Final Project - R programming - Prof. Kang Lele

Do Minh Ngoc - 502024145013

2025-07-18

1. Final Project Description

1.1. Introduction

Ethereum, launched in 2015, is a decentralized blockchain enabling smart contracts and DApps with its native currency, Ether (ETH). It has become the most popular blockchain, driving decentralized finance (DeFi) and non-fungible tokens (NFTs), with ETH ranking second in market cap. Traditional models like ARIMA focus on linear patterns and Bitcoin, while AI models such as ANN fail to incorporate economic and technological factors impacting exchange rates. This study combines economic and technological factors with LSTM networks to predict Ethereum exchange rates more effectively.

1.2. Research Question

- Can economic and technological factors improve Ethereum exchange rate prediction compared to historical data alone?
- Which factors significantly affect Ethereum exchange rates?
- Which machine learning model (LSTM, SVR, ARIMA) is most accurate?

1.3. Research Method

A two-stage approach is used: Random Forest and ANN for feature selection. Phase 2: LSTM (selected features) for prediction, compared to traditional models (SVR, ARIMA, LSTM (old price))).

Import Essential Packages

```
library('tidyr')
library('ggplot2')
library('stringr')
library('skimr')
library('readr')
library('dplyr')
library('zoo')
library('caret')
library(randomForest)
library(nnet)
library(forecast)
library(e1071)
library(gridExtra)
```

2. The data

Data Preparation

Table 1. List of economic and technological factors that can affect ETH price

Description	Previous Research
Economic Factor	
- Crude oil price	Gospodinov & Jamali, 2015
- Gold price	Gospodinov & Jamali, 2015
- NASDAQ	Gospodinov & Jamali, 2015
- Global currency ratio (USD)	Dyhrberg, 2015; Jang & Lee, 2018; Kristjanpoller & Minutolo, 2018
- EUR/USD, JPY/USD, CNY/USD	Dyhrberg, 2015; Jang & Lee, 2018; Kristjanpoller & Minutolo, 2018
Technology Factor	
- gas limit	Gospodinov & Jamali, 2015
- Bitcoin price	Saad et al. (2019)
- Blocksize	W. Chen, H. Xu, L. Jia et al (2021)
- Blocktime	W. Chen, H. Xu, L. Jia et al (2021)
- Transaction Fee	W. Chen, H. Xu, L. Jia et al (2021)
- Transaction volume	W. Chen, H. Xu, L. Jia et al (2021)
- Transaction Value	W. Chen, H. Xu, L. Jia et al (2021)
- Market capitalization	W. Chen, H. Xu, L. Jia et al (2021)
- Public Attention (Volume of Tweets on Twitter)	Dastgir, Demir, Downing, Gozgor, & Chi, 2018; Polasik et al., 2015; Zhang, Wang, Li, & Shen, 2018
Source	Synthesis of previous studies

2.1. Technical and Blockchain Data (bitinfocharts.com)

Data from bitinfocharts.com and ethereumscan.io provides detailed historical data on various aspects of the Ethereum blockchain, including:

- **Price of Ethereum**: The ETH/USD price shows how much one Ethereum (ETH) is worth in US dollars.
- **Ethereum Mention Tweets**: Measures how often Ethereum is mentioned on Twitter, showing the number of tweets containing Ethereum-related keywords.
- **Transaction Value**: Total value of transactions processed on the Ethereum network, expressed in USD or ETH.
- **Number of Transactions**: The total number of transactions recorded on the blockchain during a given period.
- **Average Transaction Fee**: The average fee paid for each transaction.
- **Block Size**: The average size of blocks added to the blockchain.
- **Block Time**: The average time it takes to create a new block.
- **Market Cap**: Ethereum's market capitalization.
- **Gas Limit** refers to the maximum amount of computational work or resources that can be used for a transaction or contract execution on the Ethereum network. It determines how much "gas" (a measure of computational effort) is allowed for a transaction or smart contract. A higher gas limit allows for more complex transactions or contracts but also increases transaction costs.
- **BTC Price** represents the market value of Bitcoin (BTC), the first and most well-known cryptocurrency. It is typically quoted in terms of USD or other fiat currencies and serves as a key indicator of

Bitcoin's value in the broader cryptocurrency market. The BTC price fluctuates based on demand, market sentiment, and external economic factors.

These metrics help assess the health and performance of the Ethereum network, which directly impacts the value of Ether (ETH).

2.2. Economic Data (investing.com)

Data from Investing.com includes global market data such as:

- **Exchange Rates**: Data on exchange rates between various currencies (e.g., CNY/USD, EUR/USD, JPY/USD).
- **Gold Price**: The international gold price.
- **Oil Price**: Crude oil prices.
- **Stock Indices**: Stock indices such as the Nasdaq.

These data points reflect macroeconomic conditions and traditional market factors that influence investor sentiment, which in turn impacts the value of digital assets like Ether.

Note : Since the data comes from different sources, some use commas (,) to separate decimal numbers, while others use periods (.). This is why the following code is used.

```
#Read csv sep by ;
data <- read_delim(
  "Dataset.csv",
  delim = ";",
  locale = locale(decimal_mark = ",", grouping_mark = ".")
)

data2 <- read_delim(
  "Dataset.csv",
  delim = ";",
  locale = locale(decimal_mark = ".", grouping_mark = ",")
)

data$btc_price <- data2$btc_price

data <- data[-1]

#Overview of data
skim_without_charts(data)
```

Table 2: Data summary

Name	data
Number of rows	2555
Number of columns	17
Column type frequency:	
character	8
numeric	9

Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
date	0	1	6	8	0	2555	0
tweets	0	1	3	6	0	2372	0
transaction_fee	0	1	3	7	0	1883	0
transaction_value	0	1	3	7	0	2442	0
number_of_transaction	0	1	4	7	0	2543	0
blocksize	0	1	4	6	0	2356	0
blocktime	0	1	3	5	0	154	0
marketcap	0	1	4	11	0	2078	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
price	0	1	895.99	1115.60	6.82	154.01	311.44	1363.00	4788.00
gas_limit	0	1	13263193.89	9561569.50	500238.00	7995029.00	8017619.00	14986614.00	30076835.00
cny_usd	0	1	0.15	0.01	0.14	0.14	0.15	0.15	0.16
eur_usd	0	1	1.13	0.06	0.96	1.10	1.13	1.18	1.25
jpy_usd	0	1	0.89	0.07	0.67	0.88	0.91	0.93	1.00
gold	0	1	1527.31	264.37	1129.80	1279.35	1477.00	1793.50	2054.60
Oil	0	1	60.93	17.87	11.57	48.62	57.59	70.37	119.78
Nasdaq	0	1	9248.86	3166.80	4594.44	6792.00	8098.38	11728.67	16057.44
btc_price	0	1	15800.97	16450.13	409.55	4033.90	9143.58	21388.09	67566.83

The data has many missing values and some columns are mixed between character and numeric types, so it needs to be cleaned.

2.3. Data Cleaning

```
data_clean <- na.omit(data)
data_clean <- as.data.frame(lapply(data_clean, function(x) ifelse(x == "null", NA, x)))
cols_need_converted <- c("tweets","transaction_fee","transaction_value","number_of_transaction","blocksi
data_clean[cols_need_converted] <- lapply(data_clean[cols_need_converted], function(x) gsub(",", ".", x)
data_clean[cols_need_converted] <- lapply(data_clean[cols_need_converted], as.numeric)
data_clean$date <- as.Date(data_clean$date, format = "%d/%m/%y")

#Forward & Backward fill to fill time series missing value
# Forward fill,then, backward fill
forward_backward_fill <- function(data_clean, col_name) {
  # Forward fill
  data_clean[[col_name]] <- na.locf(data_clean[[col_name]], na.rm = FALSE)

  # Backward fill
  data_clean[[col_name]] <- na.locf(data_clean[[col_name]], fromLast = TRUE, na.rm = FALSE)
```

```
  return(data_clean)
}

for (i in 1:8) {
  data_clean <- forward_backward_fill(data_clean, colnames(data_clean)[i])
}

skim_without_charts(data_clean)
```

Table 5: Data summary

Name	data_clean
Number of rows	2555
Number of columns	17
_____	
Column type frequency:	
Date	1
numeric	16
_____	
Group variables	None

Variable type: Date

skim_variable	n_missing	complete_rate	min	max	median	n_unique
date	0	1	2016-03-16	2023-03-14	2019-09-14	2555

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
tweets	0	1	1.529260e+04	6.406557e+04	543.00	4.207000e+03	8.385300e+03	2.421650e+04	1.482200e+05
transaction_fee	0	1	4.960000e+00	1.078000e+01	0.00	1.000000e-01	3.600000e-01	3.690000e+00	1.966800e+02
transaction_value	0	1	3.201160e+04	2.347370e+05	84.83	5.471500e+03	1.334000e+04	3.419000e+04	3.328000e+06
number_of_transaction	0	1	7.438388e+04	2.233614e+05	275.00	4.808800e+03	7.581330e+03	1.524004e+05	1.632711e+06
blocksize	0	1	3.545046e+04	2.482555e+04	1215.00	1.529150e+04	3.445900e+04	5.498300e+04	1.274680e+05
blocktime	0	1	2.400000e-01	4.000000e-02	0.20	2.200000e-01	2.300000e-01	2.400000e-01	5.100000e-01
marketcap	0	1	1.038289e+11	3.332633e+10	58122873300	1.062769e+11	1.002224e+11	1.165000e+11	1.660000e+11
price	0	1	8.959900e+01	1.215600e+03	6.82	1.540100e+00	1.214400e+01	1.263000e+02	3.788000e+03
gas_limit	0	1	1.326319e+07	6.761570e+06	238.00	7.995029e+06	1.617619e+07	1.698661e+07	3.707684e+07
cny_usd	0	1	1.500000e-01	1.000000e-02	0.14	1.400000e-01	1.500000e-01	1.500000e-01	1.600000e-01
eur_usd	0	1	1.130000e+00	6.000000e-02	0.96	1.100000e+00	1.130000e+00	1.180000e+00	1.250000e+00
jpy_usd	0	1	8.900000e-01	7.000000e-02	0.67	8.800000e-01	9.100000e-01	9.300000e-01	1.000000e+00
gold	0	1	1.527310e+03	2.643700e+02	1229.80	1.279350e+03	1.477000e+03	1.793500e+03	2.054600e+03
Oil	0	1	6.093000e+01	2.787000e+01	11.57	4.862000e+01	5.759000e+01	7.037000e+01	1.197800e+02
Nasdaq	0	1	9.248860e+03	3.866800e+03	4594.44	6.792000e+03	8.698380e+03	1.372867e+04	1.605744e+04

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
btc_price	0	1	1.580097e+04	2.445013e+04	409.55	4.033900e+03	9.343580e+03	2.038809e+04	6.756683e+04

2.4. Data Normalization

For the model to work well, the input data must be processed to prevent variables with larger values from dominating those with smaller values, as recommended by Rezakazemi et al. (2013). In other words, scaling the variables ensures that they have equal importance in the model training process.

This study used the **Min-Max normalization** method to "balance" the input variables. The Min-Max normalization formula is:

$$x_{\text{subscript i}} = \frac{(x_i - \min(X))}{(\max(X) - \min(X))}$$

This formula "shrinks" all values of a variable to the range of 0 to 1. By doing so, variables with different units of measurement and different magnitudes are treated equally when entered into the model.

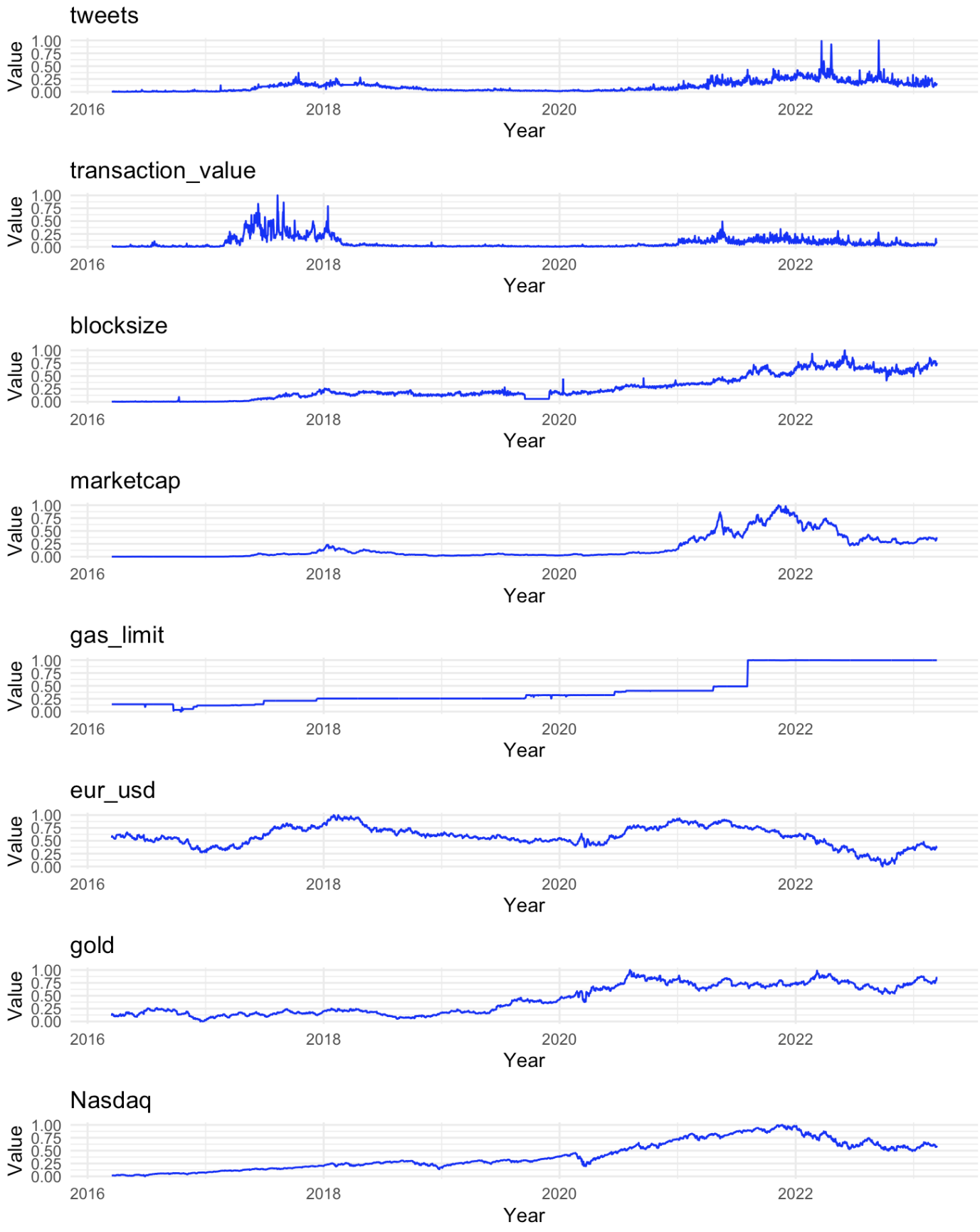
```
# Create normalize function Min-Max
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}

# Apply for all columns
columns_to_scale <- c('tweets', 'transaction_fee', 'transaction_value',
                      'number_of_transaction', 'blocksize', 'blocktime',
                      'marketcap', 'gas_limit', 'cny_usd', 'eur_usd',
                      'jpy_usd', 'gold', 'Oil', 'Nasdaq','btc_price')

scaled_data <- data_clean
scaled_data[columns_to_scale] <- lapply(scaled_data[columns_to_scale], normalize)

# Check the result
head(scaled_data)
```

```
##         date      tweets transaction_fee transaction_value
## 1 2016-03-16 0.004663088    3.462505e-05       0.011474704
## 2 2016-03-17 0.004851936    3.259127e-05       0.013232803
## 3 2016-03-18 0.004212759    2.140550e-05       0.017116470
## 4 2016-03-19 0.002905351    4.530238e-05       0.003977174
## 5 2016-03-20 0.005527430    3.411661e-05       0.002305871
## 6 2016-03-21 0.012275108    3.259127e-05       0.005515297
##   number_of_transaction   blocksize blocktime    marketcap  price gas_limit
## 1           0.000000000 0.000799981 0.1233766 0.0008013912 13.233 0.1424467
## 2           0.003115298 0.001631644 0.1168831 0.0005766796 11.603 0.1424766
## 3           0.002217714 0.001489074 0.1331169 0.0003843265 10.208 0.1424409
## 4           0.003770376 0.002209848 0.1168831 0.0004348066 10.568 0.1424454
## 5           0.002546303 0.001829659 0.1233766 0.0003885456 10.230 0.1424466
## 6           0.002579897 0.002241531 0.1266234 0.0004925616 10.977 0.1424491
##     cny_usd   eur_usd   jpy_usd      gold       Oil     Nasdaq    btc_price
## 1 0.7256637 0.5589849 0.6626866 0.1081315 0.2484983 0.01478932 1.111269e-04
## 2 0.7787611 0.5915638 0.6916418 0.1461938 0.2645781 0.01574980 1.648816e-04
```

```
## 3 0.7787611 0.5744170 0.6868657 0.1346237 0.2575548 0.01755300 0.000000e+00
## 4 0.7787611 0.5744170 0.6868657 0.1346237 0.2575548 0.01755300 1.334176e-05
## 5 0.7787611 0.5744170 0.6868657 0.1346237 0.2575548 0.01755300 6.264400e-05
## 6 0.7654867 0.5648148 0.6779104 0.1237024 0.2618982 0.01870627 5.597312e-05
```

2.5. Visualization normalized data

```
# List column
columns <- c("tweets", "transaction_fee", "transaction_value", "number_of_transaction",
             "blocksize", "blocktime", "marketcap", "price", "gas_limit", "cny_usd",
             "eur_usd", "jpy_usd", "gold", "Oil", "Nasdaq", "btc_price")

# List chart
plots <- list()

# Create chart for each variable
plots <- list()

for (col in columns) {
  p <- ggplot(scaled_data, aes(x = date, y = !!sym(col))) +
    geom_line(color = "blue") +
    labs(title = col, x = "Year", y = "Value") +
    ylim(0, 1) +
    theme_minimal()
  plots[[col]] <- p
}

# Arrange chart
grid.arrange(
  # 8 first charts
  plots[[1]], plots[[2]], plots[[3]], plots[[4]], plots[[5]], plots[[6]], plots[[7]], plots[[8]],
  # 7 remaining chart
  plots[[9]], plots[[10]], plots[[11]], plots[[12]], plots[[13]], plots[[14]], plots[[15]],
  ncol = 2
)
```


tweets
transaction_value
blocksize
marketcap
gas_limit
eur_usd
gold
Nasdaq
Value
1.00
0.75
0.50
0.25
0.00
2016
2018
2020
2022
Year

3. Modeling Stage 1: Feature Selection

Modeling Stage 1: Feature Selection using Random Forest and ANN

In Stage I, I focused on **feature selection** to identify the most predictive economic and technological factors, removing redundant ones. This stage is broken into three sub-steps:

1. Feature Selection Models I used two machine learning models, **Random Forest (RF)** and **Artificial Neural Networks (ANN)**, both known for their strong feature-filtering capabilities (Enke & Thawornwong, 2005; Tsai & Hsiao, 2011). These models help identify which features contribute most to predicting Ethereum exchange rates.

2.Factor Importance Assessment **Sensitivity analysis** was used to evaluate the importance of each factor (Dag et al., 2016). The importance values of the features were then normalized to a scale of 0 to 1.

3.Combining Feature Selection Methods As suggested by Tsai and Hsiao (2011), combining multiple feature selection techniques improves prediction performance. I employed a "crossover strategy" to filter out unimportant variables, resulting in a final set of candidate features for Stage II of modeling. The **caret package** in R was used to implement the RF models (Kuhn, 2015).

3.1. Random Forest (RF)

Random Forest is an ensemble learning algorithm that builds multiple decision trees using bootstrap and random node-splitting techniques. Predictions are made by aggregating the "votes" from individual trees. RF is robust against noisy and missing data and can handle complex interactions between categorical variables (Breiman, 2001; Svetnik et al., 2003).

Factor Importance in RF The importance of each factor is calculated using the **Out-Of-Bag Error (OOBE)** method. Two error measures are used to evaluate importance:

1. **OOBE1**: The initial error calculated using the data not used in tree construction.
2. **OOBE2**: The error after adding noise to the factor of interest in the OOB data.

Importance Measures

- **%IncMSE (Percentage Increase in Mean Squared Error)**: Measures how much the model's error increases when a variable is removed. A higher %IncMSE indicates greater importance.
- **IncNodePurity (Increase in Node Purity)**: Measures how much a variable reduces node impurity. The higher the IncNodePurity, the more important the variable.

These measures guide the feature selection process, ensuring the most relevant variables are retained for further analysis.

```
#Split into train and test dataset
set.seed(123)
train_index <- createDataPartition(scaled_data$price, p = 0.8, list = FALSE)
train_data <- scaled_data[train_index, ]
test_data <- scaled_data[-train_index, ]
train_data <- train_data[, !names(train_data) %in% "date"]
```

```r
test_data <- test_data[, !names(test_data) %in% "date"]


#Check the size of train and test dataset
dim(train_data)
```

```
## [1] 2047   16
```

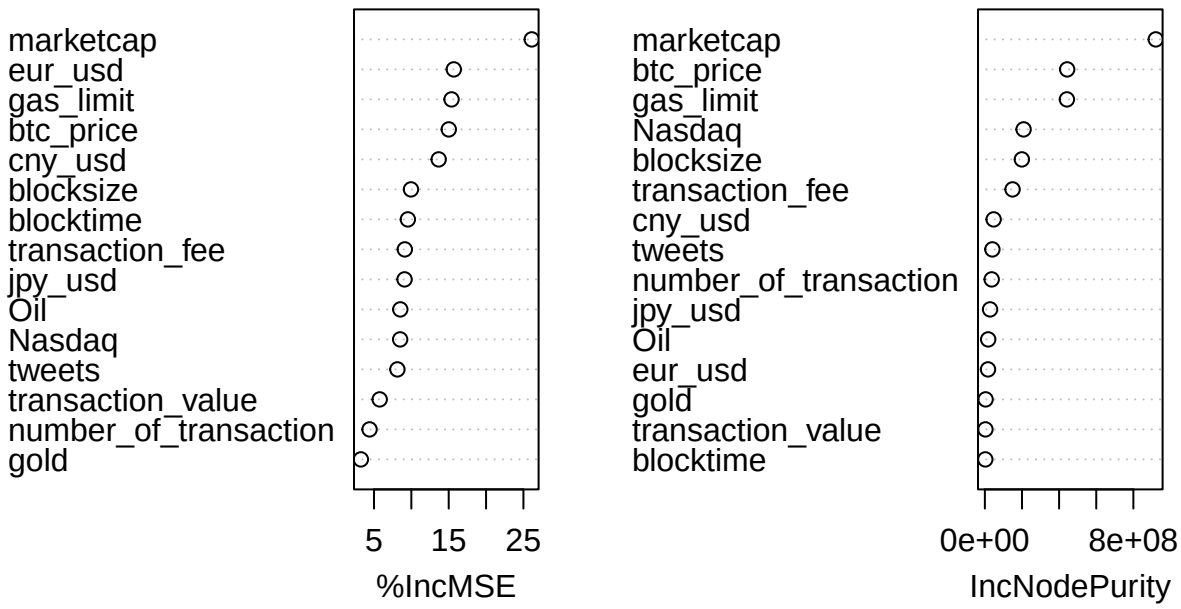
```r
dim(test_data)
```

```
## [1] 508  16
```

```r
# Feature selection with Random Forest trên train_data
rf_model <- randomForest(price ~ ., data = train_data, importance = TRUE)

# Print important features
print(rf_model)
```

```
##
## Call:
##  randomForest(formula = price ~ ., data = train_data, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 1020.737
##                     % Var explained: 99.92
```

```r
varImpPlot(rf_model)
```



3.2. ANN

Artificial Neural Network (ANN) and Feature Importance Evaluation

Artificial Neural Networks (ANN) are computational models inspired by the human brain. They consist of layers of interconnected nodes (neurons) that work together to solve complex problems. ANN models can learn intricate patterns and relationships in data, making them powerful for predictive tasks. The model used in this study is a feedforward network with 10 hidden nodes and a linear output, trained on the `train_data` dataset.

Model Training and Prediction The following code builds and trains an ANN model using the `nnet` package:

```
# Build model ANN
ann_model <- nnet(price ~ ., data = train_data, size = 10, linout = TRUE, trace = FALSE)

# Predict
ann_predictions <- predict(ann_model, test_data)

mse_full <- mean((ann_predictions - test_data$price)^2)
cat("MSE of full model: ", mse_full, "\n")
```

```
## MSE of full model:  359092.1
```

Here, the model is trained to predict the price variable using all available features. The Mean Squared Error (MSE) is calculated to assess the model's performance on the test dataset.

Feature Importance Evaluation To assess the importance of each feature in the model, we evaluate the MSE when a feature is removed from the model. A large increase in MSE indicates that the feature is important for accurate predictions.

The following function calculates the MSE when a feature is excluded:

```
mse_full <- mean((ann_predictions - test_data$price)^2)
cat("MSE of full model: ", mse_full, "\n")
```

```
## MSE of full model:  359092.1
```

```
# Function to re-calculate MSE when remove one feature
calculate_mse <- function(exclude_col) {
  # Remove one feature from train and test data set repetitively
  train_data_mod <- train_data[, !names(train_data) %in% exclude_col]
  test_data_mod <- test_data[, !names(test_data) %in% exclude_col]

  # Train model
  ann_model_mod <- nnet(price ~ ., data = train_data_mod, size = 10, linout = TRUE, trace = FALSE)

  # Predict
  ann_predictions_mod <- predict(ann_model_mod, test_data_mod)

  # Calculate MSE for changed model
  mse_mod <- mean((ann_predictions_mod - test_data_mod$price)^2)
  return(mse_mod)
}

# Apply function to calculate MSE when remove feature
features <- setdiff(names(train_data), "price") #Remove price - dependant variable from features list
mse_results <- sapply(features, calculate_mse)

# Calculate delta mse
delta_mse <- mse_results - mse_full
cat("Delta MSE for each feature removal: \n")
```

```
## Delta MSE for each feature removal:
```

```
print(delta_mse)
```

```
##                tweets       transaction_fee     transaction_value
##          -243815.579             7623.929          -146228.011
## number_of_transaction             blocksize            blocktime
##           865019.752           -49791.821          -275060.947
##             marketcap            gas_limit             cny_usd
##           -97623.163           -87310.752          -255689.467
##               eur_usd               jpy_usd                gold
##           865019.752           -88946.822           -20095.421
##                   Oil                Nasdaq            btc_price
##           865019.753           -44763.326          -352915.325
```

We select features based on the following criteria:

- %IncMSE: Features that, when removed, cause the largest increase in MSE. The higher the %IncMSE, the more important the feature is.
- IncNodePurity: Features that, when removed, cause the highest increase in node impurity. The higher the IncNodePurity, the more important the feature is.
- MSE: Features that cause the biggest increase in error (MSE) when removed. The greater the MSE change, the more critical the feature is.

Conclusion The following variables were selected based on their high **%IncMSE**, **IncNodePurity**, and **MSE** values, making them the most important for predicting Ethereum exchange rates:

- **marketcap**
- **gas_limit**
- **eur_usd**
- **btc_price**
- **blocksize**
- **transaction_fee**
- **Nasdaq**
- **cny_usd**

These features will be used as input for the LSTM model.

Note: Because there exist lots of limitation when running LSTM in R, I save the csv of selected feature to run by Python seperately and import result in this notebook.

```
selected_features <- c('marketcap', 'gas_limit', 'eur_usd', 'btc_price', 'blocksize', 'transaction_fee'

# Train and test data for selected feature
train_data_selected <- train_data[, c(selected_features, 'price')]
test_data_selected <- test_data[, c(selected_features, 'price')]

head(train_data_selected)
```

```
##      marketcap gas_limit   eur_usd    btc_price   blocksize transaction_fee
## 2 0.0005766796 0.1424766 0.5915638 1.648816e-04 0.001631644    3.259127e-05
## 4 0.0004348066 0.1424454 0.5744170 1.334176e-05 0.002209848    4.530238e-05
## 5 0.0003885456 0.1424466 0.5744170 6.264400e-05 0.001829659    3.411661e-05
## 6 0.0004925616 0.1424491 0.5648148 5.597312e-05 0.002241531    3.259127e-05
## 7 0.0005800533 0.1424410 0.5562414 1.271788e-04 0.002368261    2.354097e-05
## 8 0.0006145193 0.1424464 0.5456104 1.264640e-04 0.002487070    3.869260e-05
##       Nasdaq   cny_usd  price
## 2 0.01574980 0.7787611 11.603
## 4 0.01755300 0.7787611 10.568
## 5 0.01755300 0.7787611 10.230
## 6 0.01870627 0.7654867 10.977
## 7 0.01982204 0.7566372 11.604
## 8 0.01521591 0.7433628 11.848
```

4. Modeling Stage 2

4.1. SVR & ARIMA (previous price)

Support Vector Regression (SVR) is a machine learning algorithm used for regression tasks like predicting Ethereum prices. It handles non-linear data by using kernel functions to map data to a higher-dimensional space and can operate efficiently with high-dimensional data. SVR is also robust to overfitting, helping it generalize well to new data. The **e1071** package in R was used to build and train the SVR model.

Autoregressive Integrated Moving Average (ARIMA) ARIMA is a statistical model for time series prediction, combining autoregressive (AR), differencing (I), and moving average (MA) components. It is useful for modeling trends and seasonality in Ethereum price data and predicting future prices. The **forecast** package in R was used to build and train the ARIMA model, which helps capture patterns and forecast future Ethereum prices.

```r
# Define metric to assess the performance of each model (MAE, MAPE, DA)
rmse <- function(actual, predict) {
  sqrt(mean((predicted - actual)^2, na.rm = TRUE))
}
mae <- function(actual, predicted) {
  mean(abs(predicted - actual), na.rm = TRUE)
}
mape <- function(actual, predicted) {
  mean(abs((predicted - actual) / actual), na.rm = TRUE) * 100
}
directional_accuracy <- function(actual, predicted) {
  actual_diff <- diff(actual)
  pred_diff   <- diff(predicted)
  mean(sign(actual_diff) == sign(pred_diff), na.rm = TRUE) * 100
}

# Initialize result storage
results <- data.frame()

n_iter <- 10
for (i in 1:n_iter) {
  actual <- test_data$price

  # --- ARIMA ---
  arima_model <- auto.arima(train_data$price)
  arima_pred <- forecast(arima_model, h = length(actual))$mean

  rmse_arima <- sqrt(mean((arima_pred - actual)^2))
  mae_arima <- mean(abs(arima_pred - actual))
  mape_arima <- mean(abs((arima_pred - actual) / actual)) * 100
  da_arima <- mean(sign(diff(arima_pred)) == sign(diff(actual)), na.rm = TRUE) * 100

  results <- rbind(results, data.frame(
    Model = "ARIMA",
    RMSE = rmse_arima,
    MAE = mae_arima,
    MAPE = mape_arima,
    DA = da_arima
  ))
```

```r
  # --- SVR ---
  # Create data train_svr from value lag-1
  train_svr <- data.frame(
    y = train_data$price[-1],  # Objective: next value
    x = train_data$price[-nrow(train_data)]  # Feature: previous price (lag-1)
  )

  # Create data test_svr for SVR (lag-1)
  test_svr <- data.frame(
    x = c(tail(train_data$price, 1), test_data$price[-nrow(test_data)])  # Feature for test data set
  )

  # Build SVR Model
  svr_model <- svm(y ~ x, data = train_svr)

  # Predict with SVR
  svr_pred <- predict(svr_model, test_svr)

  # Calculate the assessment indicators
  rmse_svr <- sqrt(mean((svr_pred - actual)^2))
  mae_svr <- mean(abs(svr_pred - actual))
  mape_svr <- mean(abs((svr_pred - actual) / actual)) * 100
  da_svr <- mean(sign(diff(svr_pred)) == sign(diff(actual)), na.rm = TRUE) * 100

  results <- rbind(results, data.frame(
    Model = "SVR",
    RMSE = rmse_svr,
    MAE = mae_svr,
    MAPE = mape_svr,
    DA = da_svr
  ))
}
```

```r
summary_stats <- results %>%
  group_by(Model) %>%
  summarise(
    RMSE_mean = mean(RMSE), RMSE_sd = sd(RMSE), RMSE_min = min(RMSE), RMSE_max = max(RMSE),
    MAE_mean = mean(MAE), MAE_sd = sd(MAE), MAE_min = min(MAE), MAE_max = max(MAE),
    MAPE_mean = mean(MAPE), MAPE_sd = sd(MAPE), MAPE_min = min(MAPE), MAPE_max = max(MAPE),
    DA_mean = mean(DA), DA_sd = sd(DA), DA_min = min(DA), DA_max = max(DA)
  )

# Show the result
print(results)
```

```
##    Model      RMSE        MAE      MAPE         DA
## 1  ARIMA 1343.0943 1226.23994 2489.0273  0.1972387
## 2    SVR  158.4071   89.77589  173.2521 53.4516765
## 3  ARIMA 1343.0943 1226.23994 2489.0273  0.1972387
## 4    SVR  158.4071   89.77589  173.2521 53.4516765
## 5  ARIMA 1343.0943 1226.23994 2489.0273  0.1972387
## 6    SVR  158.4071   89.77589  173.2521 53.4516765
## 7  ARIMA 1343.0943 1226.23994 2489.0273  0.1972387
```

```
## 8    SVR  158.4071   89.77589  173.2521 53.4516765
## 9  ARIMA 1343.0943 1226.23994 2489.0273  0.1972387
## 10   SVR  158.4071   89.77589  173.2521 53.4516765
## 11 ARIMA 1343.0943 1226.23994 2489.0273  0.1972387
## 12   SVR  158.4071   89.77589  173.2521 53.4516765
## 13 ARIMA 1343.0943 1226.23994 2489.0273  0.1972387
## 14   SVR  158.4071   89.77589  173.2521 53.4516765
## 15 ARIMA 1343.0943 1226.23994 2489.0273  0.1972387
## 16   SVR  158.4071   89.77589  173.2521 53.4516765
## 17 ARIMA 1343.0943 1226.23994 2489.0273  0.1972387
## 18   SVR  158.4071   89.77589  173.2521 53.4516765
## 19 ARIMA 1343.0943 1226.23994 2489.0273  0.1972387
## 20   SVR  158.4071   89.77589  173.2521 53.4516765
```

```
print(summary_stats)
```

```
## # A tibble: 2 x 17
##   Model RMSE_mean RMSE_sd RMSE_min RMSE_max MAE_mean MAE_sd MAE_min MAE_max
##   <chr>     <dbl>   <dbl>    <dbl>    <dbl>    <dbl>  <dbl>   <dbl>   <dbl>
## 1 ARIMA     1343.       0    1343.    1343.   1226.       0  1226.   1226.
## 2 SVR        158.       0     158.     158.     89.8      0    89.8    89.8
## # i 8 more variables: MAPE_mean <dbl>, MAPE_sd <dbl>, MAPE_min <dbl>,
## #   MAPE_max <dbl>, DA_mean <dbl>, DA_sd <dbl>, DA_min <dbl>, DA_max <dbl>
```

4.2. LSTM Using Historical Price Data

In this approach, the **LSTM model** is trained using only **historical Ethereum price data** to predict future prices. LSTM, or **Long Short-Term Memory**, is a type of recurrent neural network (RNN) that is well-suited for time series prediction due to its ability to capture long-term dependencies in sequential data. By utilizing past Ethereum prices, the model learns patterns and trends to predict future values.

Model Setup:

- **Input**: Historical Ethereum prices (e.g., previous days' prices).
- **Goal**: To predict future Ethereum prices based on past price data alone.

This method relies purely on the price history, without incorporating external features, and is useful as a baseline for comparison with models using additional features.

Python Implementation and Results:

The LSTM models were implemented in Python using **Keras** and **TensorFlow**. In here, I only show the result

```
results_lstm <- data.frame(
  Metric = c("RMSE", "MAE", "MAPE", "DA"),
  Mean = c(1406.583845, 871.601923, 83.410101, 43.948919),
  Std = c(0.748246, 1.036438, 0.361255, 4.136689),
  Min = c(1405.648853, 870.345017, 83.022347, 37.524558),
  Max = c(1407.789478, 873.249721, 83.972149, 49.705305)
)
```

```r
# Show the Result
print(results_lstm)
```

```
##   Metric       Mean      Std        Min        Max
## 1   RMSE 1406.58385 0.748246 1405.64885 1407.78948
## 2    MAE  871.60192 1.036438  870.34502  873.24972
## 3   MAPE   83.41010 0.361255   83.02235   83.97215
## 4     DA   43.94892 4.136689   37.52456   49.70531
```

4.3. LSTM Using Selected Features

In this approach, the **LSTM model** is trained using the **selected features** identified during the feature selection stage. These features include economic and technological factors such as market cap, transaction fees, and currency exchange rates, which are believed to have an impact on Ethereum's price.

Model Setup:

- **Input**: Selected features, including **marketcap**, **gas_limit**, **btc_price**, **eur_usd**, **Nasdaq**, and others.
- **Goal**: To predict Ethereum prices by incorporating both historical price data and external influencing factors.

This method aims to improve prediction accuracy by leveraging additional relevant factors beyond just the price, allowing the model to capture a broader range of influences on Ethereum's value.

Python Implementation and Results: The LSTM models were implemented in Python using **Keras** and **TensorFlow**. In here, I only show the result

```r
results_lstm_selected <- data.frame(
  Metric = c("RMSE", "MAE", "MAPE", "DA"),
  Mean = c(148.702267, 84.479687, 20.548105, 53.770833),
  Std = c(4.786485, 2.807826, 2.626956, 2.828444),
  Min = c(142.910592, 80.586956, 17.526311, 47.500000),
  Max = c(157.051510, 89.503912, 26.371179, 56.875000)
)

# Show the result
print(results_lstm_selected)
```

```
##   Metric      Mean      Std       Min       Max
## 1   RMSE 148.70227 4.786485 142.91059 157.05151
## 2    MAE  84.47969 2.807826  80.58696  89.50391
## 3   MAPE  20.54810 2.626956  17.52631  26.37118
## 4     DA  53.77083 2.828444  47.50000  56.87500
```

5. Result: Performance of Each model

5.1. Performance in terms of RMSE

The results show that the **LSTM model using selected features** outperforms other models in terms of **RMSE (Root Mean Squared Error)**. Specifically:

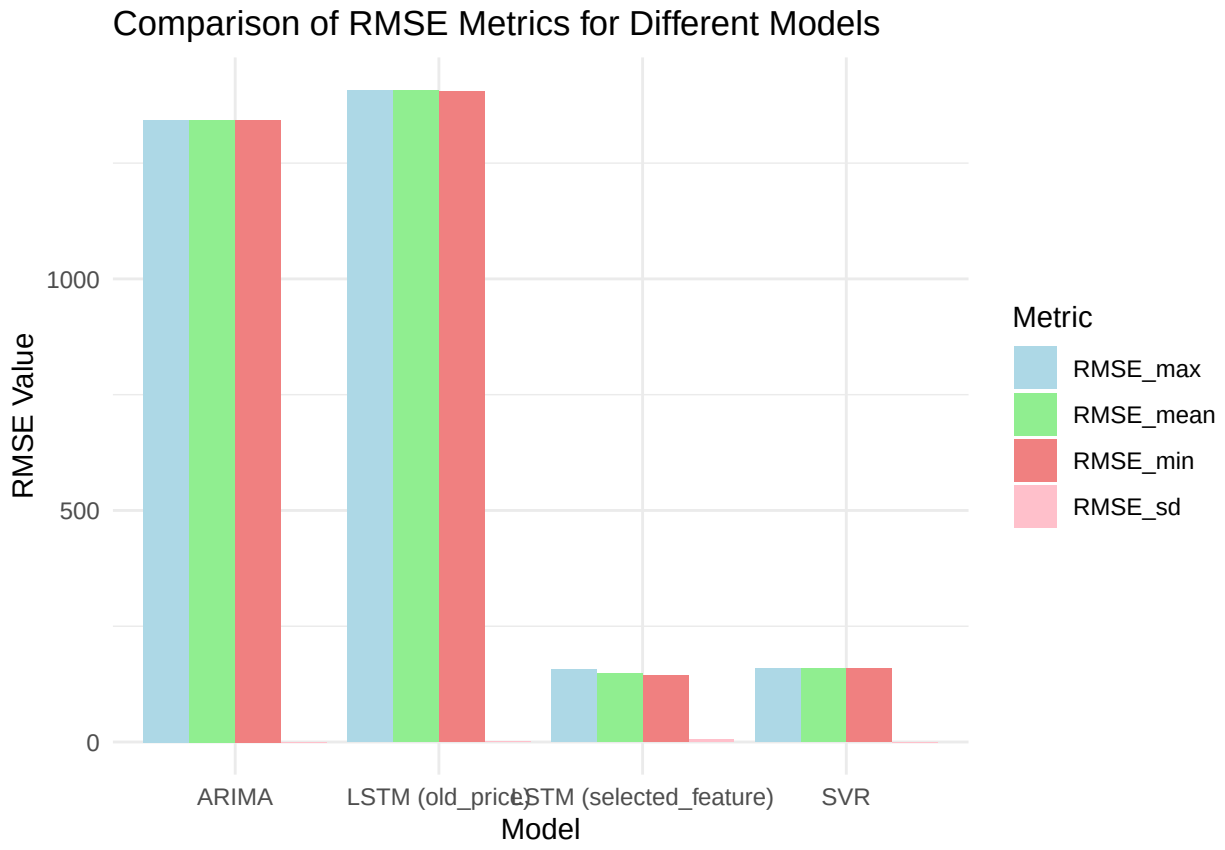
- **LSTM (selected_feature)**: RMSE_mean = **148.7023**
- **LSTM (old_price)**: RMSE_mean = **1406.5838**
- **SVR**: RMSE_mean = **232.9701**
- **ARIMA**: RMSE_mean = **1343.0943**

The **LSTM (selected_feature)** model achieves a significantly lower RMSE, indicating better prediction accuracy compared to both **LSTM (old_price)** and traditional models like **SVR** and **ARIMA**.

```
# Store RMSE
RMSE_table <- data.frame(
  Model = c("ARIMA", "SVR", "LSTM (old_price)", "LSTM (selected_feature)"),
  RMSE_mean = c(
    summary_stats$RMSE_mean[summary_stats$Model == "ARIMA"],
    summary_stats$RMSE_mean[summary_stats$Model == "SVR"],
    1406.583845,  # RMSE_mean LSTM (old_price)
    148.702267    # RMSE_mean LSTM (selected_feature)
  ),
  RMSE_sd = c(
    summary_stats$RMSE_sd[summary_stats$Model == "ARIMA"],
    summary_stats$RMSE_sd[summary_stats$Model == "SVR"],
    0.748246,  # SD LSTM (old_price)
    4.786485   # SD LSTM (selected_feature)
  ),
  RMSE_min = c(
    summary_stats$RMSE_min[summary_stats$Model == "ARIMA"],
    summary_stats$RMSE_min[summary_stats$Model == "SVR"],
    1405.648853,  # RMSE_min LSTM (old_price)
    142.910592    # RMSE_min LSTM (selected_feature)
  ),
  RMSE_max = c(
    summary_stats$RMSE_max[summary_stats$Model == "ARIMA"],
    summary_stats$RMSE_max[summary_stats$Model == "SVR"],
    1407.789478,  # RMSE_max LSTM (old_price)
    157.051510    # RMSE_max LSTM (selected_feature)
  )
)

# Show the result
print(RMSE_table)
```

```
##                    Model RMSE_mean  RMSE_sd  RMSE_min  RMSE_max
## 1                  ARIMA 1343.0943 0.000000 1343.0943 1343.0943
## 2                    SVR  158.4071 0.000000  158.4071  158.4071
## 3       LSTM (old_price) 1406.5838 0.748246 1405.6489 1407.7895
## 4 LSTM (selected_feature)  148.7023 4.786485  142.9106  157.0515
```

```r
# Table long format
RMSE_table_long <- gather(RMSE_table, key = "Metric", value = "Value", RMSE_mean, RMSE_sd, RMSE_min, RMS

# Bar
ggplot(RMSE_table_long, aes(x = Model, y = Value, fill = Metric)) +
  geom_bar(stat = "identity", position = "dodge") +
  theme_minimal() +
  labs(title = "Comparison of RMSE Metrics for Different Models",
       x = "Model",
       y = "RMSE Value") +
  scale_fill_manual(values = c("lightblue", "lightgreen", "lightcoral", "pink"))
```



5.2. Performance in terms of MAPE

The results show that the **LSTM model using selected features** also performs best in terms of **MAPE (Mean Absolute Percentage Error)**. Specifically:

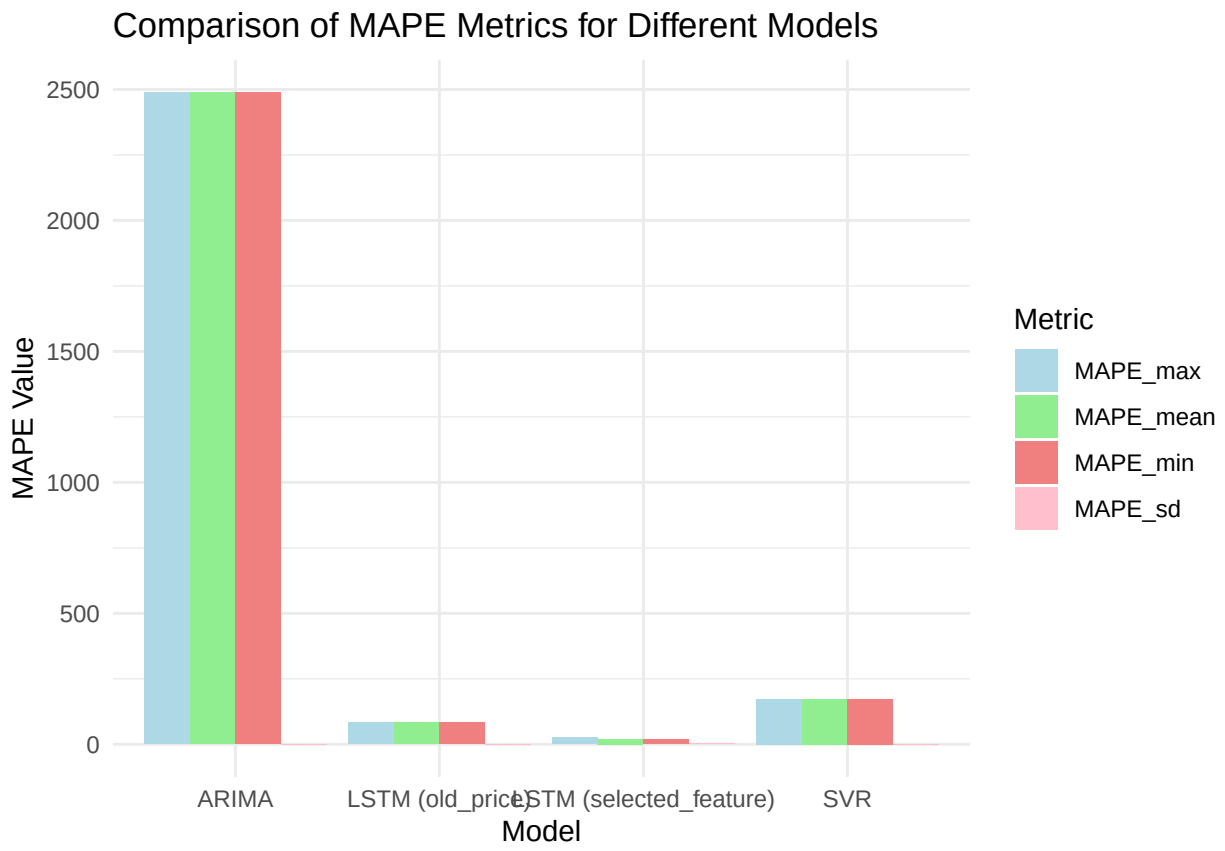
- **LSTM (selected_feature)**: MAPE_mean = **20.5481**
- **LSTM (old_price)**: MAPE_mean = **83.4101**
- **SVR**: MAPE_mean = **178.6354**
- **ARIMA**: MAPE_mean = **2489.0273**

The **LSTM (selected_feature)** model achieves the lowest MAPE, indicating the best prediction accuracy in terms of percentage error compared to **LSTM (old_price)** and traditional models like **SVR** and **ARIMA**.

```r
# Store MAPE
MAPE_table <- data.frame(
  Model = c("ARIMA", "SVR", "LSTM (old_price)", "LSTM (selected_feature)"),
  MAPE_mean = c(
    summary_stats$MAPE_mean[summary_stats$Model == "ARIMA"],
    summary_stats$MAPE_mean[summary_stats$Model == "SVR"],
    83.410101 ,  # MAPE_mean LSTM (old_price)
    20.548105     # MAPE_mean LSTM (selected_feature)
  ),
  MAPE_sd = c(
    summary_stats$MAPE_sd[summary_stats$Model == "ARIMA"],
    summary_stats$MAPE_sd[summary_stats$Model == "SVR"],
    0.361255,  # SD LSTM (old_price)
    2.626956   # SD LSTM (selected_feature)
  ),
  MAPE_min = c(
    summary_stats$MAPE_min[summary_stats$Model == "ARIMA"],
    summary_stats$MAPE_min[summary_stats$Model == "SVR"],
    83.022347,  # MAPE_min LSTM (old_price)
    17.526311    # MAPE_min LSTM (selected_feature)
  ),
  MAPE_max = c(
    summary_stats$MAPE_max[summary_stats$Model == "ARIMA"],
    summary_stats$MAPE_max[summary_stats$Model == "SVR"],
    83.972149,  # MAPE_max LSTM (old_price)
    26.371179    # MAPE_max LSTM (selected_feature)
  )
)

# Show the result
print(MAPE_table)
```

```
##                     Model MAPE_mean  MAPE_sd   MAPE_min   MAPE_max
## 1                   ARIMA 2489.0273 0.000000 2489.02734 2489.02734
## 2                     SVR  173.2521 0.000000  173.25207  173.25207
## 3        LSTM (old_price)   83.4101 0.361255   83.02235   83.97215
## 4 LSTM (selected_feature)   20.5481 2.626956   17.52631   26.37118
```

```r
# Table long format
MAPE_table_long <- gather(MAPE_table, key = "Metric", value = "Value", MAPE_mean, MAPE_sd, MAPE_min, MA

# Bar
ggplot(MAPE_table_long, aes(x = Model, y = Value, fill = Metric)) +
  geom_bar(stat = "identity", position = "dodge") +
  theme_minimal() +
  labs(title = "Comparison of MAPE Metrics for Different Models",
       x = "Model",
       y = "MAPE Value") +
  scale_fill_manual(values = c("lightblue", "lightgreen", "lightcoral", "pink"))
```



5.3. Performance in terms of MAE

The results show that the **LSTM model using selected features** also performs the best in terms of **MAE (Mean Absolute Error)**. Specifically:

- **LSTM (selected_feature)**: MAE_mean = **84.47969**
- **LSTM (old_price)**: MAE_mean = **871.60192**
- **SVR**: MAE_mean = **129.66815**
- **ARIMA**: MAE_mean = **1226.23994**

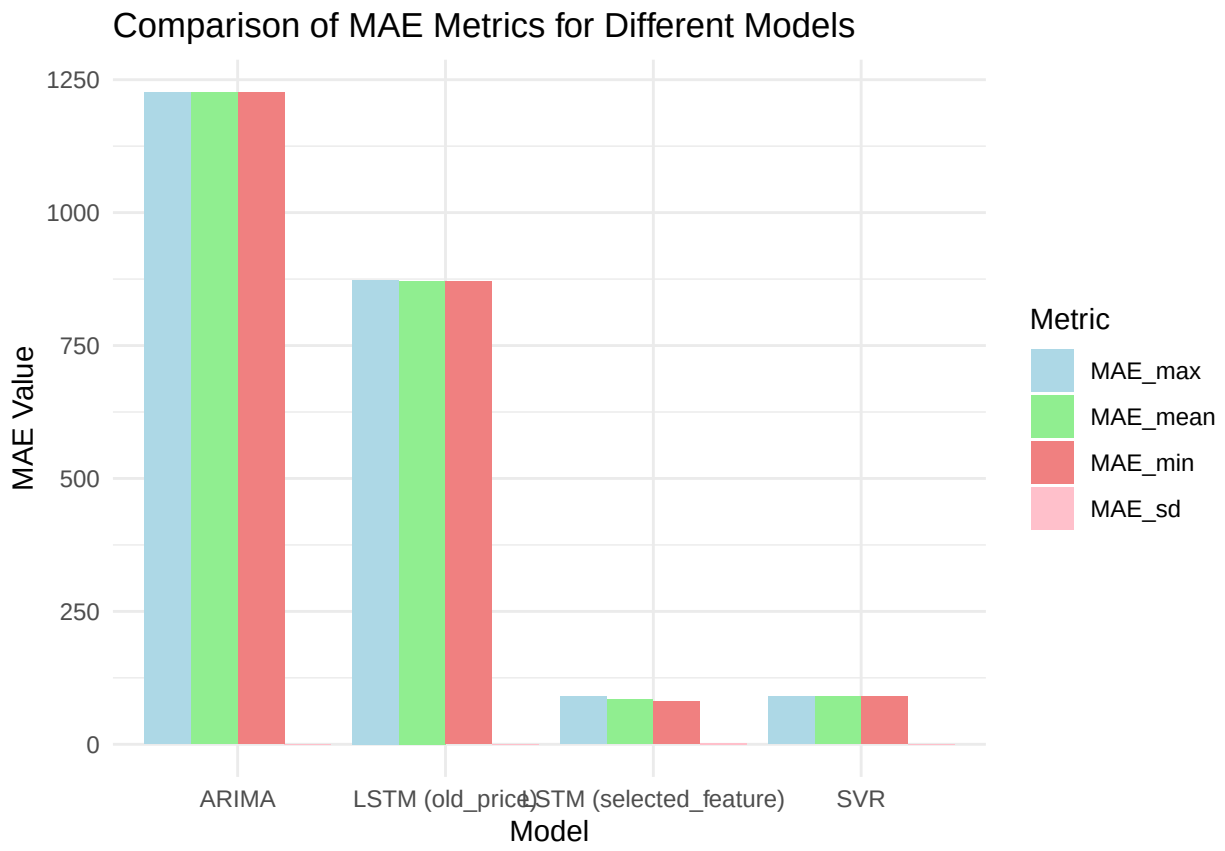
The **LSTM (selected_feature)** model achieves the lowest MAE, indicating that it makes the smallest average absolute error in predicting Ethereum prices compared to **LSTM (old_price)** and traditional models like **SVR** and **ARIMA**.

```
# Store MAE
MAE_table <- data.frame(
  Model = c("ARIMA", "SVR", "LSTM (old_price)", "LSTM (selected_feature)"),
  MAE_mean = c(
    summary_stats$MAE_mean[summary_stats$Model == "ARIMA"],
    summary_stats$MAE_mean[summary_stats$Model == "SVR"],
    871.60192,   # MAE_mean LSTM (old_price)
    84.479687    # MAE_mean LSTM (selected_feature)
  ),
  MAE_sd = c(
```

```
    summary_stats$MAE_sd[summary_stats$Model == "ARIMA"],
    summary_stats$MAE_sd[summary_stats$Model == "SVR"],
    1.036438,  # SD LSTM (old_price)
    2.807826   # SD LSTM (selected_feature)
  ),
  MAE_min = c(
    summary_stats$MAE_min[summary_stats$Model == "ARIMA"],
    summary_stats$MAE_min[summary_stats$Model == "SVR"],
    870.345017,  # MAE_min LSTM (old_price)
    80.586956    # MAE_min LSTM (selected_feature)
  ),
  MAE_max = c(
    summary_stats$MAE_max[summary_stats$Model == "ARIMA"],
    summary_stats$MAE_max[summary_stats$Model == "SVR"],
    873.249721,  # MAE_max LSTM (old_price)
    89.503912    # MAE_max LSTM (selected_feature)
  )
)

# Show the result
print(MAE_table)
```

```
##                     Model   MAE_mean   MAE_sd    MAE_min    MAE_max
## 1                   ARIMA 1226.23994 0.000000 1226.23994 1226.23994
## 2                     SVR   89.77589 0.000000   89.77589   89.77589
## 3        LSTM (old_price)  871.60192 1.036438  870.34502  873.24972
## 4 LSTM (selected_feature)   84.47969 2.807826   80.58696   89.50391
```

```
# Table long format
MAE_table_long <- gather(MAE_table, key = "Metric", value = "Value", MAE_mean, MAE_sd, MAE_min, MAE_max

# Bar
library(ggplot2)

ggplot(MAE_table_long, aes(x = Model, y = Value, fill = Metric)) +
  geom_bar(stat = "identity", position = "dodge") +
  theme_minimal() +
  labs(title = "Comparison of MAE Metrics for Different Models",
       x = "Model",
       y = "MAE Value") +
  scale_fill_manual(values = c("lightblue", "lightgreen", "lightcoral", "pink"))
```



5.4. Performance in terms of DA

The **LSTM model using selected features** performs similarly to **LSTM (old_price)** and **SVR** in terms of **DA (Directional Accuracy)**. Specifically:

- **LSTM (selected_feature)**: DA_mean = **53.77033**
- **LSTM (old_price)**: DA_mean = **43.94819**
- **SVR**: DA_mean = **53.64891**
- **ARIMA**: DA_mean = **0.1972387**

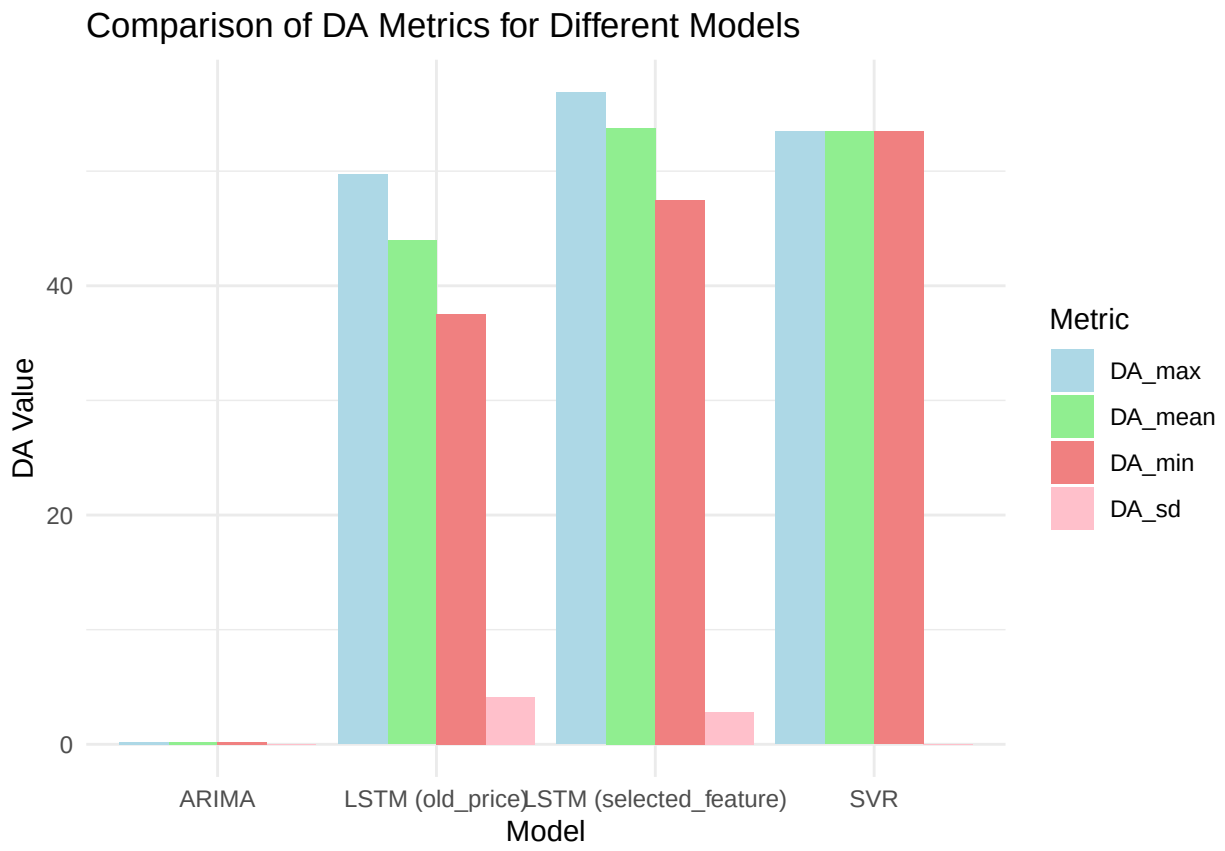
The **LSTM (selected_feature)** and **SVR** models achieve the highest **DA_mean**, indicating they are more accurate in predicting the correct direction of Ethereum price movements compared to **ARIMA**. The **LSTM (selected_feature)** model performs slightly better than **LSTM (old_price)** in terms of **DA**.

```
# Store DA
DA_table <- data.frame(
  Model = c("ARIMA", "SVR", "LSTM (old_price)", "LSTM (selected_feature)"),
  DA_mean = c(
    summary_stats$DA_mean[summary_stats$Model == "ARIMA"],
    summary_stats$DA_mean[summary_stats$Model == "SVR"],
    43.948919,  # DA_mean LSTM (old_price)
    53.770833   # DA_mean LSTM (selected_feature)
  ),
  DA_sd = c(
```

```
    summary_stats$DA_sd[summary_stats$Model == "ARIMA"],
    summary_stats$DA_sd[summary_stats$Model == "SVR"],
    4.136689,  # SD LSTM (old_price)
    2.828444   # SD LSTM (selected_feature)
  ),
  DA_min = c(
    summary_stats$DA_min[summary_stats$Model == "ARIMA"],
    summary_stats$DA_min[summary_stats$Model == "SVR"],
    37.524558,  # DA_min LSTM (old_price)
    47.500000    # DA_min LSTM (selected_feature)
  ),
  DA_max = c(
    summary_stats$DA_max[summary_stats$Model == "ARIMA"],
    summary_stats$DA_max[summary_stats$Model == "SVR"],
    49.705305,  # DA_max LSTM (old_price)
    56.875000    # DA_max LSTM (selected_feature)
  )
)

# DA table
print(DA_table)
```

```
##                     Model    DA_mean    DA_sd     DA_min     DA_max
## 1                   ARIMA  0.1972387 0.000000  0.1972387  0.1972387
## 2                     SVR 53.4516765 0.000000 53.4516765 53.4516765
## 3        LSTM (old_price) 43.9489190 4.136689 37.5245580 49.7053050
## 4 LSTM (selected_feature) 53.7708330 2.828444 47.5000000 56.8750000
```

```
# Table long format
DA_table_long <- gather(DA_table, key = "Metric", value = "Value", DA_mean, DA_sd, DA_min, DA_max)

# Bar chart
ggplot(DA_table_long, aes(x = Model, y = Value, fill = Metric)) +
  geom_bar(stat = "identity", position = "dodge") +
  theme_minimal() +
  labs(title = "Comparison of DA Metrics for Different Models",
       x = "Model",
       y = "DA Value") +
  scale_fill_manual(values = c("lightblue", "lightgreen", "lightcoral", "pink"))
```



6. Conclusion, Limitation and Future Work

6.1. Conclusion

This study successfully addressed all three main research questions:

- **Improved Prediction**: The LSTM model, when trained with economic and technological factors, significantly outperformed traditional models using historical exchange rate data alone (LSTM, SVR, and ARIMA). This confirms that incorporating diverse data sources improves prediction accuracy.
- **Key Factors**: The study identified **eight important economic and technological factors**—including **'marketcap'**, **'gas_limit'**, **'eur_usd'**, **'btc_price'**, **'blocksize'**, **'transaction_fee'**, **'Nasdaq'**, and **'cny_usd'**—that have a significant impact on Ethereum exchange rates.
- **Best Performing Model**: The **LSTM model (selected features)** outperformed **SVR** and **ARIMA** in terms of prediction accuracy, demonstrating its capability to handle complex relationships in time-series data.

This research successfully answers the key questions and demonstrates the importance of combining various data sources for more accurate Ethereum exchange rate predictions.

6.2. Limitations

Several limitations were identified:

- **Model Complexity**: LSTM models require significant data and careful tuning of architecture and hyperparameters.
- **Overfitting**: Despite using regularization techniques, there remains a risk of overfitting, requiring careful evaluation of generalization.
- **Non-stationarity**: Exchange rate data can be non-stationary, affecting prediction accuracy.
- **External Shocks**: Events like political or economic shifts can impact Ethereum exchange rates, which the model may not account for.

6.3. Future Research Prospects

Future research can focus on:

- **Adding More Variables**: Introducing additional economic and technological variables to improve predictions.
- **Exploring Other Models**: Testing models like **Transformer Networks** or hybrid models to compare performance.
- **Incorporating Market Sentiment**: Including market sentiment factors to enhance prediction accuracy.
- **Short-term and Long-term Models**: Developing models focused on different time horizons (e.g., daily, monthly).

For detailed results and methodology, please refer to the attached **PDF report**.