rolling Window size, Number of trees and depth of the tree). As rolling window size values of *100, 150* and *200* was tested, *100, 300* and *500* was used as values for parameter determining number of trees in a model and finally *3, 5, 7, 10* were the values that determined the depth of the tree. Empirical results showed that combination of parameters that yielding to lowest RMSE and MAE was rolling window size of *100*, while number of trees and depth of the tree were set to *500* and *10* respectively, which supports theoretical assumption discussed earlier. The greatest value tested for depth of the tree (*10*) yielded to lowest error. However, when it comes to assessing parameter values for number of trees, it can be observed, as theory states, that adding more trees does not necessarily increase predictive power and the model stagnates, while it grows computing time. Even though there is small difference between errors using different values of training size (rolling window), it can be concluded that errors grow with larger training size. This means, that most recent data carry most predictive power, which might be due to trends or seasonality in prices.

It is obvious that both training errors are very low 5.18 and 3.59 respectively. However, this behavior is expected. This is evidence of how well model can recognize patterns in training data. The testing RMSE and MAE yielded in 18.94 and 13.37 respectively, compared to training error value is substantially higher, the simple explanation for this would be that this data is unseen for the model. Even tough testing error is substantially higher than training, model would be assessed as very accurate, comparing to average variation in daily price (10) the model's performance is considered relatively high.

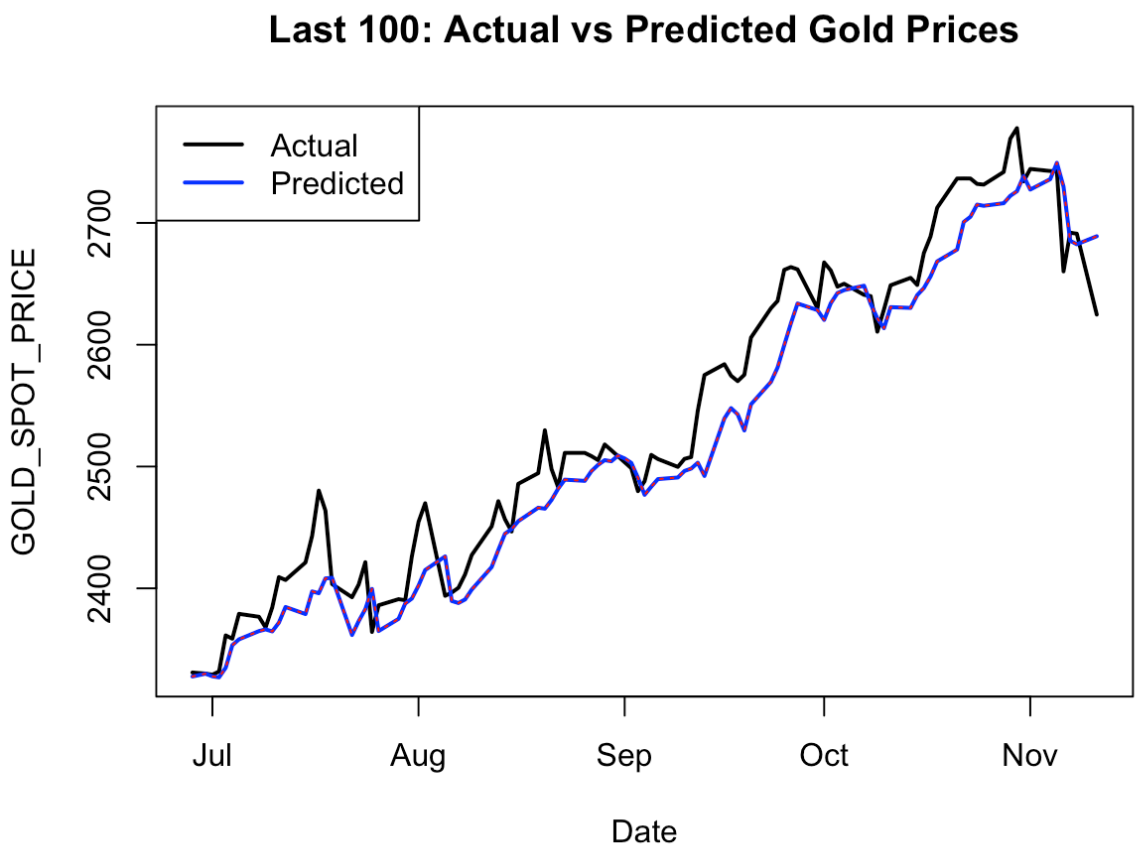
Figure 14 in *Random Forests Appendix* created in R provide overview of model performance.

Histogram of residuals clearly shows that residuals are evenly distributed around 0, meaning that the model is not systematically biased. On the histogram it can also be observed that outliers are present, but this is considered common behavior in financial time-series data, which may occur in volatile, recession or uncertainty economic periods.

To see difference between actual and predicted value over time, with the help of R actual and predicted values were illustrated. The line plot clearly shows upward trend of gold price movement. Prediction line clearly follows increasing trend; however, it is obvious that the model is underpredicting, especially in the times of significant spike or drop in a gold price. It

might be because the model is averaging noisy observations, which may make model less responsive to sharp spikes or drops.

| roll_window | ntree | mtry | test_rmse | test_mae |
|---|---|---|---|---|
| 100 | 100 | 3 | 21.4211 | 15.2668 |
| 100 | 100 | 5 | 20.2594 | 14.4025 |
| 100 | 100 | 7 | 19.4701 | 13.7796 |
| 100 | 100 | 10 | 18.9733 | 13.4141 |
| 100 | 300 | 3 | 21.3988 | 15.2691 |
| 100 | 300 | 5 | 20.1267 | 14.3122 |
| 100 | 300 | 7 | 19.4098 | 13.7406 |
| 100 | 300 | 10 | 18.9793 | 13.3772 |
| 100 | 500 | 3 | 21.3333 | 15.2395 |
| 100 | 500 | 5 | 20.0628 | 14.2322 |
| 100 | 500 | 7 | 19.4818 | 13.8058 |
| 100 | 500 | 10 | 18.8400 | 13.3272 |
| 150 | 100 | 3 | 22.8158 | 15.9892 |
| 150 | 100 | 5 | 21.1630 | 14.8187 |
| 150 | 100 | 7 | 20.3504 | 14.2205 |
| 150 | 100 | 10 | 19.4723 | 13.5617 |
| 150 | 300 | 3 | 22.7255 | 15.9602 |
| 150 | 300 | 5 | 20.9428 | 14.6598 |
| 150 | 300 | 7 | 20.2213 | 14.1006 |
| 150 | 300 | 10 | 19.4152 | 13.5117 |
| 150 | 500 | 3 | 22.5068 | 15.8592 |
| 150 | 500 | 5 | 20.9722 | 14.6815 |
| 150 | 500 | 7 | 20.1882 | 14.0761 |
| 150 | 500 | 10 | 19.4263 | 13.5292 |
| 200 | 100 | 3 | 22.8690 | 16.0083 |
| 200 | 100 | 5 | 21.2886 | 14.7196 |
| 200 | 100 | 7 | 20.3327 | 14.0633 |
| 200 | 100 | 10 | 19.2841 | 13.2959 |
| 200 | 300 | 3 | 22.8693 | 15.9764 |
| 200 | 300 | 5 | 21.0928 | 14.7037 |
| 200 | 300 | 7 | 20.0411 | 13.9092 |
| 200 | 300 | 10 | 19.1690 | 13.2653 |
| 200 | 500 | 3 | 22.9154 | 15.9769 |
| 200 | 500 | 5 | 21.0465 | 14.6683 |
| 200 | 500 | 7 | 20.0891 | 13.9403 |
| 200 | 500 | 10 | 19.0917 | 13.2166 |




*(Figure 15; source :R Interface)*

Figure 16 in *Random Forests Appendix* was obtained using *varImPlot()* function in R, which generates plot based on importance of variables within the trees. In other words the plot shows predictors that were most used in decision nodes in order to increase predictive performance e.g. lower error rate. This function relies on correlation between variables, so lagged predictors of gold were used most in splits, following with stock indexes, USD/INR exchange rate and silver. Other predictors seem to have negligible predictive performance according to this function. This provides clear reflection on which predictors were most important when conducting predictive analysis.

## 4.4. XGBoost

XGBoost abbreviation for Extreme Gradient Boosting is widely used prediction algorithm, which also uses decision trees. To fully understand this complex algorithm, I would like to first reflect on following concepts.

**Ensemble Method (Boosting)**

In the book "*Elements of Statistical Learning*" the definition of Boosting is very clear – process, which combines outputs of numerous "weak" learners to form strong overall model. Unlike Bagging, this method sequentially learns from errors of previous models, not from random sample of training data. To put it in different words, Bagging is like asking 100 people the same question and averaging their answer, while Boosting would ask single person same question, pointing out the mistake and asking again until significant improvement in an answer. Illustration in *XGBoost Appendix* provide simple diagram of how XGBoost works.

As *C.Wijaya* states in his article "*Comparing Model Ensembling"* Boosting process takes data, trains it, yields prediction and perform error correction based on the model. The goal is to minimize errors as well as determining data points that previous model was struggling to predict.

The Formula for Boosting used in XGBoost is as follows.

$$\widehat{\boldsymbol{y}_{\boldsymbol{i}}^{(\boldsymbol{t})}} = \widehat{\boldsymbol{y}_{\boldsymbol{i}}^{(\boldsymbol{t-1})}} + \boldsymbol{\alpha} \boldsymbol{f}_{\boldsymbol{t}}(\boldsymbol{x}_{\boldsymbol{i}})$$

- $\widehat{y_i^{(t)}}$ stands for prediction at iteration (step) *t.*
- $\widehat{y_i^{(t-1)}}$ is previous prediction.
- $f_t(x_i)$ new tree fitted to the gradient of loss function, where $x_i$ is vector representing predictors.
- $\alpha$ determines learning rate (*to be discussed further in the section)*

I plugged in some values to clarify how this formula is used.

$$\widehat{\boldsymbol{y}_{\boldsymbol{1}}^{(\boldsymbol{3})}} = \boldsymbol{1925 + 0.1 \cdot 30 = 1925 + 3 = 1928}$$

- 1925 stands for prediction from previous tree
- 0.1 is learning rate
- 30 is so called "correction" 3rd tree suggests based on the error from previous one.

- 1927 is the prediction that 3rd tree yielded.

**Gradient Descent**

Previous section explained how boosting process is improving sequentially built models. Gradient Descent is crucial to understand, because it determines by how much and in which direction each tree improves the prediction. Following equations show how "improvement" is computed. Research paper "*XGBoost : A Scalable Tree Boosting System* written by *Tianqi Chen* and *Carlos Guestrin* define these formulas very clearly.

### *1. Exact Objective function*

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} \left[ l\left( y_i, \widehat{y_i^{(t-1)}} + f_t(x_i) \right) \right] + \Omega(f_t)$$

- $\mathcal{L}^{(t)}$ represent value that. XGBoost wants to minimize (total loss).
- $l$ **( )** is a loss function, measures residual.
- $\widehat{y_i^{(t-1)}}$ is a prediction after *t-1* trees.
- $y_i$ **s**tands for true value of variable that we predict (gold price).
- $\Omega$ is regularization term, which adds penalty for complexity of a model.
- $f_t(x_i)$ is new prediction

### *1.1. Regularization term omega*

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

- $T$ is number of trees in the (*Ft)* model
- $\gamma$ penalty for adding new leaf (regularize tree size)
- $\lambda$ regularization term (Ridge Regression), shrinks large leaf to zero
- $w_j^2$ j-th leaf prediction

I plugged in real numbers to demonstrate clearly how the equation works. I assumed that values for *T = 4, w = [5, -3, 4, -2], γ = 1 and λ = 0.5.*

$$\boldsymbol{\Omega(f_t) = 1 \cdot 4 + \frac{1}{2} \cdot 0.5 \cdot (5^2 + (-3)^2 + 4^2 + (-2)^2)}$$

$$\boldsymbol{\Omega(f_t) = 4 + \frac{1}{2} \cdot 0.5 \cdot (25 + 9 + 16 + 4) = 4 + 0.25 \cdot 54 = 4 + 13.5 = 17.5}$$

17.5 is value of penalty introduced for complexity of the model, that will be added to prediction error, thus we get:

$$\boldsymbol{\mathcal{L}^{(t)} = \sum_{i=1}^{n} l\left(y_i, \widehat{y_i^{(t-1)}} + f_t(x_i)\right) + \Omega(f_t)}$$

I assumed values for $y_i$ *= 1950,* $\widehat{y_i^{(t-1)}}$ *= 1925 and* $f_t(x_i) = 20$. For regularization term we use $\Omega(f_t)$ *= 17.5* calculating previous equation

For the first part of the equation we have :

$$\boldsymbol{l = \frac{1}{2}(1950 - 1945)^2 = \frac{1}{2} \cdot (5)^2 = \frac{1}{2} \cdot 25 = 12.5}$$

Thus, we need to add regularization term to 12.5 from loss function.

$$\boldsymbol{\mathcal{L}^{(t)} = 12.5 + 17.5 = 30}$$

In this example Objective loss function yielded value of 30, which consists of 12.5 determining prediction error and 17.5 penalty added to the error. During one "run" or iteration XGBoost evaluates numerous different trees and computes different objective value for each. The tree yielding lowest value is added to the model to make prediction. This process is repeated many times, which increases prediction accuracy and controls model complexity.

## 2. Second-order Approximation

Objective function mentioned firstly in this paragraph is considered exact objective function, while second order is efficient approximation of objective function mentioned

in section 1.1. Even though it is approximation it still maintains training efficiency and accuracy. It is based on two essential concepts – Gradients and Curvature (Hessians). General formula for second-order approximation is as follows at step *t.*

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

Is simplified into :

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

### *2.1.Gradient (First Derivative)*

Gradients determine direction of a slope (down or up). In other words it measures how wrong the prediction is. It is simply first order derivative of loss function mentioned in section 1.1

$$g_i = \frac{\partial l(y_i, \widehat{y_i})}{\partial \widehat{y_i}}$$

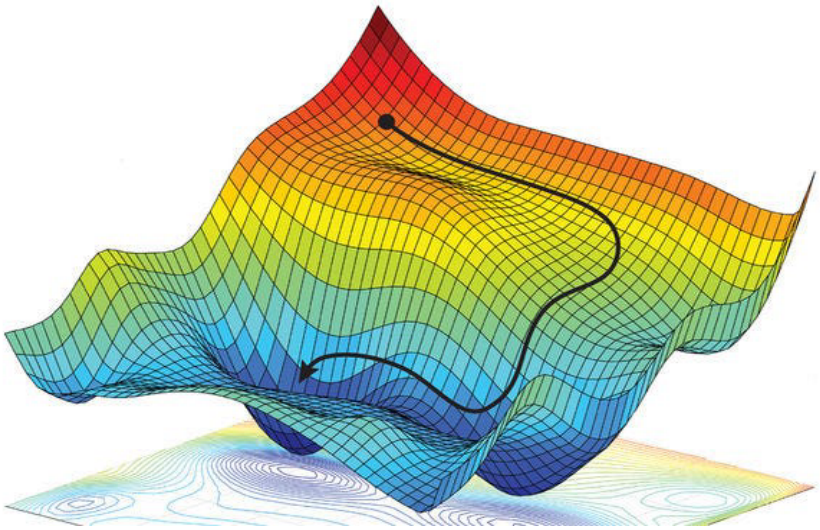
### *2.2.Hessian (Second Derivative)*

Hessians tells us how steep or flat the curve is. It reflects on how error is changing. This term states second order derivative of original loss function.

$$h_i = \frac{\partial^2 l(y_i, \widehat{y_i})}{\partial \widehat{y_i^2}}$$

To visualize how exactly second order approximation works I used hyperplane illustration. Hyperplane in this case represents space, where predictions and losses are located. The goal of the algorithm is to reach the lowest point i.e. lowest value of objective function. The path along the hyperplane reflects how the model updates Gradient and Hessian in each iteration. It moves either up or down based on Gradient, while Hessians determines how sharp or smoothly it moves. Essentially it is obvious, that algorithm improves with each observation eventually getting to the lowest point (blue area). I will not demonstrate these equations plugging in values,

because the theoretical and numerical concept of base objective loss function was already demonstrated in section 1.1.



*(Figure 18; source : Medium)*

After XGBoost computes Gradients and Hessians for each training subset, the next step is to determine numerical correction each leaf. In other words, model needs to decide by how much prediction in newly build tree will be shifted, decreasing error.

### *3. Optimal Leaf Weight*

To get to optimal leaf weight formula Ω in second order approximation loss function needs to be expanded.

$$\widetilde{\mathcal{L}^{(t)}} = \sum_{i=1}^{n} \left[ g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2$$

Can be rewritten into following form.

$$= \sum_{j=1}^{T} \left[ \left( \sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left( \sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T$$

Out of which we get :

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

Formula uses sum of gradients and hessians including penalty term. After the formula is computed, weights are assigned to each leaf, clearly stating by how much each leaf adjusted the prediction.

I assumed following values for Gradients, Hessians and penalty term.

$g_i$ = [ -5, -3, -4, -2] , $h_i$ = [1, 1, 1, 1] and $\lambda$ = 0.5

Sum of Gradients and Hessians is computed.

$$\sum_{i \in I_j} g_i = -5 + (-3) + (-4) + (-2) = -14$$

$$\sum_{i \in I_j} h_i = 1 + 1 + 1 + 1 + \lambda = 4 = 4.5$$

And we compute weight.

$$w_j^* = -\frac{-14}{4.5} = \frac{14}{4.5} \approx 3.11$$

Value of 3.11 in general means that the leaf, for which weight was calculated will reduce error by adding 3.11 to current prediction. The formula is demonstrating what is happening inside each leaf when it generates prediction.

### *4. Score Quality Equation*

Following formula represents "quality" score function of tree structure. It evaluates how decision tree is, it includes sum of gradients and hessians as well as penalty terms. It is computed during training from each leaf, and based on the formula algorithm decides whether it is useful to include the split in the tree or not. If the tree improves the score, it is kept otherwise it must be pruned.

$$\widetilde{\mathcal{L}^{(t)}}(q) = -\frac{1}{2}\sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j} g_i\right)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T$$

In the following example I plugged in assumed values to demonstrate how formula is computed.

$$= -\frac{\mathbf{1}}{\mathbf{2}}\left(\frac{(\mathbf{-8})^{\mathbf{2}}}{\mathbf{4+1}} + \frac{(\mathbf{5})^{\mathbf{2}}}{\mathbf{3+1}}\right) + \mathbf{0.5 \cdot 2}$$

$$= -\frac{\mathbf{1}}{\mathbf{2}}\left(\frac{\mathbf{64}}{\mathbf{5}} + \frac{\mathbf{25}}{\mathbf{4}}\right) + \mathbf{1} = -\frac{\mathbf{1}}{\mathbf{2}}(\mathbf{12.8 + 6.25}) + \mathbf{1} = -\frac{\mathbf{1}}{\mathbf{2}} \cdot \mathbf{19.05} + \mathbf{1} =$$

$$\mathbf{-9.525 + 1} = = \textit{-8.525}$$

The value -8.525 means, that the tree (*q)* reduces the loss by -8.525, and most likely the tree is kept, since it improves accuracy of the model.

The next step algorithm takes is Boosting, the formula that was explained earlier in XGBoost section. The model uses the formula to adjust outputs (predictions) based on the trained tree. The exact structure of the tree was determined using previous steps and equations.

### *5. Final Prediction Formula*

$$\widehat{\boldsymbol{y}}_{\boldsymbol{\iota}} = \sum_{\boldsymbol{k=1}}^{\boldsymbol{K}} \boldsymbol{\eta} \cdot \boldsymbol{f}_{\boldsymbol{k}}(\boldsymbol{x}_{\boldsymbol{i}}), \quad \boldsymbol{f}_{\boldsymbol{k}} \in \boldsymbol{\mathcal{F}}$$

- $\widehat{\boldsymbol{y}}_{\boldsymbol{\iota}}$ is final prediction of XGBoost.
- $\boldsymbol{K}$ stands for number of boosting rounds (sequentially added models).
- $\boldsymbol{\mathcal{F}}$ represents space of all decision trees,
- $\boldsymbol{\eta}$ is learning rate parameter.
- $\boldsymbol{f}_{\boldsymbol{k}}(\boldsymbol{x}_{\boldsymbol{i}})$ represents leaf value from *k-th* tree for input $x_i$.

This equation reflects how final prediction is computed. In general outcomes of all trees are added together to form final prediction. This might not seem logical, but previous formulas explained are clearly showing, that at the end of each tree is not real prediction

of gold price, but rather the adjustment added to each estimate of *Y* variable. In simple words, each tree contributes to initial prediction by small adjustments until the whole model yields final prediction.

To clearly demonstrate how the formula is used I assumed that leaves $f_1$, $f_2$ and $f_3$ yielded following values for "corrections" of predicted gold price

$$\boldsymbol{f_1(x_i) = +30, \quad f_2(x_i) = -10, \quad f_3(x_i) = +20}$$

For each boosting round $\widehat{y_i^{(1)}}$, $\widehat{y_i^{(2)}}$ and $\widehat{y_i^{(3)}}$, the initial prediction is adjusted as follows. Initial prediction is in this case assumed to be 1900.

$$\boldsymbol{\widehat{y_i^{(1)}} = 1900 + 0.1 \cdot 30 = 1903}$$

$$\boldsymbol{\widehat{y_i^{(2)}} = 1903 + 0.1 \cdot (-10) = 1902}$$

$$\boldsymbol{\widehat{y_i^{(3)}} = 1902 + 0.1 \cdot 20 = 1904}$$

*1904* represents outcome of the model i.e. final prediction.

**XGBoost**

Extreme Gradient Boosting is machine learning model that based on ensemble method known as boosting. Unlike Random Forests, XGBoost builds its decision trees sequentially. Goal of each tree is to minimize the error from the previous one. Its core step when minimizing the errors is the minimizing objective loss function, where along with other equations the model adjusts the prediction within all boosting rounds.

Here is the breakdown of all steps in algorithm:

1. Algorithm starts with prediction of *Y* variable (gold price). If it is regression task, model takes mean of the target variable. Assuming there are 5 training subsets with following target values

   ***Y* = [1920, 1940, 1900, 1950, 1890]**

   The initial prediction is following.

$$\boldsymbol{\widehat{y_i^{(0)}} = \frac{1920 + 1940 + 1900 + 1950 + 1890}{5} = \frac{9600}{5} = 1920}$$

2. Compute gradients and hessians

Computing both gradients and hessians, determines how big of a adjustments for prediction model needs. Using assumed values for predictions after each boosting round we get:

| Observation | $y_i$ | $\widehat{y_i^{(0)}}$ | $g_i = \widehat{y_i^{(0)}} - y_i$ | $h_i$ |
|---|---|---|---|---|
| 1 | 1920 | 1920 | 1920 – 1920 = 0 | 1 |
| 2 | 1940 | 1920 | 1920 – 1940 = -20 | 1 |
| 3 | 1900 | 1920 | 1920 – 1900 = 20 | 1 |
| 4 | 1950 | 1920 | 1920 – 1950 = -30 | 1 |
| 5 | 1890 | 1920 | 1920 – 1890 = 30 | 1 |

After computing for each boosting round gradients and hessians are:

$g_i$ = [0, -20, 20, -30, 30], $h_i$ = [1, 1, 1, 1]

These values reflect direction and steepness of residuals and will be crucial for 5th boosting round to build first tree.

3. Fitting regression tree to gradients

Algorithm now fits the regression tree to model the residuals. The trees are now trained to predict direction of the errors as well as their magnitude. The tree is now splitting observations based on which features yield the lower error. As a result, algorithm assigns each training observation to leaf node based on value in its predictors.

4. Compute leaf scores

After all training observations have been assigned to leaf nodes, algorithm than computes prediction value for each leaf. In this step weight formula is used. To fully demonstrate this step I assumed, that 5 training observations used before, have been split into two leaf nodes. Observation 2 and 4 from 2nd step were assigned to leaf 1 and rest were assigned to leaf 2.

$$\boldsymbol{w_1 = -\frac{-20 + (-30)}{1 + 1 + 1} = -\frac{-50}{3} = \frac{50}{3} \approx 16.67}$$

$$w_2 = -\frac{0 + 20 + 30}{1 + 1 + 1 + 1} = -\frac{50}{4} = -12.5$$

These values are than multiplied by learning rate and added to the prediction to the current decision tree. The adjusted predictions are used in the next boosting round. One can think of above calculated value as "collection of corrections" of predictions. For example, algorithm adjusts predictions by multiplying learning rate by 16.67 and some by -12.5 depending on value of their features, that led them to specific nodes.

5. Adjust the prediction

   In this step model updates the prediction by using values computed in previous step and multiplies it by learning rate 0.1.

$$\widehat{y_1^{(1)}} = 1920 + 0.1 \cdot (-12.5) = 1920 - 1.25 = 1918.75$$

$$\widehat{y_2^{(1)}} = 1920 + 0.1 \cdot 16.67 = 1920 + 1.667 = 1921.67$$

$$\widehat{y_3^{(1)}} = 1920 + 0.1 \cdot (-12.5) = 1920 - 1.25 = 1918.75$$

$$\widehat{y_4^{(1)}} = 1920 + 0.1 \cdot 16.67 = 1920 + 1.667 = 1921.67$$

$$\widehat{y_5^{(1)}} = 1920 + 0.1 \cdot (-12.5) = 1920 - 1.25 = 1918.75$$

   Each observation is adjusted by correction * learning rate, based on which leaf was it assigned to in the previous step. Even tough previous step demonstrated only two leaf nodes, each observation must be adjusted.

6. Gradients and Hessians computed for new predictions

   Step number 2 is repeated with new values computed in step number 5 – adjusting the predictions. This step is repeated across all boosting rounds to ensure lowest error rate possible.

7. Final prediction

After all boosting rounds are completed, algorithm computes final prediction by summing up all values, that were computed in trained trees, each multiplied by learning rate. I assumed that last step in 6th step computed value 5.3. Final prediction computed is than:

$$\widehat{y_5^{(2)}} = 1918.75 + 0.1 \cdot 5.3 = 1918.75 + 0.53 = 1919.28$$

***Pros and Cons of the Model***

The research paper "*Tree Boosting with XGBoost*" by *Didrik Nielsen* identifies pros and cons of the XGBoost algorithm clearly and was used as a source for this section.

XGBoost possesses high ***predictive accuracy*** due to its ability to identify patterns of complex functions. Trees are built sequentially, where each tree improves the error from the previous one. As *Nielsen* states in the paper, "boosting can be very flexible in complex regions of feature space.

Another advantage of XGBoost is build-in ***feature selection***. Like Random Forests algorithm, XGBoost can identify, which features were used at the decision tree split the most, identifying their importance and how much it contributed to the objective loss function.

XGBoost also uses L1 and L2 regularization penalties incorporated it penalty term omega, which helps preventing model from ***overfitting***. The magnitude of the penalty term omega is determined by complexity of the trees, which helps reduce variance within the model. However, if the trees grow too deep and at the same time model computes many boosting rounds, it might overfit training data, that's why proper hyperparameter tuning is required

Although it depends on the performance of the device that algorithm is ran on, with larger dataset with many predictors, XGBoost might prove to be ***computationally difficult***. When I ran the algorithm in R space computational time varied across different tuning parameters, but in general compared to linear models like LASSO its computational time was exponentially higher.

There are numerous hyperparameters, that must be tuned when running algorithm, which makes is ***sensitive to parameter tuning***. To achieve high predictive accuracy and avoid under or overfitting parameters must be tuned properly. Throughout parameter tuning will be reflected on in the next section.

**Modeling the algorithm**

1. ***Data preparation and maintaining the time-series structure for modelling***

When it comes to data preparation and keeping time-series dependencies for XGBoost, the same process as was used in Random Forests algorithm will take place. Without keeping time-dependencies XGBoost would take observations for training at random, hence future data could be used to predict the past datapoints, which would be unreasonable. To maintain time dependencies, I used rolling window method, which takes fixed number of observations for training and predicts next observation to ensure, that only past values are trained to predict the future ones. To get as high prediction accuracy as possible algorithm was ran on different parameter for size of rolling windows, and the most accurate one was chosen. After fixed number of observations (rolling window) is trained, it predicts one observation in the future and moves by one, this procedure is repeated until rolling window reaches final observation.

2. ***Model training and parameter selection***

***2.1. Rolling Window Size***

To ensure diversity between training size parameter values, algorithm was run on size of 100 and 200 as size of rolling window.

***2.2. Objective Function***

This parameter must be specified in a code as regression task. Dataset's target variable is continuous not categorical, since it takes numeric values, thus XGBoost must treat as regression problem using mean squared error as the loss function.

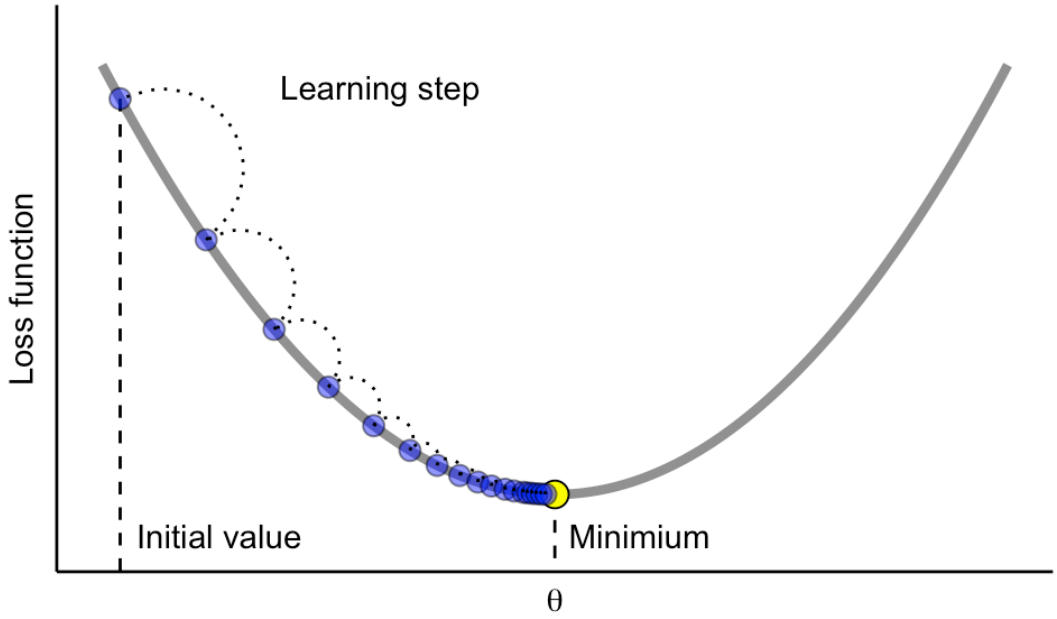
***2.3. Evaluation metric***

The algorithm was set to use root mean squared error RMSE as a metric of error to be minimized.

***2.4. Learning Rate***

Learning Rate parameter controls contribution of each tree to final prediction. According to many machine learning articles accessible on the internet it takes value within range [0,1]. The loss function is decreasing until algorithm reaches the

minimum value in error. The yellow dot on the diagram beneath reflects the point at the curve where model reached the minimal error.




(Figure 19, source ; Greenwell & Boehmke; Hands on Machine Learning)

If the learning rate is too large it can skip the minimal error, on the other hand if it is too small number of boosting rounds might not be enough to find the minimum. Both cases are illustrated on figure 20 in *XGBoost Appendix.*

### *2.5.Maximum tree depth*

As stated in book *Hands on Machine Learning* written by *Greenwell and Boehmke*, a greater tree depth allows model to find more complex patterns, but also introduces risk of overfitting, on the other hand when the tree depth is too small, the model might only capture only simple patterns. The book also states that typical range of tree depth is [3,8]. In this case, I ran algorithm using 3 and 6 as values for tree depth.

### *2.6. Subsample of observations*

The subsample parameter determines percentage of chosen training data randomly shuffled into the training window. Since I maintained time dependencies by using rolling

window this does not break it. Having randomly shuffled training data ensures that variance is reduced and at the same time reduces risk of overfitting. In this case the value was fixed at 0.8 = 80%. If the value is too high algorithm would use all training data for the tree, which may cause overfitting, on the other hand is the parameter value is set too low it may decrease predictive performance.

### *2.7. Feature subsamples per tree*

This parameter determines how many parameters are chosen for each tree randomly. Since predictors in the dataset are assumed to be highly correlated, this parameter ensures diversity of predictors among trees. The value was set to 0.8 = 80%

### *2.8. Number of rounds*

Number of boosting rounds determines number of boosting iterations i.e. number of trees built. In general number of boosting rounds controls complexity of a model and how much it learns. If the number is too small, the algorithm may not be able to fully identify patterns and underfit the real values. If the number of boosting rounds is too large it may lead to overfitting. Most of the articles and research papers discussed number of boosting rounds in range [100,1000].

| | roll_window | eta | max_depth | nrounds | train_mae | test_mae |
|---|---|---|---|---|---|---|
| 1 | 100 | 0.1 | 3 | 100 | 2.2736253947 | 11.79629 |
| 2 | 200 | 0.1 | 3 | 100 | 4.1999907909 | 11.62697 |
| 3 | 100 | 0.3 | 3 | 100 | 0.1256374111 | 12.44145 |
| 4 | 200 | 0.3 | 3 | 100 | 0.8559079224 | 12.40673 |
| 5 | 100 | 0.5 | 3 | 100 | 0.0091153895 | 13.63848 |
| 6 | 200 | 0.5 | 3 | 100 | 0.2225380938 | 14.12360 |
| 7 | 100 | 0.1 | 6 | 100 | 0.7661324717 | 11.77768 |
| 8 | 200 | 0.1 | 6 | 100 | 1.3349266664 | 11.59542 |
| 9 | 100 | 0.3 | 6 | 100 | 0.0007495425 | 12.85023 |
| 10 | 200 | 0.3 | 6 | 100 | 0.0088513202 | 12.85699 |
| 11 | 100 | 0.5 | 6 | 100 | 0.0004123079 | 13.92095 |
| 12 | 200 | 0.5 | 6 | 100 | 0.0004480381 | 14.51633 |
| 13 | 100 | 0.1 | 3 | 200 | 0.4738053006 | 11.37904 |
| 14 | 200 | 0.1 | 3 | 200 | 1.6577083594 | 11.35039 |
| 15 | 100 | 0.3 | 3 | 200 | 0.0024042724 | 12.52678 |
| 16 | 200 | 0.3 | 3 | 200 | 0.0954837240 | 12.35840 |
| 17 | 100 | 0.5 | 3 | 200 | 0.0004209873 | 13.76960 |
| 18 | 200 | 0.5 | 3 | 200 | 0.0064820220 | 13.91931 |
| 19 | 100 | 0.1 | 6 | 200 | 0.0233080237 | 11.74203 |
| 20 | 200 | 0.1 | 6 | 200 | 0.1022998996 | 11.43042 |
| 21 | 100 | 0.3 | 6 | 200 | 0.0004325746 | 12.73584 |
| 22 | 200 | 0.3 | 6 | 200 | 0.0004589802 | 12.88191 |
| 23 | 100 | 0.5 | 6 | 200 | 0.0003885165 | 14.43882 |
| 24 | 200 | 0.5 | 6 | 200 | 0.0004067752 | 14.46397 |
| 25 | 100 | 0.1 | 3 | 500 | 0.0094212501 | 11.41209 |
| 26 | 200 | 0.1 | 3 | 500 | 0.1759145652 | 11.04937 |
| 27 | 100 | 0.3 | 3 | 500 | 0.0004283505 | 12.41754 |
| 28 | 200 | 0.3 | 3 | 500 | 0.0004981608 | 12.43289 |
| 29 | 100 | 0.5 | 3 | 500 | 0.0003854070 | 13.86806 |
| 30 | 200 | 0.5 | 3 | 500 | 0.0004089991 | 13.70652 |
| 31 | 100 | 0.1 | 6 | 500 | 0.0005406645 | 11.68685 |
| 32 | 200 | 0.1 | 6 | 500 | 0.0005653016 | 11.44484 |
| 33 | 100 | 0.3 | 6 | 500 | 0.0004062899 | 12.67138 |
| 34 | 200 | 0.3 | 6 | 500 | 0.0004228903 | 12.60251 |
| 35 | 100 | 0.5 | 6 | 500 | 0.0003710870 | 14.19024 |
| 36 | 200 | 0.5 | 6 | 500 | 0.0003849695 | 14.38261 |

### *3. Evaluation of the model*

The number of possible combinations using all parameters is too large, and since the algorithm is computationally costly I decided to use following values for parameters.

- Rolling window size : [100,200]
- Learning rate : [0.1, 0.3, 0.5]
- Maximum tree depth [3, 6]
- Number of boosting rounds [100, 200, 500]

Algorithm was ran on 36 times using all possible parameter combinations. Algorithm was running on loop, where each loop used different combination of hyperparameters, what accounted for approximately 3 hours, which supports the mentioned fact about XGBoost being computationally expensive. Overall training error rate ranged ~ [0.0004, 4.19] while testing error rate ranged ~ [11,14].

After obtaining the results some insights could be concluded. Firstly, I divided combinations based on rolling window. Combinations with rolling window 200 resulted in slightly lower error as well as higher standard deviation (1.19). I decided to further consider only combinations of parameters with rolling window of 200, since the model uses more historical data, which might lead to higher flexibility and ability to learn more complex patterns. After this division, there was 18 combinations to consider.

Next step in choosing optimal parameter values was comparing maximum tree depth (x-axis) with learning rate (y-axis) using heatmap. Heatmap was plotted using all three values for boosting rounds (100, 200, 500). For heatmap see section XGBoost in Appendix**.** From the heatmap it is easy to conclude that lowest learning rate yields lowest error, which might suggest that higher learning rates may skip mentioned “minimum” on the parabola plot shown earlier in the section explaining hyperparameters. Interesting situation occurs when looking at the maximum tree depth. Tree depth of 3 resulted in lower error compared to tree depth 6, but only when number of boosting rounds was 200 and 500. Based on this observation I decided to compare errors considering only maximum tree depth parameter. The results showed that value of 3 for tree depth resulted in lower error. When comparing boosting rounds logically most boosting rounds (500) resulted in lowest error, following 200 and 100. However, to generalize all 36 combinations, looking at training and testing error, it might be concluded that most of the combinations are potentially overfitting. Only four observations yielded training error greater than one. Three of them used learning rate of 0.1 and tree depth of 3, the fourth one used same

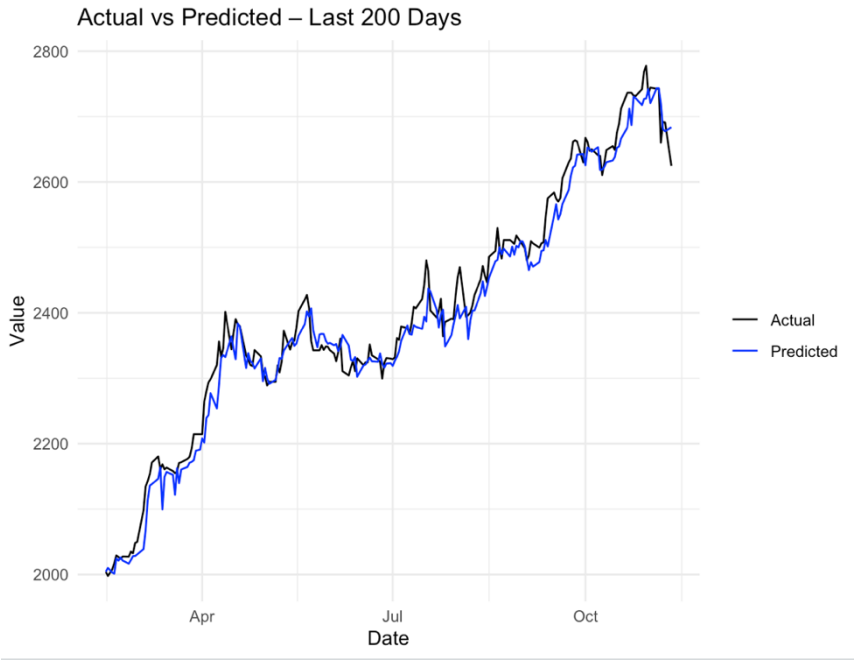
learning rate but tree depth of 6. Based on previous observations I decided to use rolling window = 200, learning rate = 0.1, tree depth = 3 and number of boosting = 100. This combination of parameters resulted in training error of 4.2 and testing error of 11.63 and are within first 10 best performing combinations out of all 36. Even though some parameter combinations yielded lower error, I decided to use this one in the final model, since training error rate is substantially higher than for the rest of parameter combinations, hence might be least prone to overfitting.

Following table represents the results from running XGBoost using chosen parameters.

Looking at both training error metrics it is easy to conclude that both errors are very low, meaning that model is fitting the training data very accurately. The difference between RMSE and MAE values shows that there are some large errors, because RMSE penalizes large errors more. When it comes to testing error there is also a gap between the two metrics, which suggests, that some outliers might be present in the dataset. The reason why there is a gap between training and testing values, might be a fact, that I used rolling window method to maintain time dependencies. Model is being trained on first 200 observations, predicts 201$^{st}$ observation and moves in time by one. In other words, after model predicted 201$^{st}$ values it now uses 2$^{nd}$ to 201$^{st}$ to predict 202$^{nd}$ observation and so on. Although model is always tested on unseen values, the training subsets are overlapping significantly across iterations, meaning that model reuses the datapoints multiple times.

Following plots and histograms will help evaluate model further.

I did plot actual and predicted values using plot over time. To clearly see the difference, I used last 200 observations. It is obvious, that the model tends to underpredict actual values most of the time. While model seems to capture overall upward trend, it seems like it does not predict well when it comes to sudden spikes in gold price. The model is obviously also not very accurate when it comes to spikes downward, where it adjusts slowly. One of possible reason this is occurring, might be, that gold itself require more than just one- or two-day lags. Another logical way to improve model's performance could be adding volatility index as one of the predictors, where model would be able to recognize more and less volatile periods and adjusts predictions accordingly.




(Figure 21; source ; R interface)

Histogram in *XGBoost Appendix* displays distribution of residuals. In general, they are centered around zero, which means that model is not making systematic errors, and that model is considered unbiased. However, histogram also displays that distribution of residuals is right-skewed, which supports underestimation fact drawn from previous plot. It is also possible to see couple of outliers on both sides, but that could be the reason of sudden spikes in gold price, where model failed to predict spike, which is expected.

# 6. Conclusion

The thesis compared four predictive models each resulting in different result. The worst performing is LASSO regression, potentially because high- dimensionality or non-linear relationships between variables. Than two machine learning models – Random Forests and XGboost yielded pretty decent error. The two models possibly perform substantially better than LASSO regression because they deal better with non-linearity and can identify complex patterns. Lastly ARIMA model, performed surprisingly the best. The possible reason might be, that real world daily price of gold is influences by large number of factors, some of which cant even be projected into numbers. Thus the conclusion of the thesis is, that when it comes to complex financial data like gold daily price, it might be good idea to just use past values as predictors for forecasting.

# 7. Appendix

## 7.1.General Data processing

Dataset before omitting NA values.

| | Date | GOLD_SPOT_PRICE | ALUMINIUM_DAILY | CORN_DAILY | GASOLINE_DAILY | LONDON_SEX_DAILY | LUMBER_DAILY | NYSE_DAILY |
|---|---|---|---|---|---|---|---|---|
| 9923 | 1997-06-04 | 341.75 | NA | 2.7125 | 0.583 | NA | 363.70 | NA |
| 9924 | 1997-06-05 | 340.80 | NA | 2.7425 | 0.577 | NA | 367.10 | NA |
| 9925 | 1997-06-06 | 344.00 | NA | 2.7400 | 0.550 | NA | 357.10 | NA |
| 9926 | 1997-06-09 | 344.25 | NA | 2.7400 | 0.538 | NA | 347.10 | NA |
| 9927 | 1997-06-10 | 343.75 | NA | 2.7175 | 0.540 | NA | 354.70 | NA |
| 9928 | 1997-06-11 | 343.80 | NA | 2.7250 | 0.541 | NA | 352.40 | NA |
| 9929 | 1997-06-12 | 342.20 | NA | 2.7125 | 0.546 | NA | 362.40 | NA |
| 9930 | 1997-06-13 | 341.40 | NA | 2.7150 | 0.548 | NA | 369.10 | NA |
| 9931 | 1997-06-16 | 341.50 | NA | 2.6850 | 0.554 | NA | 373.90 | NA |
| 9932 | 1997-06-17 | 342.15 | NA | 2.7025 | 0.559 | NA | 372.00 | NA |
| 9933 | 1997-06-18 | 341.60 | NA | 2.6650 | 0.540 | NA | 381.30 | NA |
| 9934 | 1997-06-19 | 340.20 | NA | 2.6525 | 0.533 | NA | 385.90 | NA |

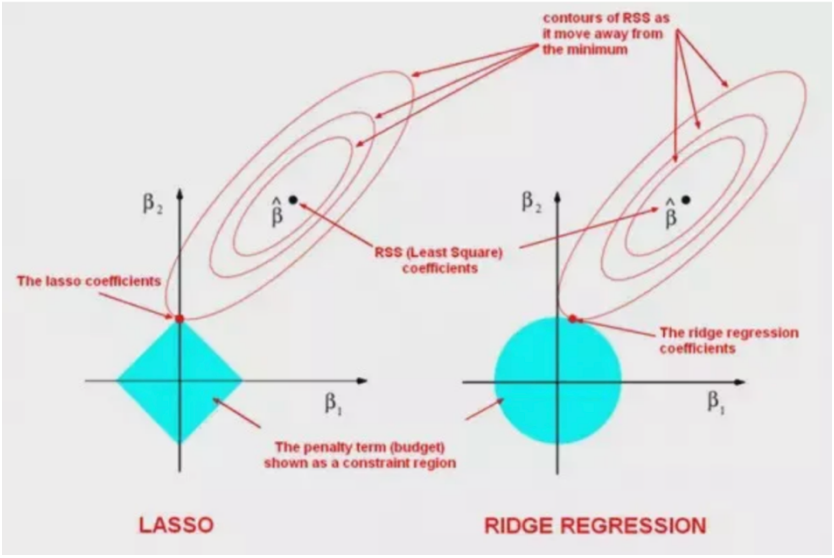
*(Figure 1, source: R Interface.)*

Dataset after omitting NA values.

| | Date | GOLD_SPOT_PRICE | ALUMINIUM_DAILY | CORN_DAILY | GASOLINE_DAILY | LONDON_SEX_DAILY | LUMBER_DAILY | NYSE_DAILY |
|---|---|---|---|---|---|---|---|---|
| 1 | 2015-03-02 | 1212.50 | 1799.50 | 3.8800 | 1.742 | 2450 | 296.30 | 18539.37 |
| 2 | 2015-03-03 | 1212.75 | 1790.75 | 3.9100 | 1.763 | 2405 | 292.60 | 18564.48 |
| 3 | 2015-03-04 | 1199.50 | 1797.25 | 3.8950 | 1.730 | 2400 | 289.20 | 18344.46 |
| 4 | 2015-03-05 | 1202.00 | 1798.75 | 3.9050 | 1.703 | 2403 | 286.80 | 18308.42 |
| 5 | 2015-03-06 | 1175.75 | 1778.25 | 3.8600 | 1.691 | 2435 | 281.80 | 18518.34 |
| 6 | 2015-03-09 | 1168.50 | 1775.00 | 3.8875 | 1.686 | 2454 | 286.60 | 18539.14 |
| 7 | 2015-03-10 | 1162.00 | 1756.75 | 3.8800 | 1.634 | 2435 | 285.00 | 18406.05 |
| 8 | 2015-03-11 | 1150.00 | 1740.00 | 3.9100 | 1.642 | 2468 | 281.60 | 18513.74 |
| 9 | 2015-03-12 | 1152.25 | 1742.50 | 3.8850 | 1.618 | 2490 | 279.40 | 18712.48 |

*(Figure 2, source: R Interface)*

## 7.2.Rolling Cross Validation




*(figure 4 source : Medium)*

Next screenshot represent results from running cross validation on multiple combinations. Based on the error of the combination, best performing parameters were chosen, and algortihm was ran once again on fixed parameters.

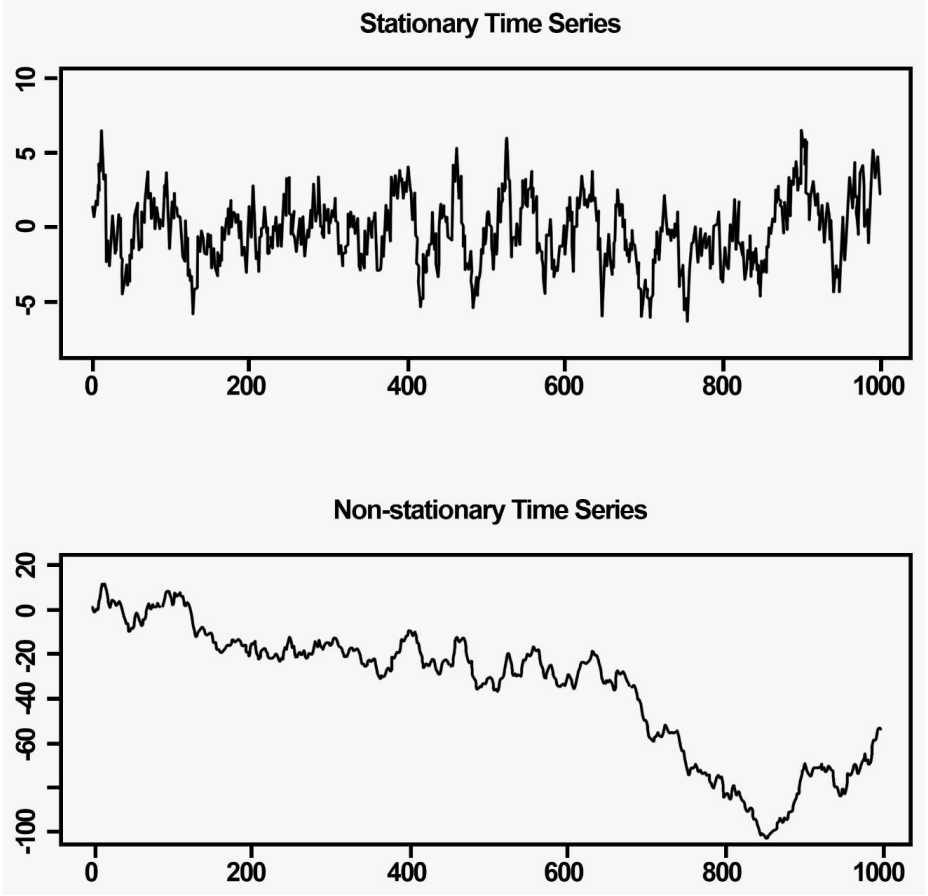
| | Window_Size | Horizon | Mean_RMSE | Mean_MAE | Num_Selected_Features |
|---|---|---|---|---|---|
| 1 | 100 | 1 | 10.079773 | 10.079773 | 24 |
| 2 | 100 | 3 | 12.860755 | 11.386288 | 31 |
| 3 | 100 | 5 | 14.482042 | 12.487067 | 31 |
| 4 | 100 | 10 | 17.520840 | 14.946902 | 21 |
| 5 | 100 | 20 | 22.647081 | 19.293930 | 26 |
| 6 | 200 | 1 | 9.555695 | 9.555695 | 34 |
| 7 | 200 | 3 | 11.523380 | 10.071357 | 41 |
| 8 | 200 | 5 | 12.380677 | 10.481535 | 47 |
| 9 | 200 | 10 | 14.056042 | 11.733595 | 45 |
| 10 | 200 | 20 | 16.694711 | 13.877912 | 33 |
| 11 | 300 | 1 | 9.265992 | 9.265992 | 30 |
| 12 | 300 | 3 | 11.039832 | 9.552461 | 33 |
| 13 | 300 | 5 | 11.680190 | 9.762308 | 35 |
| 14 | 300 | 10 | 12.571780 | 10.299951 | 28 |
| 15 | 300 | 20 | 14.015647 | 11.426743 | 32 |
| 16 | 400 | 1 | 9.519077 | 9.519077 | 41 |
| 17 | 400 | 3 | 11.098118 | 9.607892 | 44 |
| 18 | 400 | 5 | 11.621454 | 9.711112 | 31 |
| 19 | 400 | 10 | 12.415779 | 10.125185 | 38 |
| 20 | 400 | 20 | 13.600478 | 10.982912 | 34 |
| 21 | 500 | 1 | 9.841144 | 9.841144 | 40 |
| 22 | 500 | 3 | 11.437928 | 9.861158 | 35 |
| 23 | 500 | 5 | 11.971048 | 9.963031 | 29 |
| 24 | 500 | 10 | 12.726222 | 10.307551 | 37 |
| 25 | 500 | 20 | 13.665784 | 10.922311 | 34 |

Following illustration is screenshot taken from R Interface. It represents coefficients for each variable, that algorithm computed using rolling cross validation. All other variable coefficients, that are not displayed in the screenshot, were set to 0, and removed from analysis.

```
Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)              5.635e+01  3.712e+02   0.152 0.879442
CORN_DAILY              -1.899e+00  4.869e+00  -0.390 0.696875
GASOLINE_DAILY          -3.517e+00  6.569e+00  -0.535 0.592806
LONDON_SEX_DAILY         1.836e-02  8.514e-03   2.156 0.031936 *
NYSE_DAILY              -7.900e-04  5.948e-03  -0.133 0.894439
S_P500_DAILY            -5.379e-02  2.422e-02  -2.221 0.027209 *
SHANGHAI_COMPOSITE      -2.374e-02  1.175e-02  -2.020 0.044362 *
SILVER_DAILY             2.264e+01  1.939e+00  11.679  < 2e-16 ***
USD_EUR                  2.720e+01  6.457e+01   0.421 0.673967
USD_INR_DAILY           -1.938e+00  7.364e+00  -0.263 0.792614
US_INFLATION_DAILY      -7.652e+01  3.062e+01  -2.499 0.013048 *
USD_CHY                 -4.440e+01  1.904e+01  -2.333 0.020402 *
BTC_DAILY                1.962e-04  2.157e-04   0.909 0.363907
GOLD_SPOT_PRICE_Lag1     6.720e-01  5.651e-02  11.890  < 2e-16 ***
GOLD_SPOT_PRICE_Lag2     2.097e-01  5.475e-02   3.831 0.000159 ***
ALUMINIUM_DAILY_Lag1    -8.426e-03  1.415e-02  -0.596 0.551953
LONDON_SEX_DAILY_Lag2   -2.052e-02  8.482e-03  -2.420 0.016200 *
LUMBER_DAILY_Lag1       -3.725e-02  2.468e-02  -1.509 0.132349
NYSE_DAILY_Lag2          3.109e-03  5.905e-03   0.526 0.599037
PLATINUM_DAILY_Lag1     -4.968e-02  6.916e-02  -0.718 0.473162
PLATINUM_DAILY_Lag2     -2.001e-02  6.848e-02  -0.292 0.770350
S_P500_DAILY_Lag1        7.027e-02  2.427e-02   2.895 0.004100 **
SILVER_DAILY_Lag1       -7.739e+00  2.920e+00  -2.651 0.008507 **
SILVER_DAILY_Lag2       -9.276e+00  2.400e+00  -3.865 0.000139 ***
USD_EUR_Lag2             4.672e+01  6.917e+01   0.676 0.499924
WHEAT_DAILY_Lag2        -5.714e+00  2.749e+00  -2.079 0.038553 *
USD_INR_DAILY_Lag1       4.735e+00  9.105e+00   0.520 0.603417
USD_INR_DAILY_Lag2       2.164e+00  6.987e+00   0.310 0.757050
US_INFLATION_DAILY_Lag1  9.499e+01  3.092e+01   3.072 0.002345 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

### 7.3.ARIMA

Following illustration shows difference between stationary and non-stationary time series data.




Following screenshot was taken in R interface. The code prints out results from ARIMA model.

```
> cat("   Training RMSE:", round(training_rmse, 4), "\n")
   Training RMSE: 13.421
> cat("   Training MAE :", round(training_mae, 4), "\n")
   Training MAE : 9.5235
> cat("   Testing RMSE :", round(testing_rmse, 4), "\n")
   Testing RMSE : 14.8673
> cat("   Testing MAE  :", round(testing_mae, 4), "\n")
   Testing MAE  : 9.9658
```

| Error Metrics | Train | Test |
|---|---|---|
| **RMSE** | 13.42 | 14.86 |
| **MAE** | 9.52 | 9.96 |

Next histogram visualized in R displays training RMSE across training windows, where most errors range up to 10.

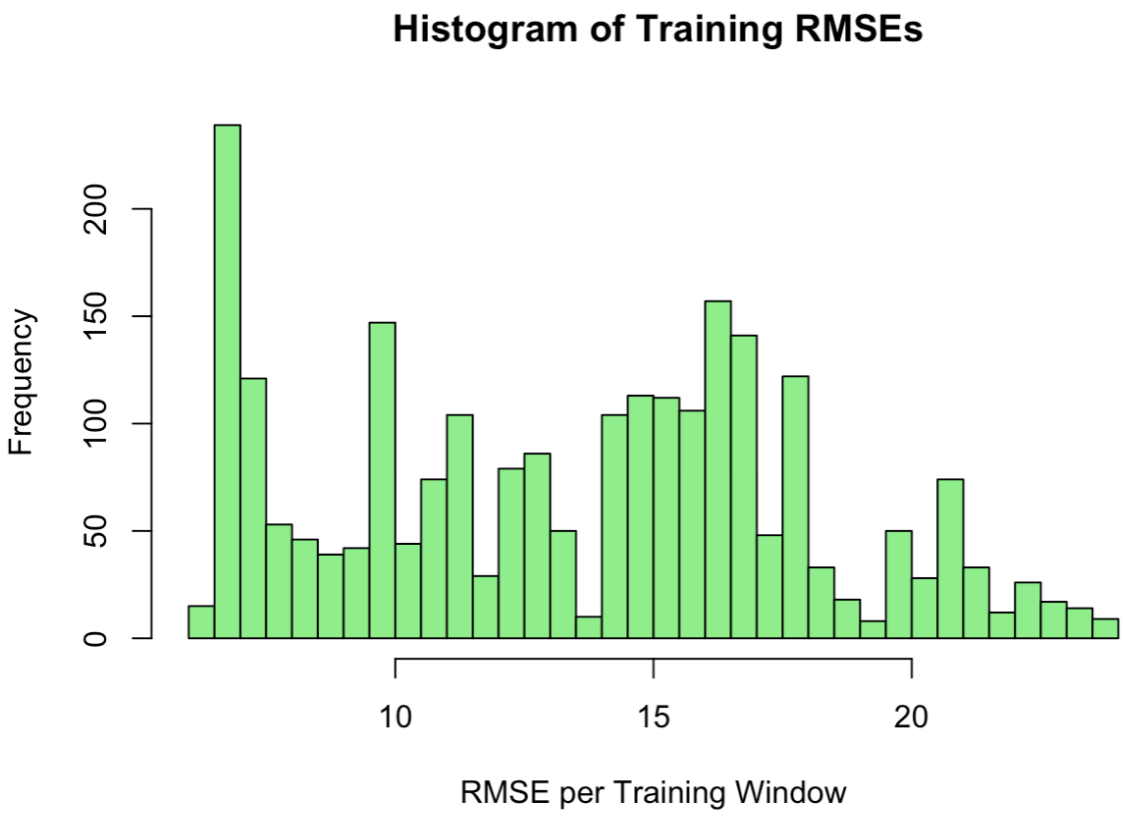
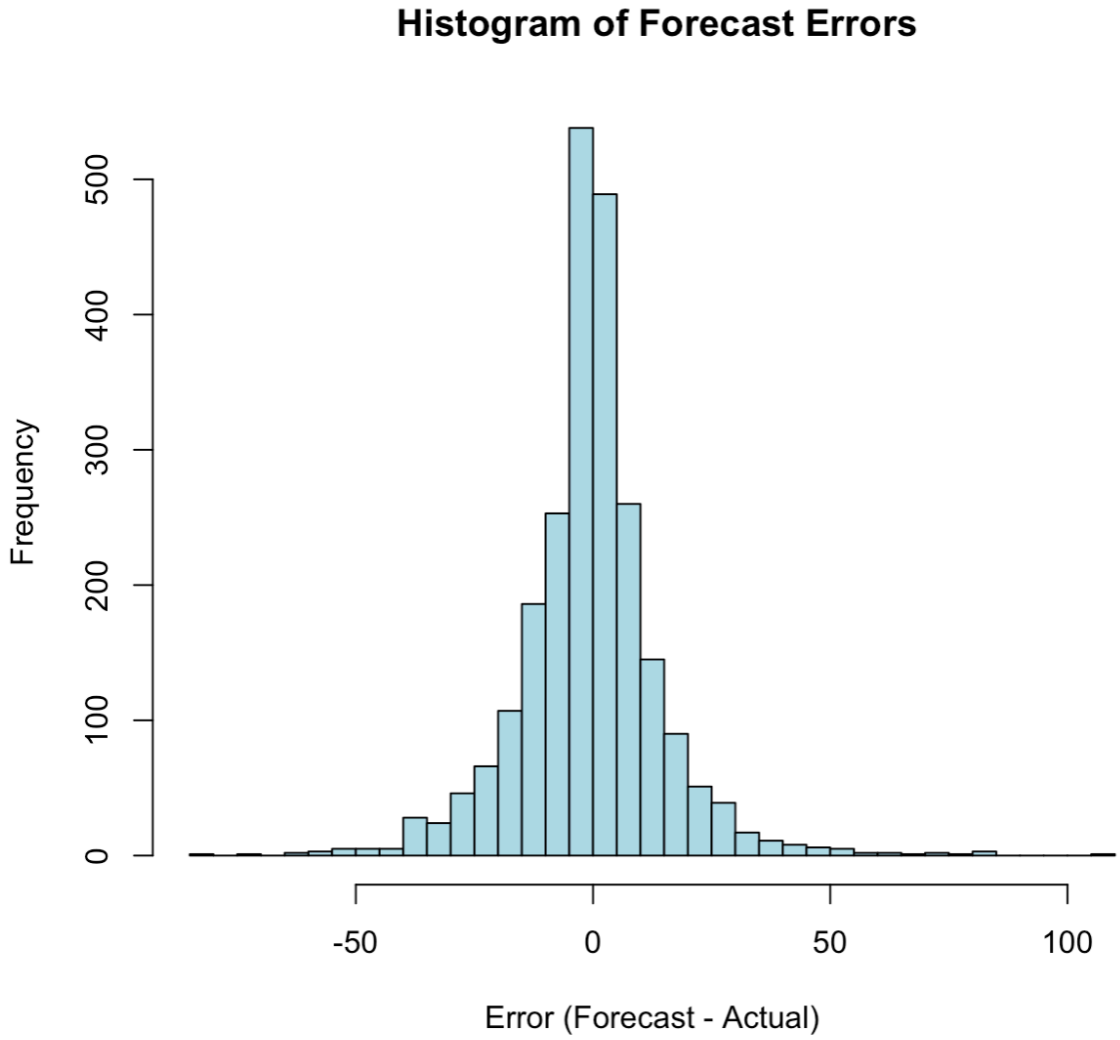
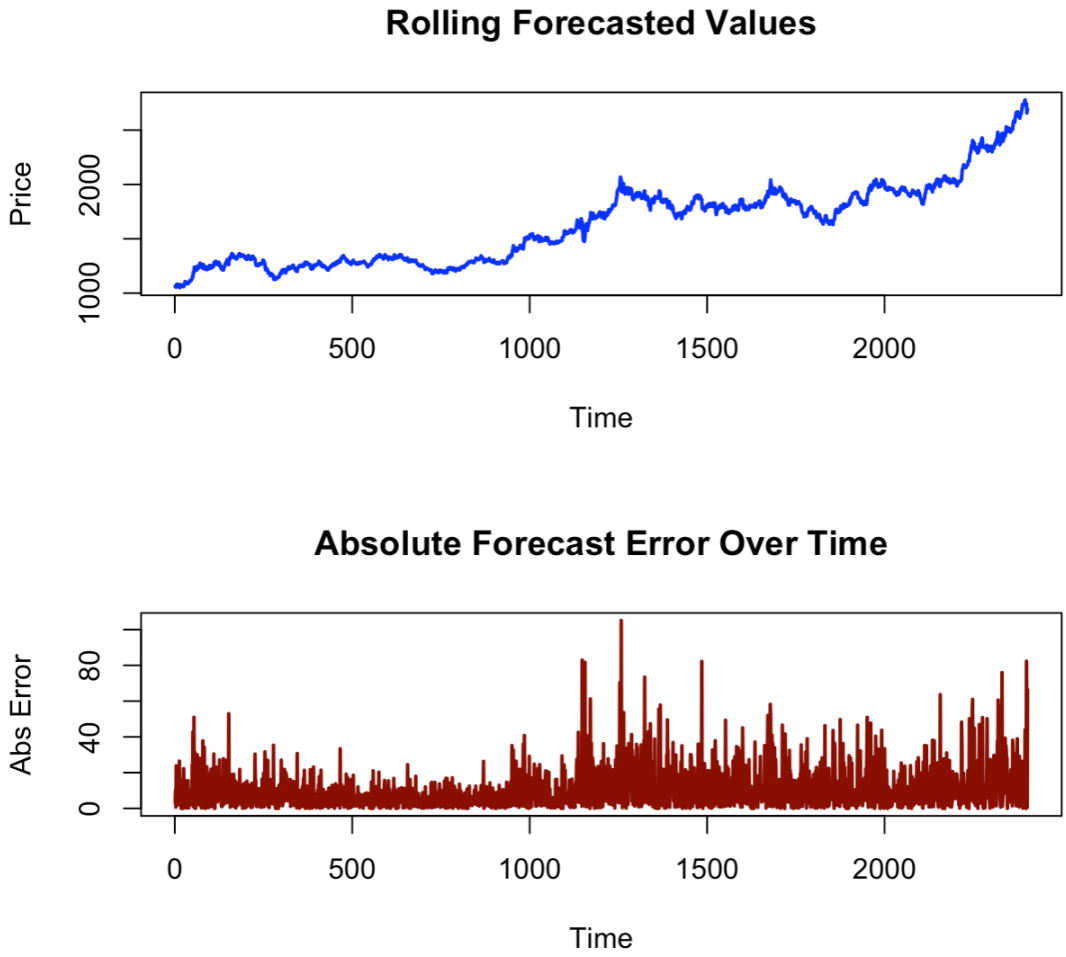



Figure 6 displaying distribution of residuals for ARIMA model.




*(Figure 6; source : R Interface)*

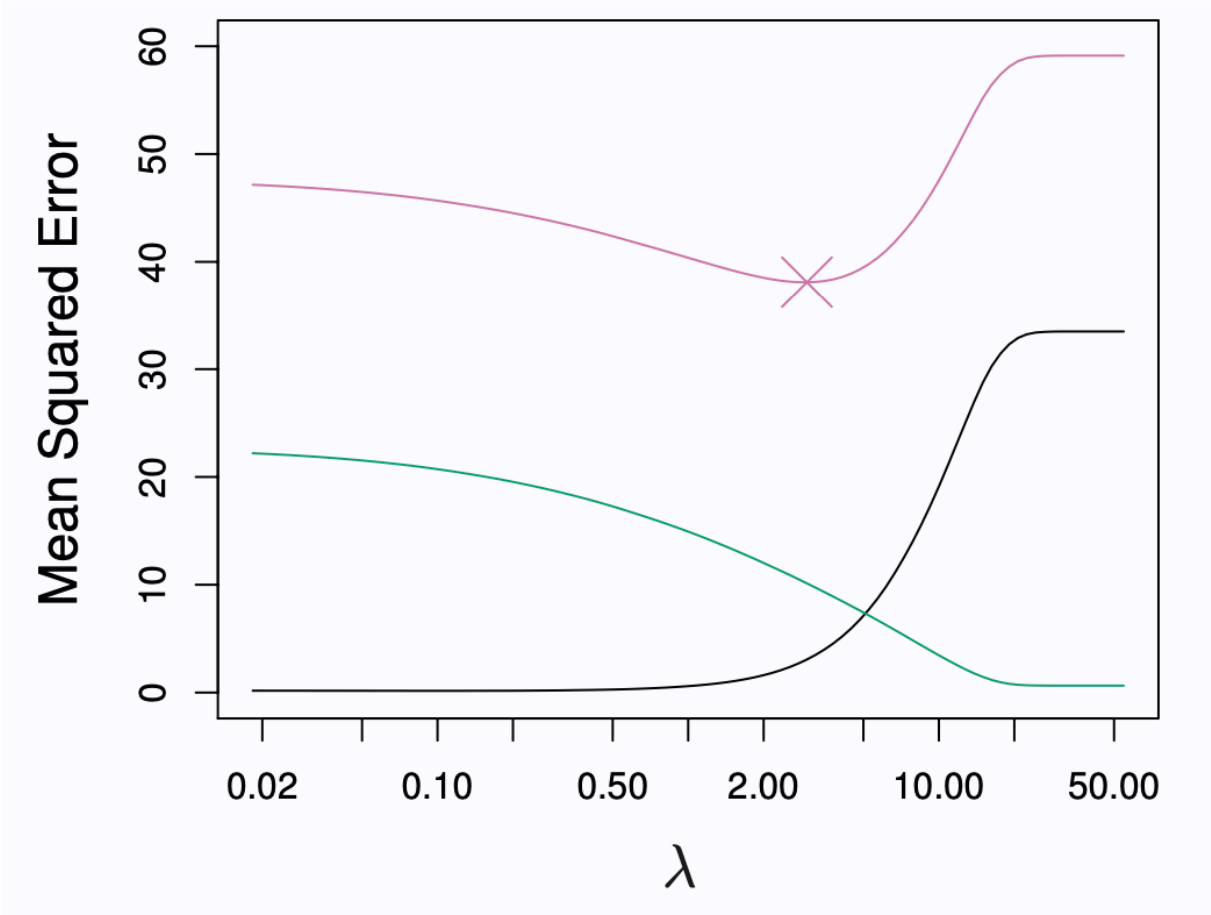
Figure 7.




*(Figure 7; source : R Interface)*

## 7.4.Appendix Lasso

Figure 8 representing Bias – Variance Trade-off




(Figure 8; Bias-Variance Tradeoff, source : Lecture presentation 3; Linear Regression

Following table contains error metrics

| Metric | Training | Validation | Testing |
|---|---|---|---|
| RMSE | 12.23 | 17.2 | 21.6 |
| MAE | 8.16 | 12.94 | 16.41 |
| ME | -7.8e-14 | -3.16 | -11.74 |

Following plot displays actual and predicted values using LASSO regression. Since there are too many observations, it is difficult to see actual residuals.

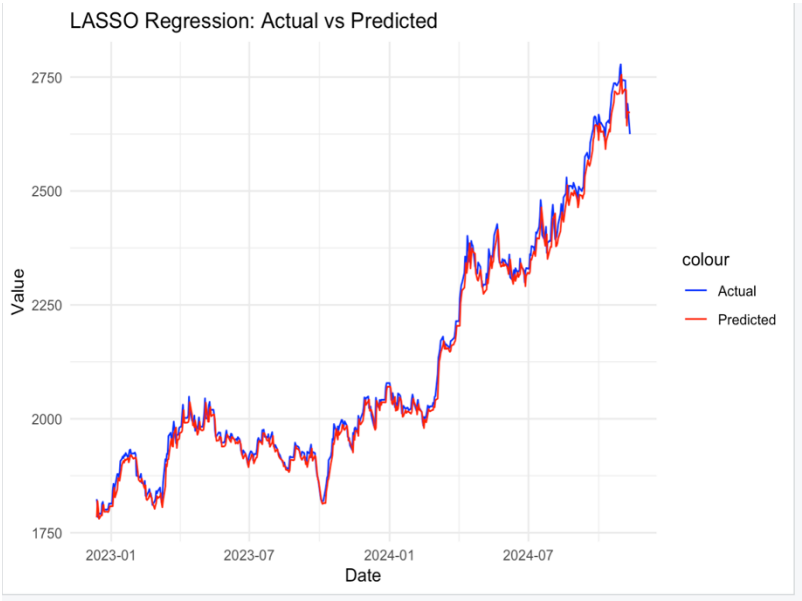
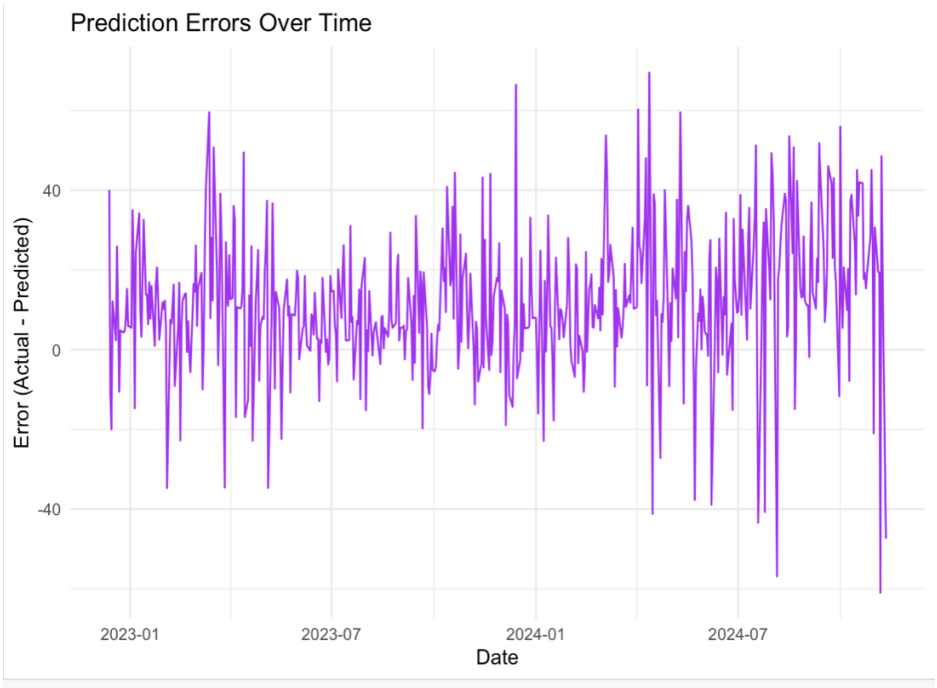
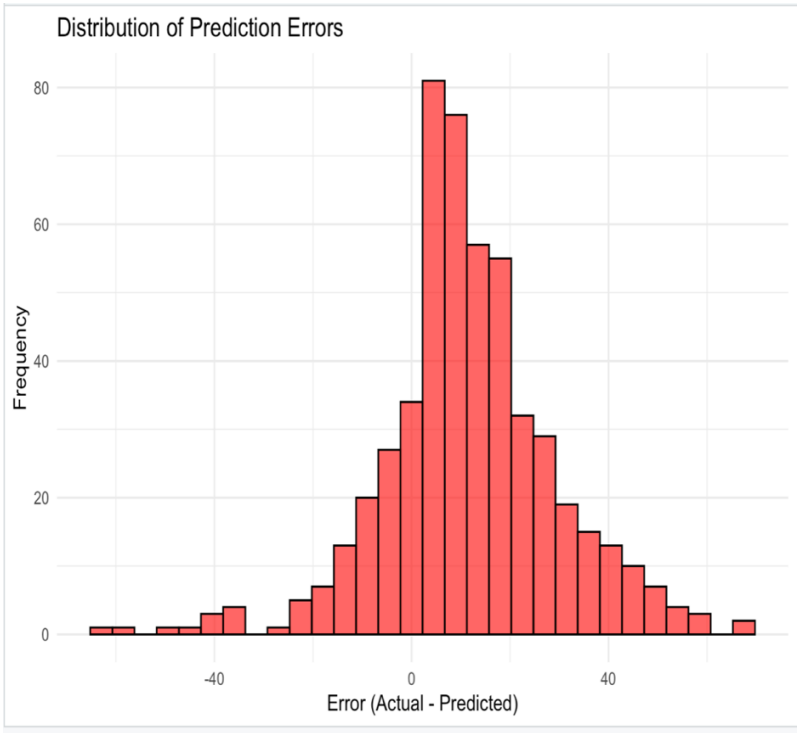



Figure 10 displaying residuals over time.




*(Figure 10;; R Interface)*

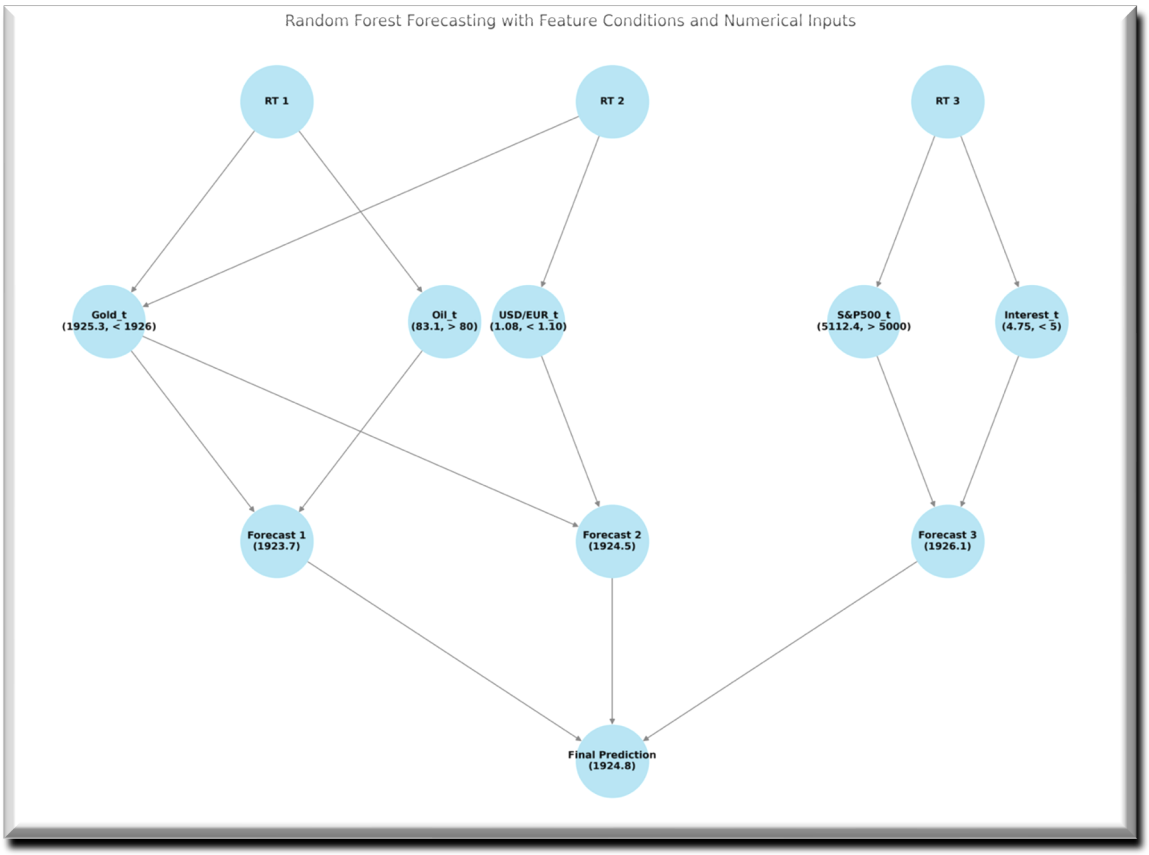
Figure 11 ; Distribution of residuals using LASSO regression




*(Figure 11; R Interface)*

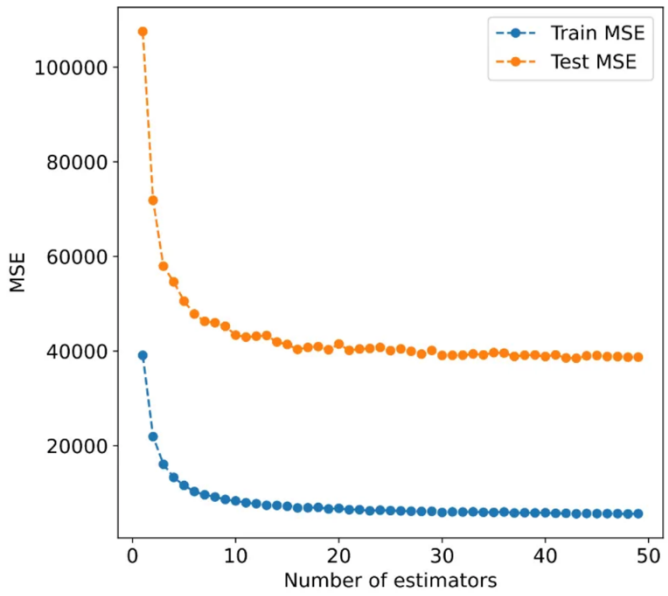
## 7.5.Appendix Random Forests

Random Forests diagram




(Figure 12; source : ChatGpt)

Train and Test error against increasing number of estimators.




(Figure 13; source : Medium, Alex Molas, Can Random Forests Overfit ?)

Following table shows different error metrics for various parameter values.

| roll_window | ntree | mtry | test_rmse | test_mae |
|---|---|---|---|---|
| 100 | 100 | 3 | 21.4211 | 15.2668 |
| 100 | 100 | 5 | 20.2594 | 14.4025 |
| 100 | 100 | 7 | 19.4701 | 13.7796 |
| 100 | 100 | 10 | 18.9733 | 13.4141 |
| 100 | 300 | 3 | 21.3988 | 15.2691 |
| 100 | 300 | 5 | 20.1267 | 14.3122 |
| 100 | 300 | 7 | 19.4098 | 13.7406 |
| 100 | 300 | 10 | 18.9793 | 13.3772 |
| 100 | 500 | 3 | 21.3333 | 15.2395 |
| 100 | 500 | 5 | 20.0628 | 14.2322 |
| 100 | 500 | 7 | 19.4818 | 13.8058 |
| 100 | 500 | 10 | 18.8400 | 13.3272 |
| 150 | 100 | 3 | 22.8158 | 15.9892 |
| 150 | 100 | 5 | 21.1630 | 14.8187 |
| 150 | 100 | 7 | 20.3504 | 14.2205 |
| 150 | 100 | 10 | 19.4723 | 13.5617 |
| 150 | 300 | 3 | 22.7255 | 15.9602 |
| 150 | 300 | 5 | 20.9428 | 14.6598 |
| 150 | 300 | 7 | 20.2213 | 14.1006 |
| 150 | 300 | 10 | 19.4152 | 13.5117 |
| 150 | 500 | 3 | 22.5068 | 15.8592 |
| 150 | 500 | 5 | 20.9722 | 14.6815 |
| 150 | 500 | 7 | 20.1882 | 14.0761 |
| 150 | 500 | 10 | 19.4263 | 13.5292 |
| 200 | 100 | 3 | 22.8690 | 16.0083 |
| 200 | 100 | 5 | 21.2886 | 14.7196 |
| 200 | 100 | 7 | 20.3327 | 14.0633 |
| 200 | 100 | 10 | 19.2841 | 13.2959 |
| 200 | 300 | 3 | 22.8693 | 15.9764 |
| 200 | 300 | 5 | 21.0928 | 14.7037 |
| 200 | 300 | 7 | 20.0411 | 13.9092 |
| 200 | 300 | 10 | 19.1690 | 13.2653 |
| 200 | 500 | 3 | 22.9154 | 15.9769 |
| 200 | 500 | 5 | 21.0465 | 14.6683 |
| 200 | 500 | 7 | 20.0891 | 13.9403 |
| 200 | 500 | 10 | 19.0917 | 13.2166 |

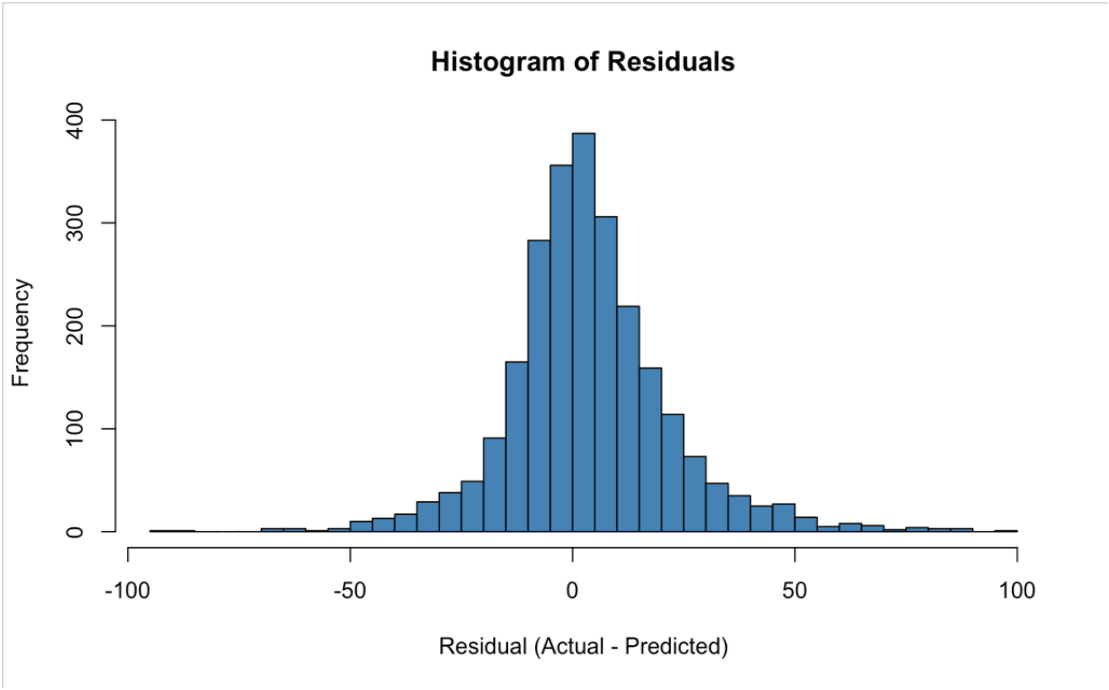
Code, that run Random Forests on fixed parameters yielded following results.

```
> my_config_results <- rf_run_with_metrics(X, y, roll_window = 100, ntree = 500, mtry = 10)
> print(my_config_results)
  roll_window ntree mtry train_rmse train_mae test_rmse test_mae
1         100   500   10   5.177144  3.587425  18.94273 13.37629
```

Following table contains error metrics

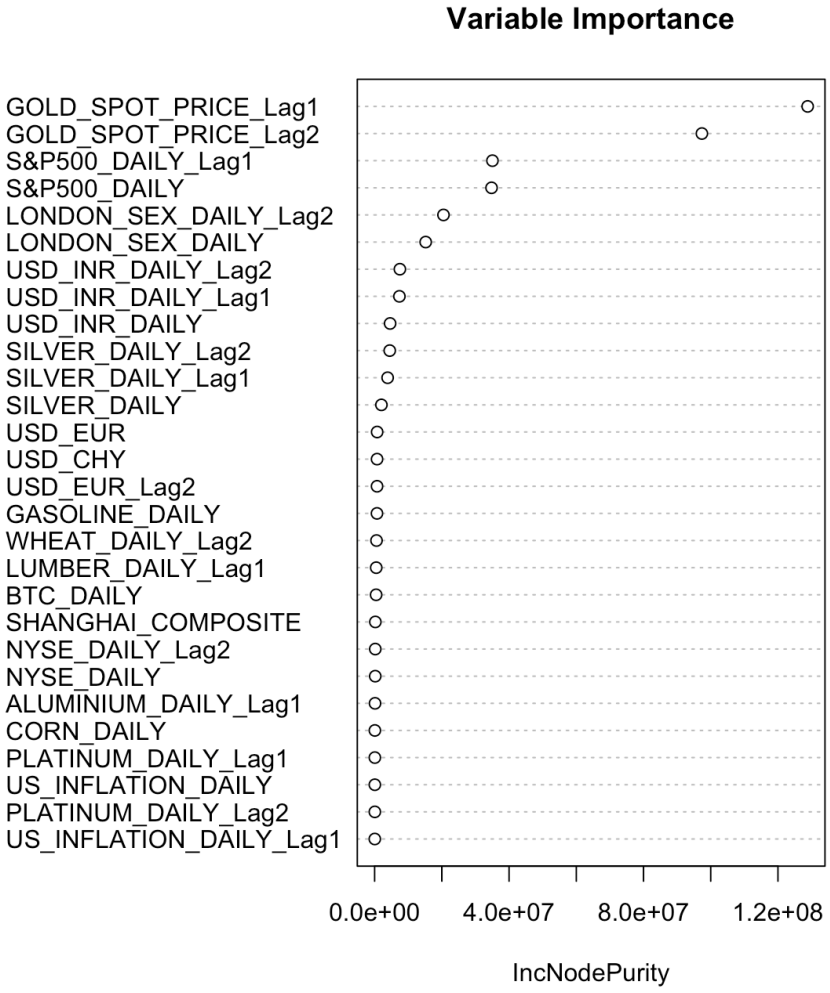
| Metric | Training | Testing |
|---|---|---|
| RMSE | 5.18 | 18.94 |
| MAE | 3.59 | 13.37 |
| | | |

Figure 14; Histogram of Residuals using Random Forests Method



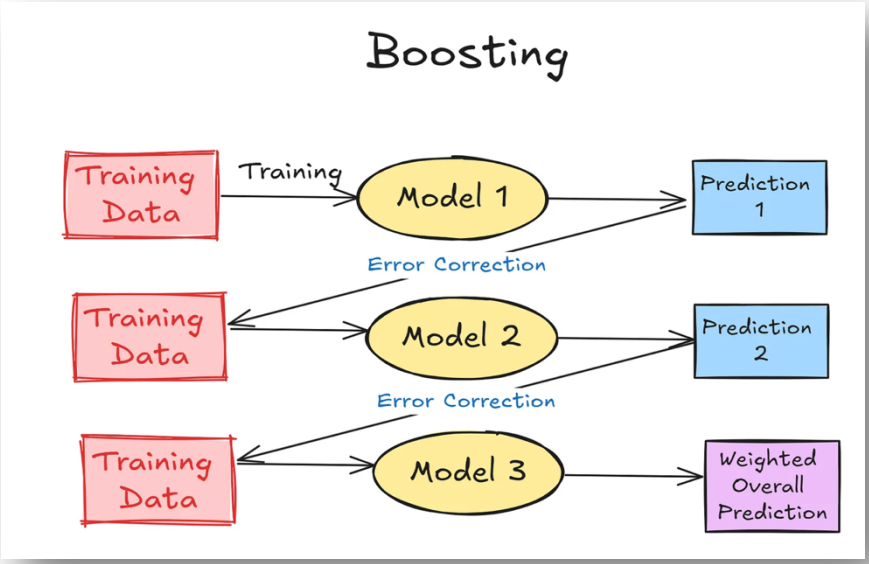

(*Figure 14; source :R Interface)*

Figure 16, displaying variable importance – how many times was variable used in decision split.




## 7.6.XGBoost Appendix

Illustration in *XGBoost Appendix* provide simple diagram of how XGBoost works.




*(Figure 17, source: Comparing Model Ensembling, Wijaya.C)*

Following screenshots, showcase different parameters algorithm was run on.

**Rolling Window**

| | roll_window | mean_test_rmse | sd_test_rmse | min_test_rmse | max_test_rmse |
|---|---|---|---|---|---|
| 1 | 100 | 12.73685 | 1.009259 | 11.37904 | 14.43882 |
| 2 | 200 | 12.73051 | 1.193057 | 11.04937 | 14.51633 |

**Tree Depth**

| | max_depth | mean_test_rmse | sd_test_rmse | min_test_rmse | max_test_rmse |
|---|---|---|---|---|---|
| 1 | 3 | 12.56797 | 1.028141 | 11.04937 | 14.12360 |
| 2 | 6 | 12.89939 | 1.151862 | 11.43042 | 14.51633 |

**Number of Rounds**

| | nrounds | mean_test_rmse | sd_test_rmse | min_test_rmse | max_test_rmse |
|---|---|---|---|---|---|
| 1 | 100 | 12.79593 | 1.036852 | 11.59542 | 14.51633 |
| 2 | 200 | 12.74971 | 1.168261 | 11.35039 | 14.46397 |
| 3 | 500 | 12.65541 | 1.150187 | 11.04937 | 14.38261 |

Figure 20 displaying too small and too large learning rate.

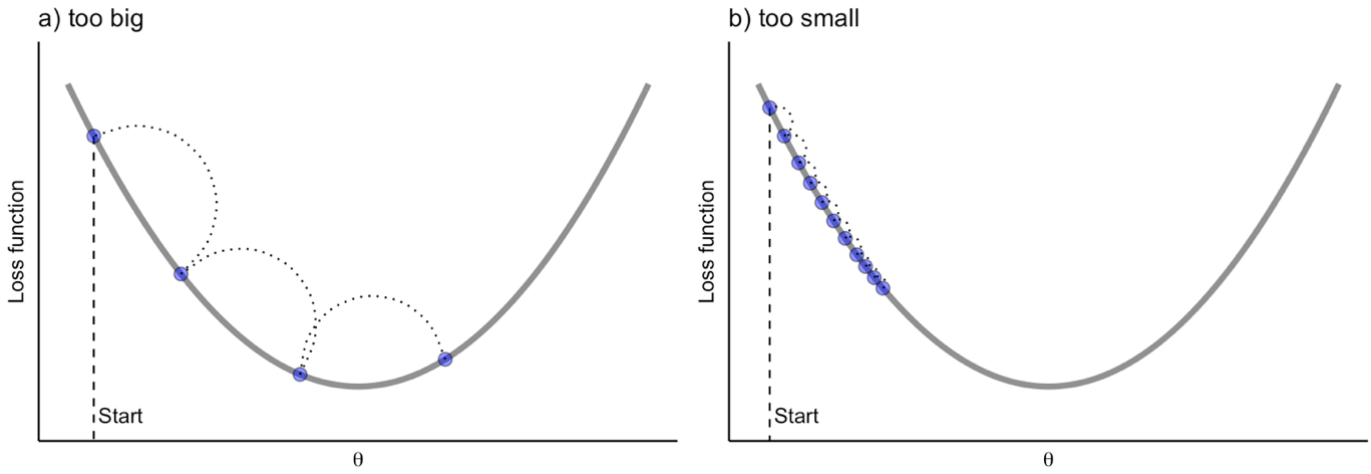
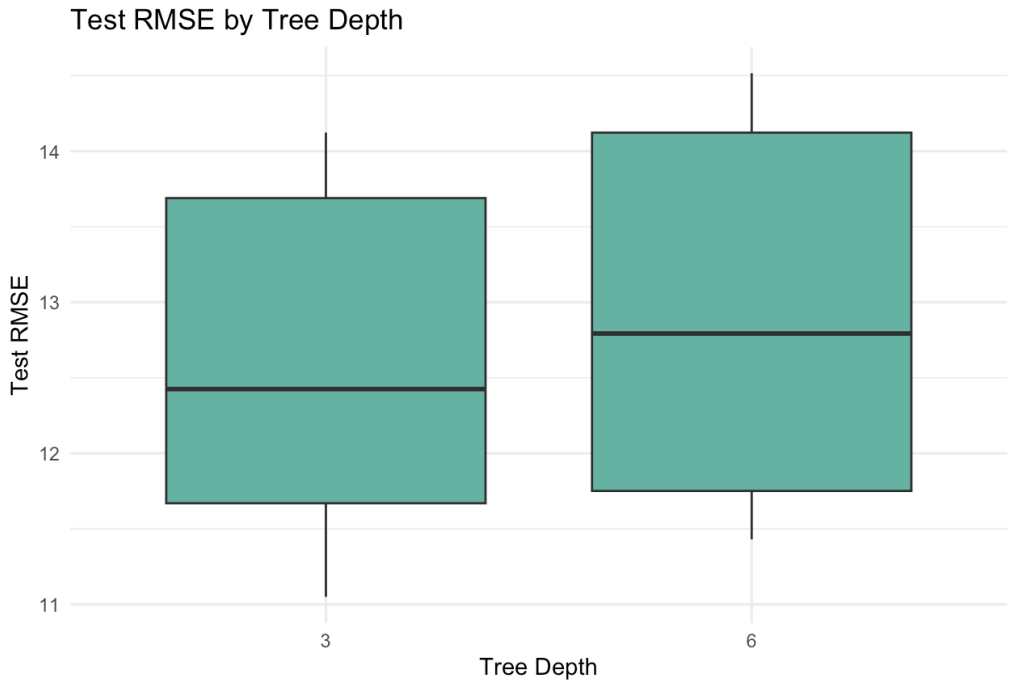



Figure 20, source ; Greenwell & Boehmke; Hands on Machine Learning)

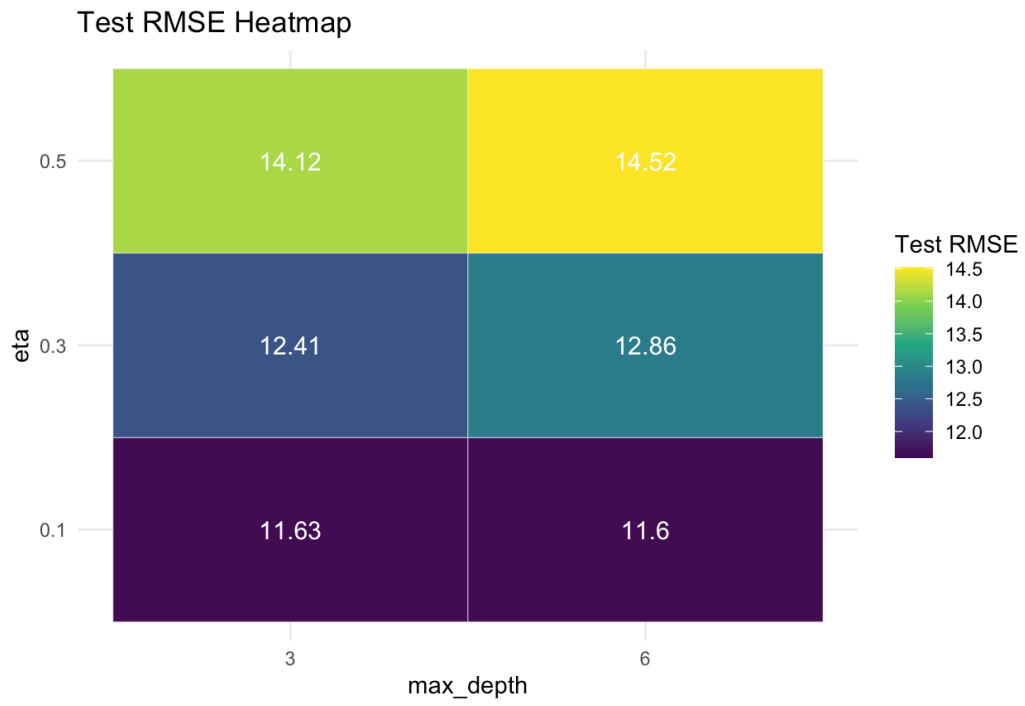
**Tree Depth**

Following plot clearly shows, that XGBoost with lower number of trees results in better RMSE

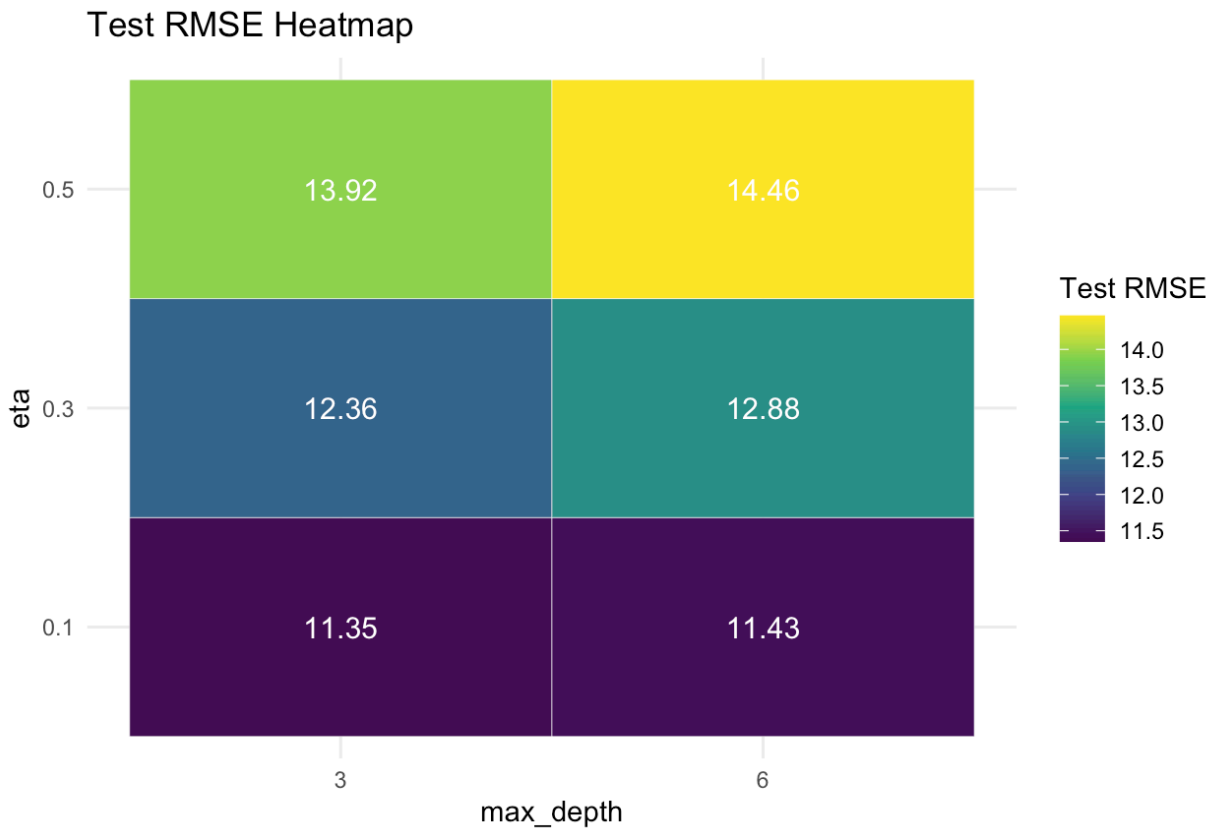



Following 3 screenshots taken in R interface, compare learning rate against tree depth, algorithm was run on different boosting rounds

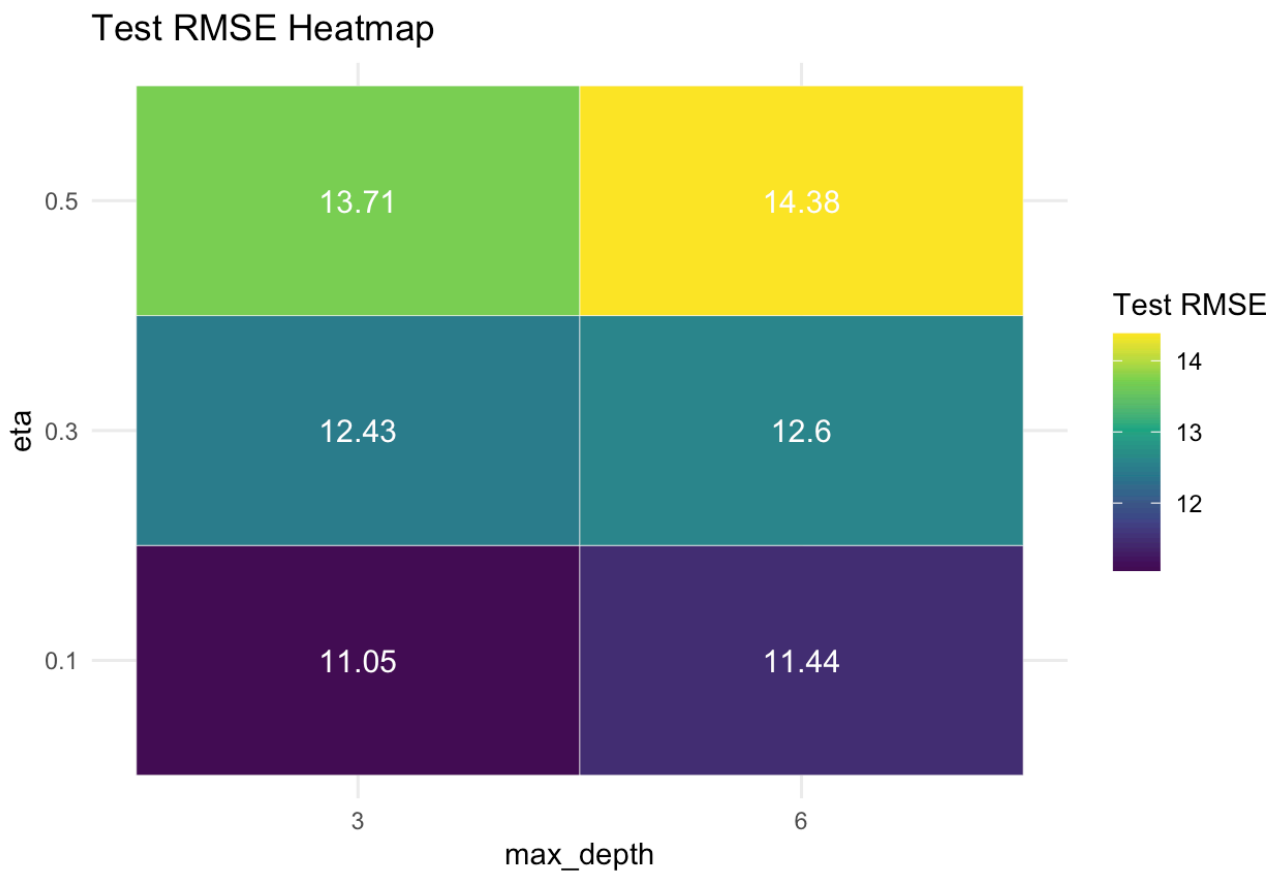
**Boosting rounds = 100**



**Boosting rounds = 200**




**Boosting rounds = 500**




Following table shows error results for all 36 possible parameter combinations

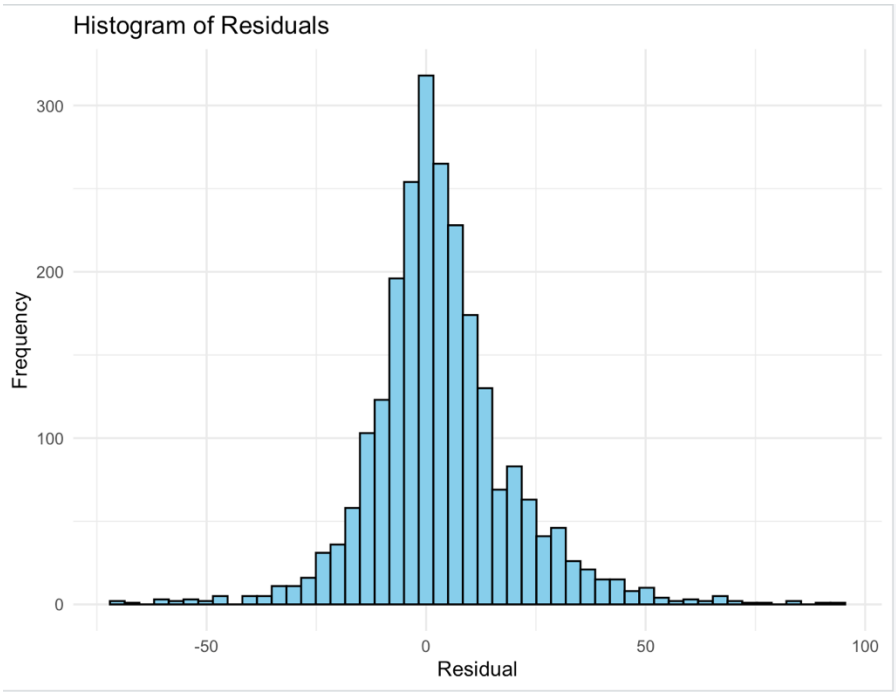
| | roll_window | eta | max_depth | nrounds | train_mae | test_mae |
|---|---|---|---|---|---|---|
| 1 | 100 | 0.1 | 3 | 100 | 2.2736253947 | 11.79629 |
| 2 | 200 | 0.1 | 3 | 100 | 4.1999907909 | 11.62697 |
| 3 | 100 | 0.3 | 3 | 100 | 0.1256374111 | 12.44145 |
| 4 | 200 | 0.3 | 3 | 100 | 0.8559079224 | 12.40673 |
| 5 | 100 | 0.5 | 3 | 100 | 0.0091153895 | 13.63848 |
| 6 | 200 | 0.5 | 3 | 100 | 0.2225380938 | 14.12360 |
| 7 | 100 | 0.1 | 6 | 100 | 0.7661324717 | 11.77768 |
| 8 | 200 | 0.1 | 6 | 100 | 1.3349266664 | 11.59542 |
| 9 | 100 | 0.3 | 6 | 100 | 0.0007495425 | 12.85023 |
| 10 | 200 | 0.3 | 6 | 100 | 0.0088513202 | 12.85699 |
| 11 | 100 | 0.5 | 6 | 100 | 0.0004123079 | 13.92095 |
| 12 | 200 | 0.5 | 6 | 100 | 0.0004480381 | 14.51633 |
| 13 | 100 | 0.1 | 3 | 200 | 0.4738053006 | 11.37904 |
| 14 | 200 | 0.1 | 3 | 200 | 1.6577083594 | 11.35039 |
| 15 | 100 | 0.3 | 3 | 200 | 0.0024042724 | 12.52678 |
| 16 | 200 | 0.3 | 3 | 200 | 0.0954837240 | 12.35840 |
| 17 | 100 | 0.5 | 3 | 200 | 0.0004209873 | 13.76960 |
| 18 | 200 | 0.5 | 3 | 200 | 0.0064820220 | 13.91931 |
| 19 | 100 | 0.1 | 6 | 200 | 0.0233080237 | 11.74203 |
| 20 | 200 | 0.1 | 6 | 200 | 0.1022998996 | 11.43042 |
| 21 | 100 | 0.3 | 6 | 200 | 0.0004325746 | 12.73584 |
| 22 | 200 | 0.3 | 6 | 200 | 0.0004589802 | 12.88191 |
| 23 | 100 | 0.5 | 6 | 200 | 0.0003885165 | 14.43882 |
| 24 | 200 | 0.5 | 6 | 200 | 0.0004067752 | 14.46397 |
| 25 | 100 | 0.1 | 3 | 500 | 0.0094212501 | 11.41209 |
| 26 | 200 | 0.1 | 3 | 500 | 0.1759145652 | 11.04937 |
| 27 | 100 | 0.3 | 3 | 500 | 0.0004283505 | 12.41754 |
| 28 | 200 | 0.3 | 3 | 500 | 0.0004981608 | 12.43289 |
| 29 | 100 | 0.5 | 3 | 500 | 0.0003854070 | 13.86806 |
| 30 | 200 | 0.5 | 3 | 500 | 0.0004089991 | 13.70652 |
| 31 | 100 | 0.1 | 6 | 500 | 0.0005406645 | 11.68685 |
| 32 | 200 | 0.1 | 6 | 500 | 0.0005653016 | 11.44484 |
| 33 | 100 | 0.3 | 6 | 500 | 0.0004062899 | 12.67138 |
| 34 | 200 | 0.3 | 6 | 500 | 0.0004228903 | 12.60251 |
| 35 | 100 | 0.5 | 6 | 500 | 0.0003710870 | 14.19024 |
| 36 | 200 | 0.5 | 6 | 500 | 0.0003849695 | 14.38261 |

Results of XGBoost after running it on chosen parameter values.

```
> mean(results$train_rmse)
[1] 4.197031
> mean(results$test_rmse)
[1] 11.53553
> mean(results$train_mae)
[1] 3.255684
> mean(results$test_mae)
[1] 11.53553
>
```

| Error Metric | Training | Testing |
|---|---|---|
| RMSE | 4.19 | 16.68 |
| MAE | 3.26 | 11.53 |

Figure 22 : Histogram of Residuals Using XGBoost




*(Figure 22; source : R Interface)*

# 8. Bibliography

Following sources were used to obtain the data, that were used for predictive analysis.


NYSE historical daily price : https://www.nasdaq.com/market-activity/index/nya/historical?page=1&rows_per_page=10&timeline=y10

Gold spot prices: https://www.gold.org/goldhub/data/gold-prices#from-login=1&just-verified=1

S&P 500 data: https://www.nasdaq.com/market-activity/index/spx/historical

USD/EUR data: https://www.wsj.com/market-data/quotes/fx/USDEUR/historical-prices

Breakeven inflation rate US (explain):https://fred.stlouisfed.org/series/T10YIE :

US/YUAN exchange rate: https://fred.stlouisfed.org/series/DEXCHUS#0

London stock exchange daily: https://www.investing.com/equities/london-stock-exchange-historical-data

Shanghai Composite index daily prices https://www.investing.com/indices/shanghai-composite-news

Lumber daily prices: https://www.macrotrends.net/2637/lumber-prices-historical-chart-data#google_vignette

Wheat daily prices: https://www.macrotrends.net/2534/wheat-prices-historical-chart-data

OIL and Gasoline prices: http://www.jodidata.org/oil/database/data-downloads.aspx

Silver daily prices: https://www.investing.com/commodities/silver-historical-data

Aluminum daily prices: https://www.investing.com/commodities/aluminum-historical-data

Copper daily prices: https://www.macrotrends.net/1476/copper-prices-historical-chart-data#google_vignette

Platinum daily prices: https://www.macrotrends.net/2540/platinum-prices-historical-chart-data#google_vignette

US 10 Year Bond Yield: https://fred.stlouisfed.org/series/DFII10


Following sources, were used to obtain further knowledge about the specific method, its tuning or pros and cons.


Wikipedia. 2025. "*Shanghai Stock Exchange".* Wikipedia, [Online], accessed January 24, 2025, https://en.wikipedia.org/wiki/Shanghai_Stock_Exchange

Wikipedia. 2025. "*London Stock Exchange".* Wikipedia, [Online], accessed January 24, 2025, https://en.wikipedia.org/wiki/London_Stock_Exchange

Finance Magnates. 2024. *Gold and Silver : A Correlated Duo*. [Online]. Finance Magnates, accessed January 31,2025, https://www.financemagnates.com/thought-leadership/gold-and-silver-a-correlated-duo/

GoldPriceForecast. *Gold to Platinum Ratio.* [Online]. GoldPriceForecast, accessed January 31, 2025, https://www.goldpriceforecast.com/explanations/gold-platinum-ratio/

Morningstar. 2024. *Will inflation stick around ? What oil, gold and stocks are signaling abuout rising prices.* [Online]. Morningstar, accessed January 31, 2025, https://www.morningstar.com/news/marketwatch/20241124170/will-inflation-stick-around-what-oil-gold-and-stocks-are-signaling-about-rising-prices?utm

World Gold Council. 2023. *Gold miners' costs reached a record high in 2022 but dropped in the final quarter of the year.* [Online]. World Gold Council, accessed January 31, 2025, https://www.gold.org/goldhub/gold-focus/2023/04/gold-miners-costs-reached-record-high-2022-dropped-final-quarter-year

CME Group. 2022. *Crop Prices and Inflation: What is the relationship ?.* [Online]. CME Group, accessed February 1, 2025, https://www.cmegroup.com/insights/economic-research/2022/crop-prices-and-inflation-what-is-the-relationship.html

FFR Trading. 2023. *The Intricate Connection Between Lumber, Gold and the Stock Market : Understanding Their Relationship and Predictive Power.* [Online]. FFR Trading, accessed February 1, 2025, https://www.ffrtrading.com/the-intricate-connection-between-lumber-gold-and-the-stock-market-understanding-their-relationship-and-predictive-power/

Michele Patane, Mattia Tadesco and Setfano Zedda. 2017. "*Dynamic Relationship of Commodities Prices and USD/EUR Exchange Rate Trends in Recent Past".* Scientific Research, https://www.scirp.org/journal/paperinformation?paperid=78165

Statista. 2023. *Consumption of gold worldwisen 2023, by selected country.* [Online]. Statista, accessed February 1, 2025, https://www.statista.com/statistics/299638/gold-consumer-demand-by-top-consuming-country/

Investopedia. 2025. *Has Gold Been a Good Investment Over the Long Term ?.* [Online]. Investopedia, accessed February 3, 2025, https://www.investopedia.com/ask/answers/020915/has-gold-been-good-investment-over-long-term.asp

Investopedia. 2025. *Understanding Dynamics Behind Gold Prices.* [Online]. Investopedia, accessed February 4, 2025, https://www.investopedia.com/financial-edge/0311/what-drives-the-price-of-gold.aspx?utm_source=chatgpt.com

Investopedia, 2025. *Bond Yield: What It Is, Why It Matters and How It's Calculated ?.* [Online], accessed 5 February 5, 2025, https://www.investopedia.com/terms/b/bond-yield.asp

Economic Times. 2015. *Top eight reasons why gold prices are falling.* [Online]. Economic Times, accessed February 5, 2025, https://economictimes.indiatimes.com/markets/commodities/top-eight-reasons-why-gold-

prices-are-falling/articleshow/48384920.cms?utm_source=chatgpt.com&from=mdr#goog_rewarded

Chen, Wei-Jie & Yao, Jing-Jing & Shao, Yuanhai. 2022. "*Volatility forecasting using deep neural network with time-series feature embedding".* Economic Research-Ekonomska Istraživanja, https://www.tandfonline.com/doi/full/10.1080/1331677X.2022.2089192

Mittal, Neetu, and Ashwani Kumar. 2022."*Analysis of Supervised Feature Selection in Bioinformatics.*" Science Direct, https://www.sciencedirect.com/science/article/pii/B9780323906159000086

Hyndman, R.J & Anthanasopoulos, G. 2021. *Forecasting : Principles and Practice (3rd edition), https://otexts.com/fpp3/*

Shruti Dhumme. 2023. "*Elastic Net Regression detailed guide!".* [Online]. Medium, accessed February 10, 2025, https://medium.com/@shruti.dhumne/elastic-net-regression-detailed-guide-99dce30b8e6e

Chamlal ,H. , Benzmane, A. & Oudarerhman. 2024. "Elastic net-based high dimensional data selection for regression. Science Direct. [Online], accessed February 10, 2025, https://www.sciencedirect.com/science/article/pii/S0957417423034607

Jain, A. 2024. "*Elastic Net Regression – Combined Features of L1 and L2 Regularization.* Medium. [Online], accessed February 10, 2025, https://medium.com/@abhishekjainindore24/elastic-net-regression-combined-features-of-l1-and-l2-regularization-6181a660c3a5

Negi, J. 2020. "*In layman's terms, what is lasso and ridge regression?".* Medium. [Online], accessed February 15,2025, https://medium.com/analytics-vidhya/in-laymans-terms-what-is-lasso-and-ridge-regression-54f9d6827b13

Housainy, L., Amal, M. and Haitham, F. "*Time Series Forecasting Using Tree Based Methods".* Research Gate. [Online], accessed February 20, 2025, https://www.researchgate.net/publication/349710965_Time_Series_Forecasting_Using_Tree_Based_Methods

Diettreich, T.G. "*Ensemble methods in Machine Learning".* Oregon State. [Online], accessed February 21, 2025. https://web.engr.oregonstate.edu/~tgd/publications/mcs-ensembles.pdf

Alam, M. 2024. "*Understanding Random Forests Algorithm – A Comprehensive Guide".* Data Science Dojo. [Online}, accessed Febraury 21, 2025, https://datasciencedojo.com/blog/random-forest-algorithm/

Alex Molas. 2022. "*Can Random Forests Overfit?".* Medium. [Online], accessed February 21, 2025. https://medium.com/@alexmolasmartin/can-random-forests-overfit-a743755251b4

Probst, P., Wright, M. and Boulesteix, A. 2019. „*Hyperparameters and Tuning Strategies for Random Forests".* Arxiv. [Online], accessed February 22, 2025, https://arxiv.org/pdf/1804.03515

Soni, B. 2023. „*Understanding Boosting in Machine Learning : A Comprehensive Guide“.* Medium. [Online], accessed March 1, 2025 https://medium.com/@brijesh_soni/understanding-boosting-in-machine-learning-a-comprehensive-guide-bdeaa1167a6

Chen, T. and Guestrin, C. 2016. “*XGboost : A Scalable Tree Boosting System”.* Arxiv. [Online], accessed March 3, 2025 https://arxiv.org/abs/1603.02754

Nielsen, D. 2016. “*Tree Boosting with XGBoost”.* NTNU. [Online], accessed March 5, 2025, https://pzs.dstu.dp.ua/DataMining/boosting/bibl/Didrik.pdf

Wikipedia. 2025. “*Autoregressive Integrated Moving Average”.* Wikipedia, [Online], accessed March 10, 2025, https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average

*Stefan Gudmunsson. AU Lecture Presentation – Linear Regression.*