STAT2011 Project Report
Analysis of Prediction Models for Bitcoin Prices

Members:

Kwok Tsun Yau 1155194200
Shi Shing Ho 1155194415
Lam Hoi Chun 1155192755
Chow Ho Fung 1155193434
Yim Sum Yuet 1155193313

Content

Introduction

In 2009, Bitcoin was created as the first cryptocurrency in the world. Bitcoin and even the cryptocurrency market have attracted an enormous amount of interest and investment. Bitcoin was so pioneering as it is a decentralized digital currency, which means transactions can be made peer-to-peer and do not require intermediaries such as banks or financial institutions.

Economically speaking, Bitcoin has recently improved its function as a form of money. It can now store purchasing power and be a medium of exchange. Therefore, its value has been significantly increased, thus leading to intense interest from investors, traders, economists, and researchers worldwide.

Therefore, in order to understand more about the Bitcoin market as well as its value, it is essential to investigate its price dynamics. After that, we can have broader views about cryptocurrency markets, and their financial implications.

This data analysis project aims to investigate the historical price movements of Bitcoin. The analysis will be carried out by fitting various time series models to the data, which are the linear trend model, quadratic trend model, moving average, centered moving average, exponential smoothing, and autoregressive integrated moving average (ARIMA). We expect to gain insights into the trend of Bitcoin price and make a valid prediction on its price.

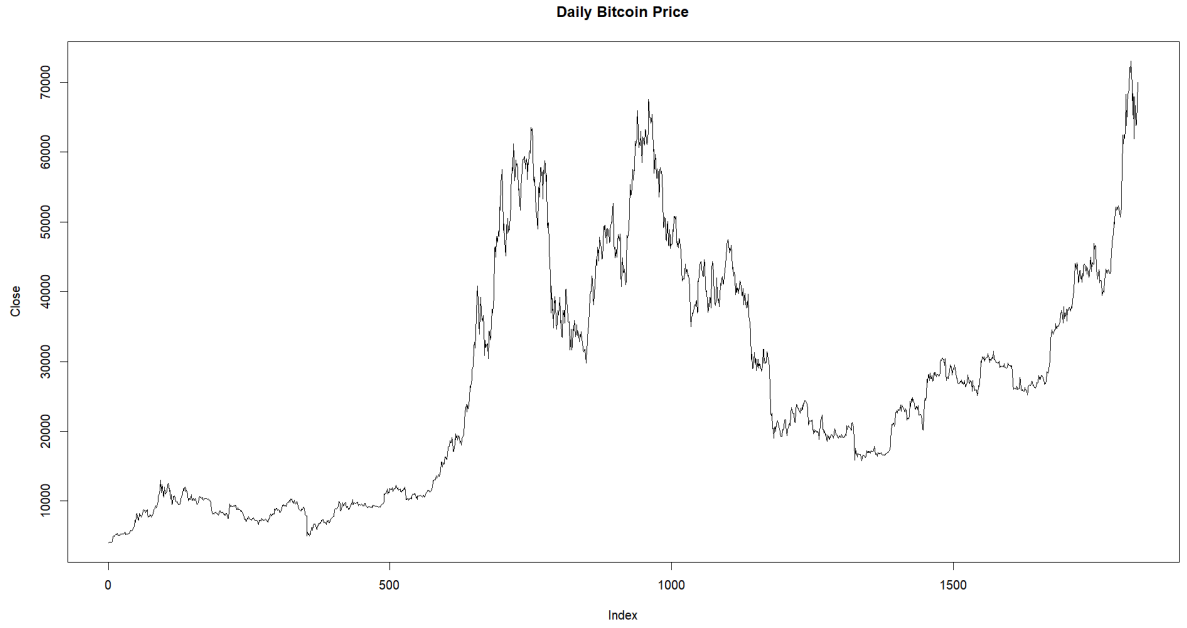
Data Prepossessing

```
> head(data)
        Date     Open     High      Low    Close Adj.Close      Volume
1 2019-03-26 3969.229 3985.081 3944.753 3985.081  3985.081 10707678815
2 2019-03-27 3984.245 4087.066 3977.811 4087.066  4087.066 10897131934
3 2019-03-28 4087.584 4094.902 4040.266 4069.107  4069.107  9353915899
4 2019-03-29 4068.300 4113.501 4034.097 4098.375  4098.375 10918665557
5 2019-03-30 4092.136 4296.807 4053.910 4106.660  4106.660  9732688060
6 2019-03-31 4105.456 4113.023 4094.101 4105.404  4105.404  9045122443
```

We decided to use the Bitcoin data from 26 March, 2019 to 26 March, 2024, which includes 1828 observations in total. We chose to analyze this time range since a longer time range, such as from the start of Bitcoin will yield a large error during the analysis. Meanwhile, we did not choose a time range shorter than 5 years as COVID-19 may have a significant influence on the Bitcoin market. Therefore, we considered the time just before COVID-19. The dataset consists of 7 variables but we decided to use only one variable “Close” to perform analysis.

```
> sum(is.na(data))
[1] 0
```

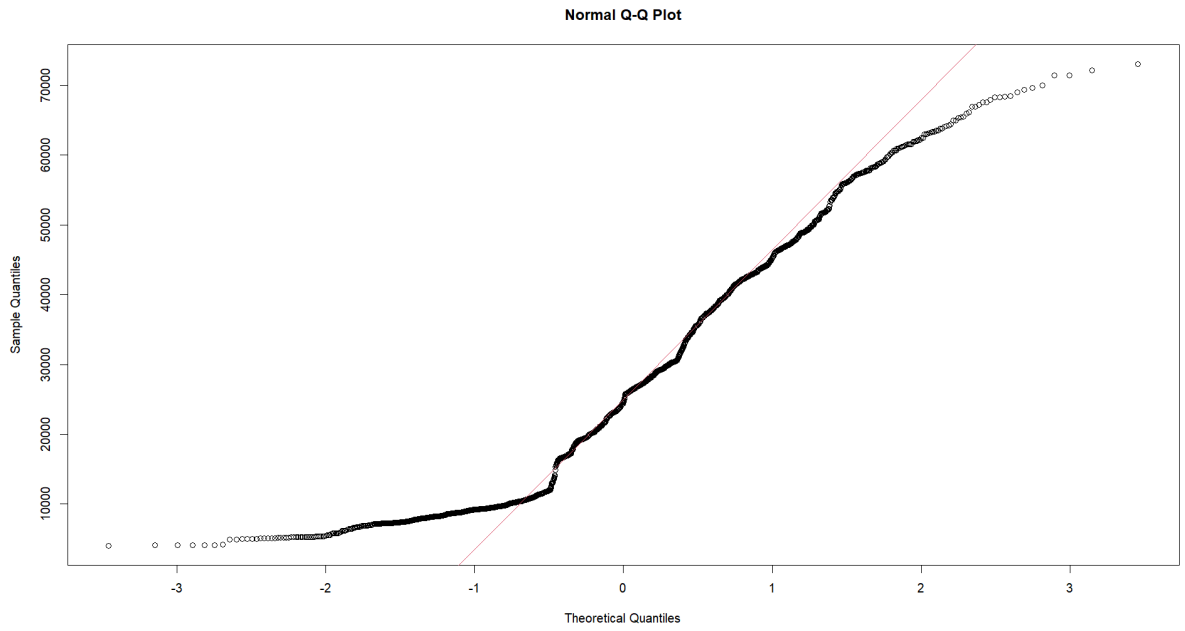
We confirmed that there is no missing value within the dataset.



First, the graph above shows the daily Bitcoin (close) price. Given that our dataset consists of Bitcoin price data from 2019 to 2024, we can first analyze some visible trends before heading to practical trend prediction. Starting in early 2019, the price was stable with only a small amount of fluctuations. After June 2019, there was a sudden and significant upsurge in the Bitcoin price, which surpassed the $10,000 threshold in a short period of time. Nonetheless, by mid-December, Bitcoin price had declined to around $6,000.

Then, in 2020, there was an outbreak of the Covid-19 pandemic, which shut down the global economy immediately. Thus, the economy was significantly disrupted, and the government lockdown and other related policies exacerbated investors' worries regarding the overall economic situation.

Nevertheless, in early 2021, there was again a skyrocketing trend in the Bitcoin price. In April 2021, Bitcoin prices soared to the highest and unprecedented price ever, exceeding $60,000. This surge can be explained by the initial public offering (IPO) of Coinbase, a prominent cryptocurrency exchange, which yields more confidence and interest in the Bitcoin market.



The Q-Q plot suggests that the Bitcoin price is not normally distributed. It can be explained by the market sentiment, which can lead to periods of irrational exuberance or panic selling, causing price movements that deviate from a normal distribution.

Linear Trend Model


```
> summary(linear_model)

Call:
lm(formula = close ~ time)

Residuals:
   Min     1Q Median     3Q    Max
-18211  -9820  -6381   9341  39948

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 1.144e+04  6.548e+02   17.46   <2e-16 ***
time        1.688e+01  6.201e-01   27.21   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 13990 on 1826 degrees of freedom
Multiple R-squared:  0.2885,    Adjusted R-squared:  0.2881
F-statistic: 740.5 on 1 and 1826 DF,  p-value: < 2.2e-16
```

Although the Bitcoin price is not very likely to have a linear relationship with time, we tried to perform linear regression to yield a linear trend model. In the summary of the linear model, the multiple R-squared measures the proportion of variance in Bitcoin price that can be explained by time, which is around 28.85%. Therefore, we cannot show a significant relationship between the Bitcoin price and time.

Errors of the linear model:
Mean Squared Error (MSE): 195,555,363
Root Mean Squared Error (RMSE): 13,983
Mean Absolute Error (MAE): 11,553

Quadratic Trend Model


```
> summary(quadratic_model)

Call:
lm(formula = close ~ time + time_squared)

Residuals:
   Min     1Q Median     3Q    Max
-20195 -10486  -4204   7378  41586

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)  363.813560 919.605039   0.396    0.692
time          53.175714   2.322214  22.899   <2e-16 ***
time_squared  -0.019847   0.001229 -16.144   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 13090 on 1825 degrees of freedom
Multiple R-squared:  0.3774,    Adjusted R-squared:  0.3768
F-statistic: 553.2 on 2 and 1825 DF,  p-value: < 2.2e-16
```

This aims to fit the quadratic trend model to the bitcoin price time series using linear regression and evaluate its ability to predict future prices. The graph provided includes a red line, representing the predicted values of the bitcoin price based on the quadratic trend model, and a black line, representing the actual bitcoin price over time.

Upon examining the graph, it is evident that there is minimal overlap between the red and black lines. This lack of overlap suggests that the predicted values from the quadratic trend model do not align well with the actual Bitcoin price observations. We can conclude that there may not be a strong quadratic relationship between time and the Bitcoin price.

It is important to note that drawing definitive conclusions only based on the graph is limited, further analysis is necessary to fully understand the relationship between time and the bitcoin price.

Errors of the quadratic trend model:
Mean Squared Error (MSE): 171,124,531
Root Mean Squared Error (RMSE): 13,081
Mean Absolute Error (MAE): 10,746

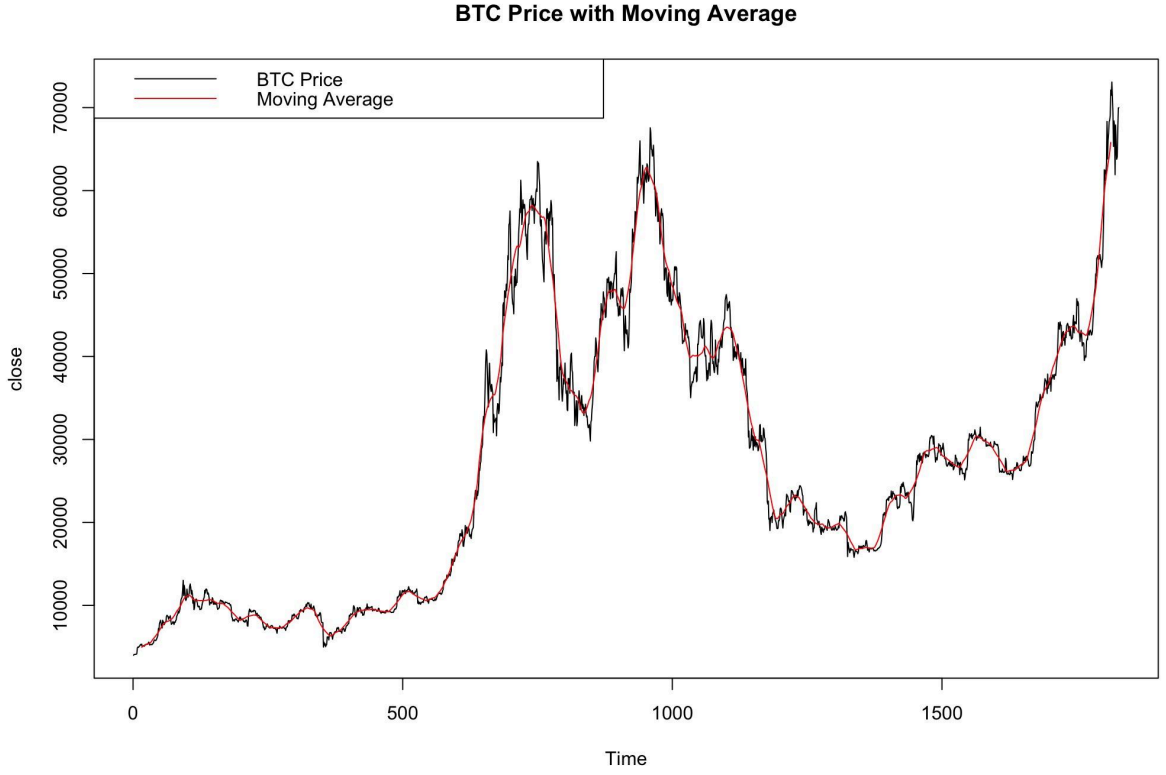
Moving Average

The moving average is calculated by taking the average of a set number of points from the dataset. It helps to smooth out short-term fluctuations and highlight longer-term trends by averaging the data points over a specified number of periods.

A moving average is calculated by taking the average of a set number of points from the dataset. For instance, a 30-day moving average calculates the average of the closing prices for the last 30 days and "moves" or progresses with each new day added to the dataset. This process is repeated throughout the length of the time series, creating a smoothed line that can make it easier to identify patterns and trends within volatile datasets like Bitcoin prices.

The following is the graph we plot using the moving average method. The black line is the actual value of the bitcoin price while the red one is the moving average line.

Errors of the moving average model:
Mean Squared Error (MSE): 29,96,257
Root Mean Squared Error (RMSE): 1,731
Mean Absolute Error (MAE): 1,135



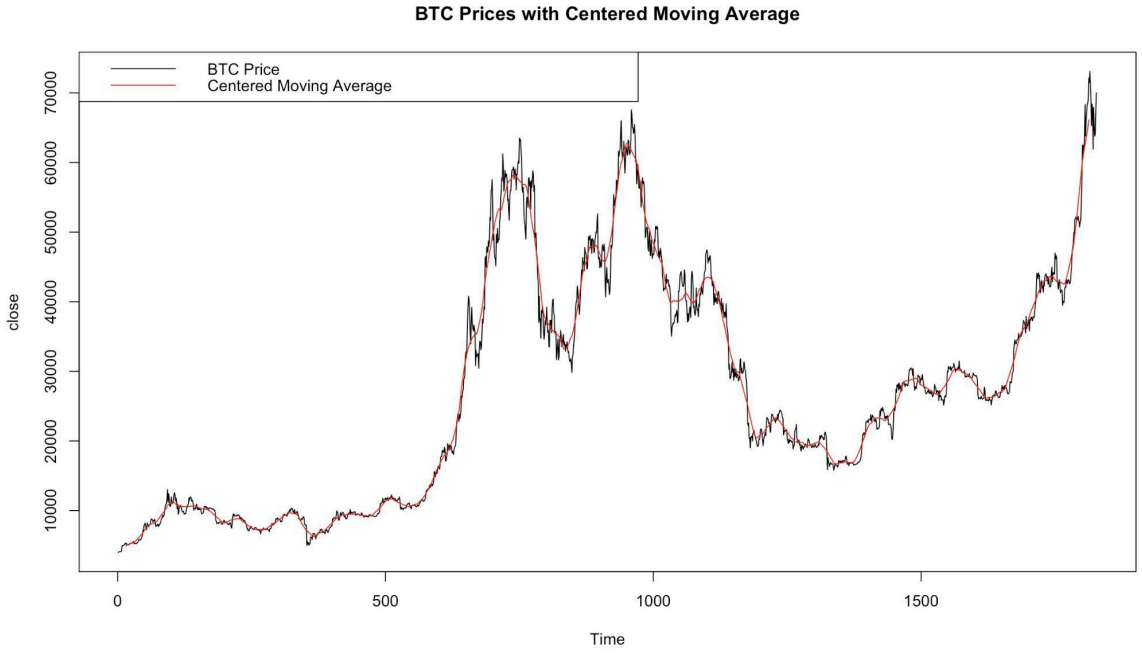
Centered Moving Average

Apart from the moving average method, a centered moving average model is applied to the bitcoin price time series. It is another type of moving average used in time series analysis. Unlike the more common trailing moving average, which is aligned to the right of the data point and uses previous data points to calculate the average, the centered moving average is aligned to the center of the data window and uses data points from both before and after the point in time for which it's calculated. This method can provide a better indicator of the trend for that moment in time because it centers the average around the period of interest.

The following graph is plotted using the centered moving average method. The black line and red line represent the actual value and centered moving average line, which is calculated over a window size of 30, respectively. Each point on this red line is the average of the 15 days before and 15 days after the date, including the date itself. This calculation balances the data and aligns the moving average precisely with the period it's averaging, which can provide a more accurate reflection of the trend at each point.

The centered moving average can show the underlying trend in the Bitcoin price without the noise of daily fluctuations. The CMA line will be smoother than the price line and can be used to identify upward or downward trends.

Errors of the centered moving average model:
Mean Squared Error (MSE): 3,009,322
Root Mean Squared Error (RMSE): 1,735
Mean Absolute Error (MAE): 1,136



Exponential Smoothing

Introduction:

Time series data can be analyzed statistically using the exponential moving average (EMA). It is one moving average method, where earlier data points are kept included but recent data points are given more weight. EMA gives the most recent data points more weight, and progressively less weight is assigned to older data points, in contrast to simple moving averages, which offer equal weight to all data points. This indicates that EMA can assist in identifying trends and patterns faster and is more responsive to recent changes in the data.

$$F_{t+1} = \alpha Y_t + (1-\alpha) F_t$$

The forecast for the next period (F_{t+1}) is calculated using the exponential smoothing formula above, which takes into account both the current period (F_t) forecast and the actual value (Y_t). When calculating the updated forecast, the weight assigned to the actual value vs the anticipated value is determined by the smoothing constant (α).

For our Bitcoin price dataset, we can apply the exponential smoothing formula to compute the forecast for the Bitcoin price in the next period based on the actual Bitcoin price in the current period and the forecast for the current period.

By adjusting the value of α, we can control the degree to which the forecast responds to changes in the actual values. A bigger value of α would result in a more responsive forecast that is better able to capture abrupt changes in the data, while a smaller value would place more weight on the prior forecast, producing a smoother forecast.

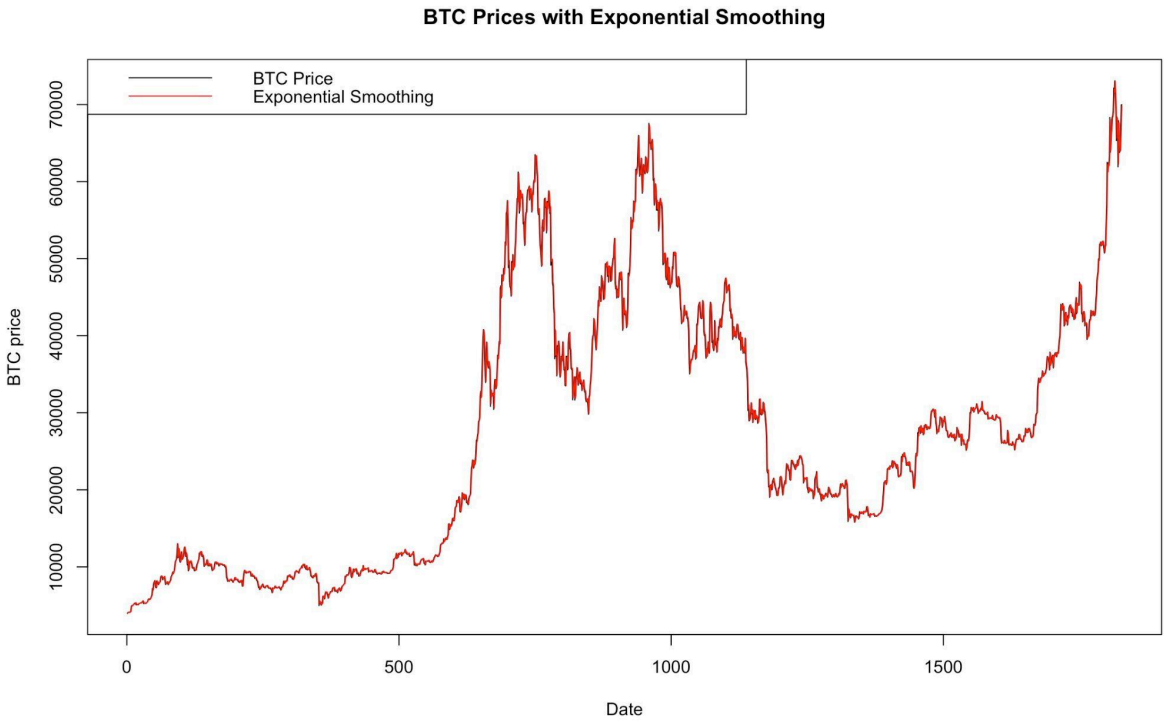
Determining the smoothing constant (α) — Ets() function:

The ets() function in R can assist us in obtaining the best-fitted value of α. In the R console, we put our Bitcoin time series object to the ets() function, and then we can find all parameters of exponential smoothing. Putting the time series value of the data into the function and the below parameters can be obtained: The compiler tells us the best-fitted value of α is 0.9395. It assists us in building up the best model.

```
> summary(ets_BTC)
ETS(M,A,N)

Call:
 ets(y = close)

  Smoothing parameters:
    alpha = 0.9395
    beta  = 8e-04

  Initial states:
    l = 3894.1253
    b = 45.4575

  sigma:  0.0351

     AIC     AICc      BIC
37962.09 37962.12 37989.64

Training set error measures:
                    ME     RMSE      MAE         MPE     MAPE      MASE         ACF1
Training set 1.823861 1111.385 646.5592 -0.1246465 2.347185 0.9952803 0.009438803
```

Graph Plotting

We calculate the forecast value each year, and we compare the forecast values (F_t) with the actual values (Y_t).



The black and red lines represent the actual value (Y_t) and the forecast value (F_t) obtained through exponential smoothing respectively.

It can be observed that the exponential smoothing method is accurate because the red line is very close to the black line. When comparing its forecast error with other methods, exponential smoothing gives us the smallest error, meaning that its accuracy is the highest among other prediction methods in our analysis.

Errors of the exponential smoothing model:
Mean Squared Error (MSE): 1,235,176
Root Mean Squared Error (RMSE): 1,111
Mean Absolute Error (MAE): 646

Nonetheless, although we obtained the most accurate forecast result by exponential smoothing this time, it is not necessarily the most accurate method in all situations since it might be affected by other factors. We can obtain the most accurate forecast result is likely because we used R to find the most suitable smoothing constant (α), at this time, the best-fitted value of α is 0.9395, meaning that our smoothing method places more weight on the actual values than the previous forecast, resulting in a more responsive forecast that is better able to capture sudden changes in the data. Therefore, it is suitable for our Bitcoin price dataset, which is unstable and fluctuates.

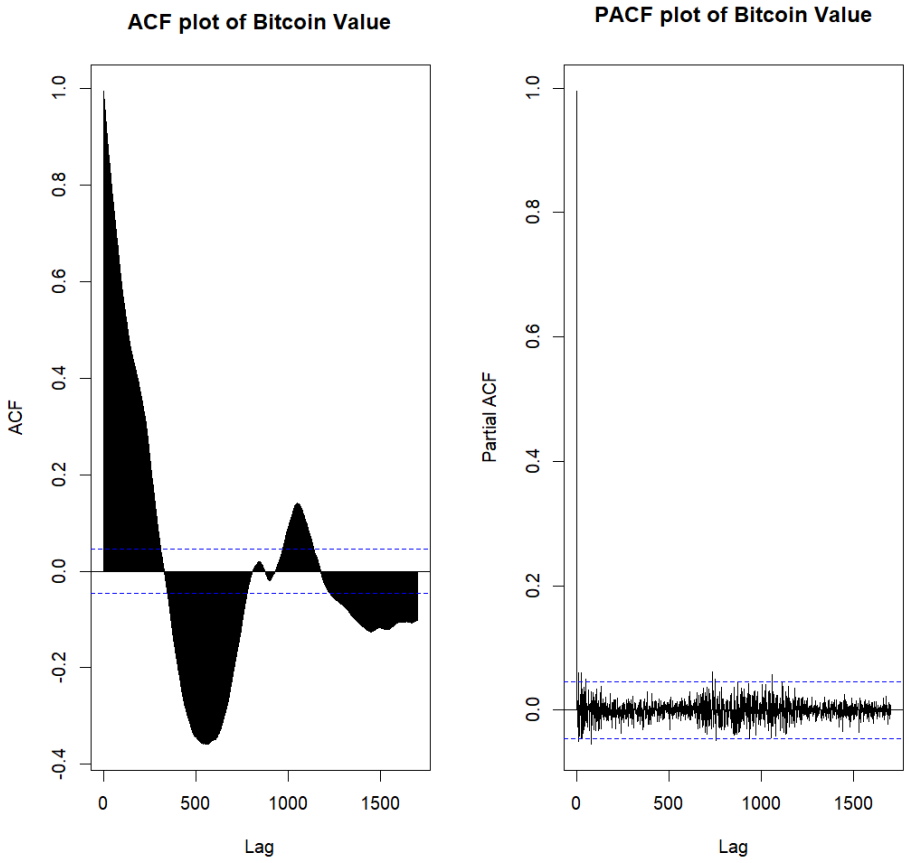
Autoregressive Integrated Moving Average (ARIMA)

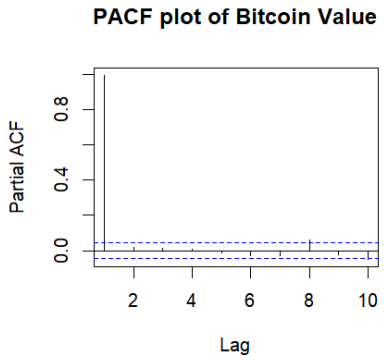
Introduction:

A ARIMA model can be divided into 3 components: autoregression (AR), differencing (I) and moving average (MA), which corresponds to the parameter of the mode: p, d, and q. The selection of these parameters is important to ensure the performance of the model. They can be determined using techniques such as autocorrelation function (ACF) and partial autocorrelation function (PACF) plots.

We decided to use the ARIMA model for prediction since Bitcoin price maintains significant dependencies, which means past prices have significant impacts on future prices. The AR component can effectively capture such a relationship.

Besides, Bitcoin prices may exhibit trends and seasonality. The differencing component can be used to remove the trends and seasonality, making the data stationary and suitable for modeling. Therefore, we can forecast the Bitcoin price effectively.





Firstly, the ACF (Autocorrelation Function) plot shows a fluctuating pattern indicating a trend in the bitcoin price. From the graph, at lag *k*, it measures the correlation between observations that are *k* days apart. Moreover, since there is a sharp drop-off in the ACF graph, it implies that previous observations are not valuable in predicting future values.

Besides, the PACF (Partial Autocorrelation Function) shows the correlation between the Bitcoin price and its lag, after removing the linear dependence of the Bitcoin price on the values at shorter lags. The graph has only one significant lag, which is near 0. It implies that the time series is non-stationary.

Gathering the information from the previous ACF and PACF plots, they both do not suggest that the time series is stationary. Therefore, to confirm its stationarity, we decided to conduct an ADF Unitroot test.

An Augmented Dickey-Fuller (ADF) test is a statistical test that is commonly used to conduct hypothesis testing to determine whether a time series is stationary or non-stationary.

H_0: The data has a unit root, i.e. non-stationary
H_1: The data does not have a unit root, i.e. stationary

We have to determine the *k* in the adf.test() function, which is the lag order. The lag order is the number of past observations included in the model to account for potential serial correlation.

A typical method to determine the lag order (*k*) is to interpret the PACF plot and find a point where the autocorrelation drops off sharply. In Fig.1, it can be observed that there is a huge drop where *k* is close to 0. Therefore, we take a close look at the PACF graph, by setting lag.max = 10. It can be seen that the huge drop happens when lag order = 2. Thus, we will use *k* = 2 in the operation afterward.

```
> adf.test(close, k=2)

        Augmented Dickey-Fuller Test

data:  close
Dickey-Fuller = -1.0946, Lag order = 2, p-value = 0.9235
alternative hypothesis: stationary
```

For *k* = 2, the corresponding p-value = 0.9235. Consider a common significance level of 0.05, since the calculated p-value > 0.05, we fail to reject H_0. Then, we can conclude that the time series is considered non-stationary.

```
close.diff1 <- diff(close,1)
ts.plot(close.diff1)
```

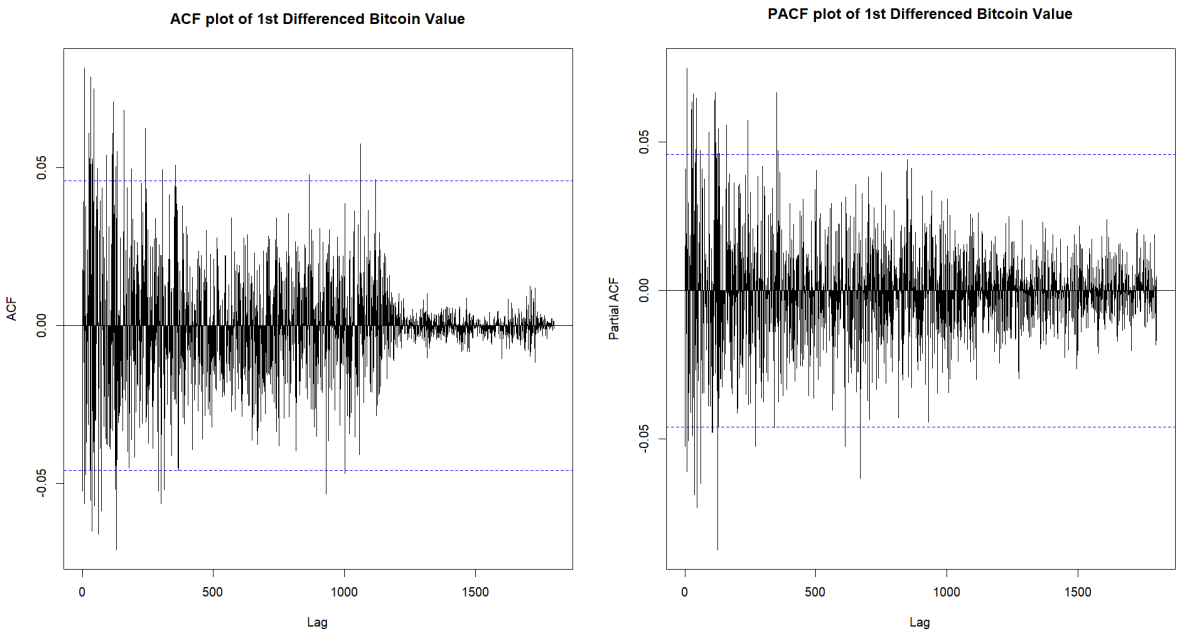
If time series data is non-stationary, due to different trends, seasonality, or other patterns, is impossible to conduct time series analysis and forecasting because the ARIMA model assumes stationarity.

Therefore, after confirming the original time series is non-stationary, we decided to transform the data into stationary using a common technique, which is differencing. Differencing achieves the goal by removing non-constant trends. It excludes the trends and seasonality, thus stabilizing mean and variance over time. ([1904.07632] Why Are the ARIMA and SARIMA not Sufficient (arxiv.org))

Therefore, we apply differencing to the data and transform the data into time series data.

```
        Augmented Dickey-Fuller Test

data:  close.diff1
Dickey-Fuller = -23.734, Lag order = 2, p-value = 0.01
alternative hypothesis: stationary

警告訊息：
於 adf.test(close.diff1, k = 2)： p-value smaller than printed p-value
```

Similarly, we conduct an ADF test on the new data after differencing. We keep the lag order (k) to be 2. Now, p-value < 0.01. So, we reject H_o and conclude that the time series is now stationary.



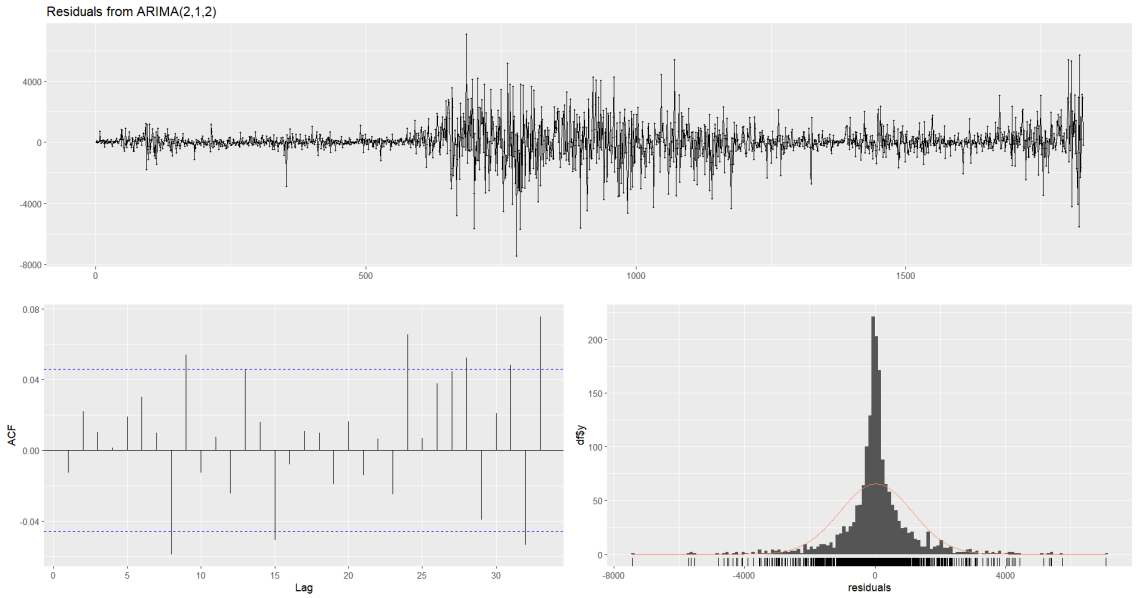
After transforming the data into stationary, we can create an ARIMA model. Usually, we observe the spikes of the ACF and PACF plots to determine the parameters for the ARIMA model: p and q. However, from the ACF and PACF plots using the first-differenced data, it is too many values to be tested. Therefore, we decide to use the auto.arima() function to determine p and q.

Using auto.arima() function, we can find out that ARIMA(2,1,2) is the best ARIMA model. We decided to only use the first 1800 observations to train our model, and the rest will be compared with the predicted values.

```
> checkresiduals(arima_model)

        Ljung-Box test

data:  Residuals from ARIMA(2,1,2)
Q* = 15.839, df = 6, p-value = 0.01465

Model df: 4.   Total lags used: 10
```

H_0 : The residuals / data are independently distributed
H_1 : The residuals / data are dependently distributed

The calculated p-value from the Ljung-Box test is less than the common significance level of 0.05. Therefore, we can conclude that the data is not independent.

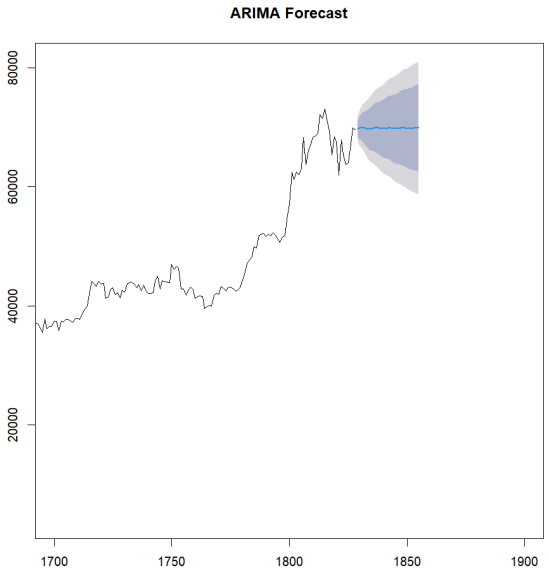


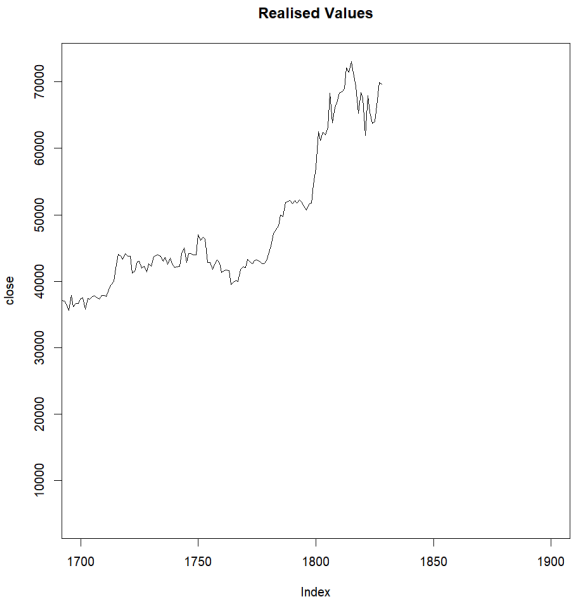
From the ACF plot, it can be observed that there are significant spikes, which are beyond the dashed lines (95% confidence intervals). It suggests that the residuals are not white noise and that there are still patterns or dependencies in the data that the model has not captured. This indicates potential inadequacies in the model.

Also, as there is a large number of observations, the residuals appear to be normally distributed.

```
> (arima_model_forecast = forecast(arima_model,h = 27))
     Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
1829       69719.76 68297.83 71141.69 67545.11 71894.41
1830       69960.97 67992.68 71929.26 66950.73 72971.21
1831       70071.97 67679.42 72464.52 66412.88 73731.07
1832       69954.31 67175.78 72732.85 65704.91 74203.72
1833       69758.60 66620.95 72896.25 64959.97 74557.22
1834       69697.22 66241.05 73153.39 64411.46 74982.98
1835       69816.76 66086.65 73546.87 64112.05 75521.47
1836       69970.69 65996.39 73944.99 63892.52 76048.86
1837       69995.36 65786.44 74204.28 63558.38 76432.34
1838       69881.54 65438.00 74325.08 63085.73 76677.35
1839       69764.50 65091.76 74437.23 62618.17 76910.83
1840       69765.76 64879.53 74651.99 62292.92 77238.60
1841       69869.11 64787.29 74950.93 62097.13 77641.08
1842       69954.71 64687.79 75221.63 61899.65 78009.77
1843       69936.21 64485.90 75386.51 61600.68 78271.73
1844       69845.88 64211.70 75480.06 61229.15 78462.62
1845       69786.27 63972.56 75599.98 60894.97 78677.58
1846       69815.19 63831.50 75798.89 60663.92 78966.47
1847       69891.55 63747.11 76035.99 60494.45 79288.66
1848       69930.34 63629.57 76231.11 60294.14 79566.53
1849       69896.15 63439.31 76352.99 60021.27 79771.03
1850       69833.57 63221.17 76445.97 59720.78 79946.37
1851       69810.95 63046.93 76574.97 59466.27 80155.63
1852       69846.66 62937.29 76756.02 59279.70 80413.62
1853       69896.38 62846.77 76945.99 59114.94 80677.83
1854       69906.89 62719.05 77094.73 58914.04 80899.75
1855       69872.26 62546.42 77198.10 58668.35 81076.17
```

We used the forecast() function to predict the Bitcoin price for the next 27 days. The function provides the point forecast, lower bound and upper bound for 80% and 95% confidence.

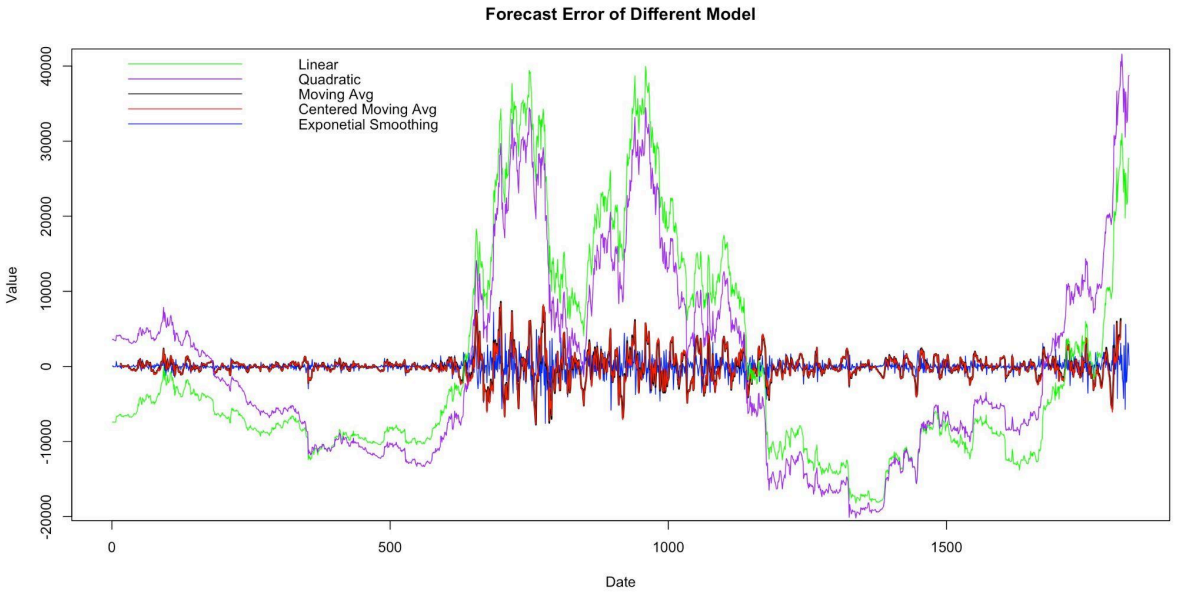


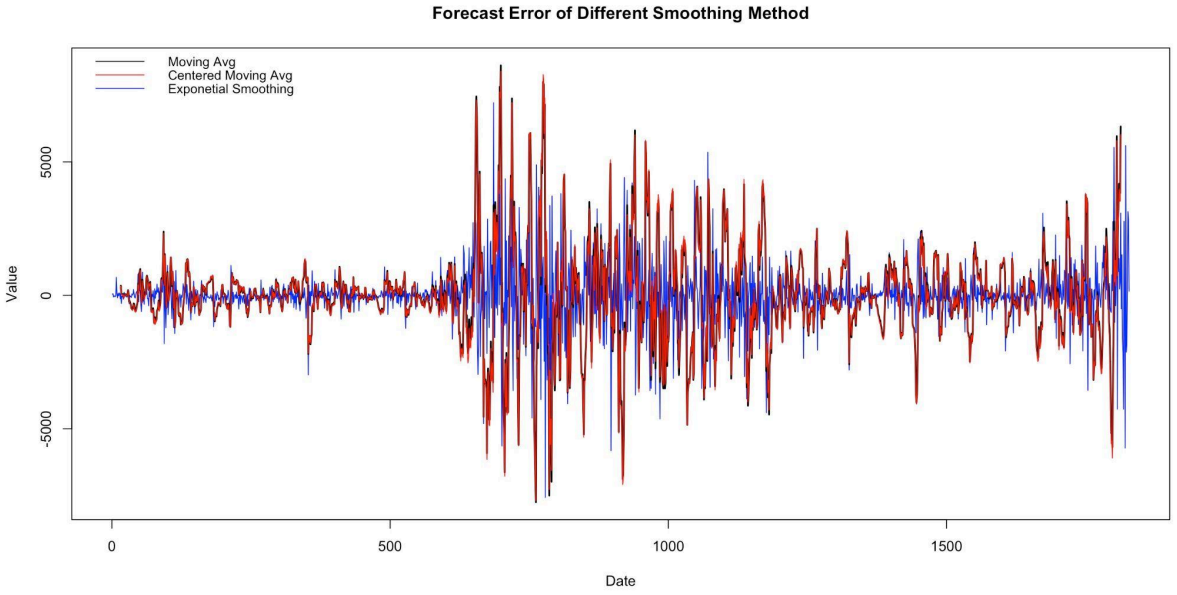



```
> # Calculating RMSE
> arima_model_forecast_mean <- arima_model_forecast$mean
> arima_rmse <- sqrt(mean((arima_model_forecast_mean - close[1801:1827])^2))
> print(paste("Root Mean Squared Error (RMSE): ", arima_rmse))
[1] "Root Mean Squared Error (RMSE):  4604.82650447842"
>
> # Calculating MSE
> arima_mse <- mean((arima_model_forecast_mean - close[1801:1827])^2)
> print(paste("Mean Squared Error (MSE): ", arima_mse))
[1] "Mean Squared Error (MSE):  21204427.1363469"
>
> # Calculating MAE
> arima_mae <- mean(abs(arima_model_forecast_mean - close[1801:1827]))
> print(paste("Mean Absolute Error (MAE): ", arima_mae))
[1] "Mean Absolute Error (MAE):  3783.798452776"
```

In ARIMA model prediction, we decided to use the first 1800 observations as training data and the rest will be compared with the predicted values. The model prediction graph and their errors are shown above.

Forecast Error Comparison





Forecast Error Comparison Table

	Mean Squared Error	Root Mean Squared Error	Mean Absolute Error
Linear	195,555,363	13,983	11,553
Quadratic	171,124,531	13,081	10,746
M.A	2,996,257	1,730	1,135
C.M.A	3,009,322	1,734	1,136
Exponential	1,235,176	1,111	646
ARIMA	21,204,427	4,604	3783

Conclusion and Insights

In conclusion, based on the provided error metrics, the exponential smoothing model demonstrates the best forecasting performance, followed by the moving average model, centered moving average model, ARIMA model quadratic model and linear model.

To summarize all the processes we have gone through, firstly, the ADF test conducts hypothesis testing to determine whether a time series is stationary or non-stationary. After the test, we can conclude that the time series is non-stationary. Therefore, the linear trend model is not accurate as data fluctuates. Linear trend models can only provide a simple and intuitive approach to forecasting, however, they may not be suitable for all types of data, especially non-stationary data. Besides, linear trend models are rigid in their assumptions and may not capture complex or nonlinear relationships between variables. They do not account for interactions, non-linear dependencies, or other factors that can influence the data's behavior.

Secondly, it can be observed that the calculated errors from the moving average method and the centered moving average method are nearly the same. The difference between the two models is solely that the centered moving average model calculates the average of a fixed window with an equal number of data points on both sides of the central point. Moving averages are usually applied to smooth out short-term fluctuations and identify trends. In contrast, centered moving averages offer a smoother representation and prove especially beneficial in capturing cyclic or seasonal patterns present in the data. Therefore, centered moving averages would be more appropriate if data consists of cyclic or seasonal patterns.

Furthermore, exponential smoothing is better than the above-mentioned because the exponential smoothing method combines the advantages of the full-period average and the moving average. It does not discard past data but instead assigns diminishing weights as the data points get farther away. It implies that the influence of past data gradually weakens, with the weights converging towards zero.

In addition, there are several reasons that the ARIMA model is not the best prediction model. To begin with, ARIMA models assume that there are linear relationships between variables. However, cryptocurrency prices like Bitcoin are highly fluctuating and can exhibit nonlinear patterns. Moreover, we use the auto.arima() function to help us to choose the parameters p and q for the model. Although it is certain that they can yield accurate prediction results, we are not able to conclude that they are the best parameters. Selecting the optimal parameters requires iterative testing and tuning, which is time-consuming and computationally intensive.

Appendix

```
# ============================ Import Packages ================================ #
install.packages('TSA')
install.packages('tseries')
install.packages('forecast')
install.packages("lmtest")
library(TSA)
library(tseries)
library(psych)
library(ggplot2)
library(forecast)
library(lmtest)
library(zoo)
# ========================== II. Data Prepossessing ============================ #
# Import Data
data <- read.csv("BTC-USD.csv")
head(data)
# Check for any missing values
sum(is.na(data))
# Basic Plot for all variables against time
plot(data, main='Time Series of Daily Bitcoin Price')
# Extract the "Close" variable
close <- data[,"Close"]
head(close)
# Plot: "Close" against time
plot(close, type="l", main='Daily Bitcoin Price', ylab="Close")
# Normal QQ-Plot
qqnorm(close)
qqline(close, col=2)
# ========================= III. Linear Trend Model ============================ #
time <- 1:length(close)
linear_model <- lm(close ~ time)
summary(linear_model)
# Visualization
plot(close, type="l", main='Daily Bitcoin Price', ylab="Close")
abline(linear_model, col="red")
# Check whether the residuals ~ N
linear_residuals <- linear_model$residuals
qqnorm(linear_residuals)
qqline(linear_residuals, col="red")
# Calculate RMSE
linear_rmse <- sqrt(mean(linear_residuals^2))
print(paste("Root Mean Squared Error (RMSE): ", linear_rmse))
# Calculate MAE
linear_mae <- mean(abs(linear_residuals))
print(paste("Mean Absolute Error (MAE): ", linear_mae))
# ============================ IV. Quadratic Model ============================= #
quadratic_model <- lm(close ~ time + time^2)
summary(quadratic_model)
# Plotting the time series and quadratic trend model
# // Cant generate correct graph // #
plot(time, close, type = "l", xlab = "Time", ylab = "BTC Price", main = "Quadratic Trend Model of BTC price")
lines(time, predict(quadratic_model), col = "red")
legend("topleft", legend = c("Actual Price", "Quadratic Model"), col = c("black", "red"), lty = c(1, 1))
# Calculate RMSE
print(paste("Root Mean Squared Error (RMSE): ", sqrt(mean(quadratic_model$residuals^2))))
```

```r
# Calculate MAE
print(paste("Mean Absolute Error (MAE): ", mean(abs(quadratic_model$residuals))))
# ========================== V. Moving Average= ============================== #
# Calculate the moving average with a window size of 30
moving_avg <- ma(close, order = 30)
# Plot the original time series and the moving average
plot(close, main = "Time Series with Moving Average")
lines(moving_avg, col = "red")
legend("topleft", legend = c("Time Series", "Moving Average"), col = c("black", "red"), lty = c(1, 1))
# Calculate RMSE
ma_rmse <- sqrt(mean((close-moving_avg)^2, na.rm = TRUE))
print(paste("Root Mean Squared Error (RMSE): ", ma_rmse))
# Calculate MAE
ma_mae <- mean(abs(close-moving_avg), na.rm = TRUE)
print(paste("Mean Absolute Error (MAE): ", ma_mae))
# ========================= VI. Centered Moving Average ========================== #
centered_ma <- rollmean(close, k = 30, align = "center", fill = NA)
# Plot the original time series and the centered moving average
plot(close, main = "BTC Prices with Centered Moving Average")
lines(centered_ma, col = "red")
legend("topleft", legend = c("BTC Price", "Centered Moving Average"), col = c("black", "red"), lty = c(1, 1))
# Calculate RMSE
centered_ma_rmse <- sqrt(mean((close - centered_ma)^2, na.rm = TRUE))
print(paste("Root Mean Squared Error (RMSE): ", centered_ma_rmse))
# Calculate MAE
centered_ma_mae <- mean(abs(close - centered_ma), na.rm = TRUE)
print(paste("Mean Absolute Error (MAE): ", centered_ma_mae))
# ========================= VII. Exponential Smoothing ========================== #
ets_BTC<- ets(close)
# Plot
plot(close, main = "BTC Prices with Exponential Smoothing", xlab = "Date", ylab = "BTC price")
lines(ets_BTC$fitted, col = "red", lwd = 2)
# Calculate RMSE
ets_rmse <- sqrt(mean((close - ets_BTC$fitted)^2, na.rm = TRUE))
print(paste("Root Mean Squared Error (RMSE): ", ets_rmse))
# Calculate MAE
ets_mae <- mean(abs(close - ets_BTC$fitted), na.rm = TRUE)
print(paste("Mean Absolute Error (MAE): ", ets_mae))
# ======================== VIII. ARIMA Model (Final) ============================ #
# Different ACF & PACF Plots
par(mfrow=c(1,2))
acf(close,lag.max = 800, main="ACF plot of Bitcoin Value")
pacf(close,lag.max = 800, main="PACF plot of Bitcoin Value")
par(mfrow=c(1,1))
pacf(close,lag.max = 10, main="PACF plot of Bitcoin Value")
# ADF test for stationary data
adf.test(close, k=2)
close.diff1 <- diff(close,1)
ts.plot(close.diff1)
adf.test(close.diff1, k=2)
# ACF & PACF Plots with 1st Differenced Bitcoin Value
par(mfrow=c(1,2))
acf(close.diff1, lag.max = 1800, main = "ACF plot of 1st Differenced Bitcoin Value")
pacf(close.diff1, lag.max = 1800, main = "PACF plot of 1st Differenced Bitcoin Value")
# ARIMA Model Selection
auto.arima(close,stepwise = F,approximation = F,trace = T)
arima_model = Arima(close, order = c(2,1,2), include.drift = FALSE)
checkresiduals(arima_model)
```

```
# ARIMA Model Forecast Value & Plots
(arima_model_forecast = forecast(arima_model,h = 27))
par(mfrow=c(1,2))
plot(arima_model_2_forecast, xlim=c(1700,1900), main = "ARIMA Forecast")
plot(close,type="l", xlim=c(1700,1900), main = "Realised Values")
# Calculating RMSE
arima_model_forecast_mean <- arima_model_forecast$mean
arima_rmse <- sqrt(mean((arima_model_forecast_mean - close[1801:1827])^2))
print(paste("Root Mean Squared Error (RMSE): ", arima_rmse))
# Calculating MAE
arima_mae <- mean(abs(arima_model_forecast_mean - close[1801:1827]))
print(paste("Mean Absolute Error (MAE): ", arima_mae))
# ========================= IX Forecast Comparsion ============================== #
# Combined (All) Plot
ts_of_forecast_error_of_linear_model <- ts(linear_residuals)
ts_of_forecast_error_of_quadratic_model <- ts(quadratic_model$residuals)
ts_of_forecast_error_of_moving_avg <- ts(close-moving_avg)
ts_of_forecast_error_of_centered_moving_avg <- ts(close - centered_ma)
ts_of_forecast_error_of_ets <- ts(close - ets_BTC$fitted)
plot(ts_of_forecast_error_of_ets, type = "l", col = "blue", xlab = "Date", ylab = "Value", main = "Forecast Error of Different
Model", ylim=c(min(ts_of_forecast_error_of_linear_model),max(ts_of_forecast_error_of_linear_model)))
lines(ts_of_forecast_error_of_moving_avg, type = "l", col = "black", lwd=2)
lines(ts_of_forecast_error_of_centered_moving_avg, type = "l", col = "red")
lines(ts_of_forecast_error_of_linear_model, type = "l", col = "green")
lines(ts_of_forecast_error_of_quadratic_model, type = "l", col = "purple")
par(xpd = TRUE)
legend("topleft", legend = c("Linear", "Quadratic","Moving Avg","Centered Moving Avg","Exponetial Smoothing"), col =
c("green", "purple","black","red","blue"), lty = c(1,1,1,1,1),bty = "n", xjust = 0, yjust = 0)
par(xpd = FALSE)
# Combined (Smoothing) Plot
plot(ts_of_forecast_error_of_ets, type = "l", col = "blue", xlab = "Date", ylab = "Value", main = "Forecast Error of Different
Smoothing Method",
ylim=c(min(ts_of_forecast_error_of_moving_avg,na.rm=TRUE),max(ts_of_forecast_error_of_moving_avg,na.rm=TRUE)))
lines(ts_of_forecast_error_of_moving_avg, type = "l", col = "black", lwd=2)
lines(ts_of_forecast_error_of_centered_moving_avg, type = "l", col = "red")
par(xpd = TRUE)
legend("topleft", legend = c("Moving Avg","Centered Moving Avg","Exponetial Smoothing"), col = c("black","red","blue"),
lty = c(1,1,1),bty = "n", xjust = 0, yjust = 0, cex=0.9)
par(xpd = FALSE)
```

Reference

Edwards, J. (2024, March 14). Bitcoin's Price History. Investopedia. Retrieved April 17, 2024, from https://www.investopedia.com/articles/forex/121815/bitcoins-price-history.asp#:~:text=The%20pandemic%20shutdown%20and%20subsequent,the%20start%20of%20that%20year

Coin market cap. (n.d.).Bitcoin USD (BTC-USD).Yahoo Finance. Retrieved April 17, 2024, from https://finance.yahoo.com/quote/BTC-USD/history/