Project 3

Sia Gao

2024-12-06

Introduction

Bitcoin, the world's first decentralized cryptocurrency, has become a focal point of interest for financial analysts, data scientists, and economists. Since its inception in 2009, Bitcoin has witnessed a meteoric rise in popularity and valuation, making it a significant subject of study in the realm of financial forecasting. Forecasting Bitcoin returns is particularly challenging due to its inherent volatility, speculative nature, and sensitivity to global economic events.

This project aims to develop and compare various time series forecasting models to predict Bitcoin returns, leveraging the robust tools available in R. The dataset, obtained from the R package's Bitcoin data, provides historical values of Bitcoin prices, which will be transformed into returns for analysis. By examining and refining different models, including ARIMA, ETS, Holt-Winters, NNETAR, Prophet, and a combination of these methods, the study will try to identify the best forecasting approach based on training and testing errors. Insights from this analysis can contribute to a better understanding of Bitcoin's price dynamics and aid in financial decision-making.

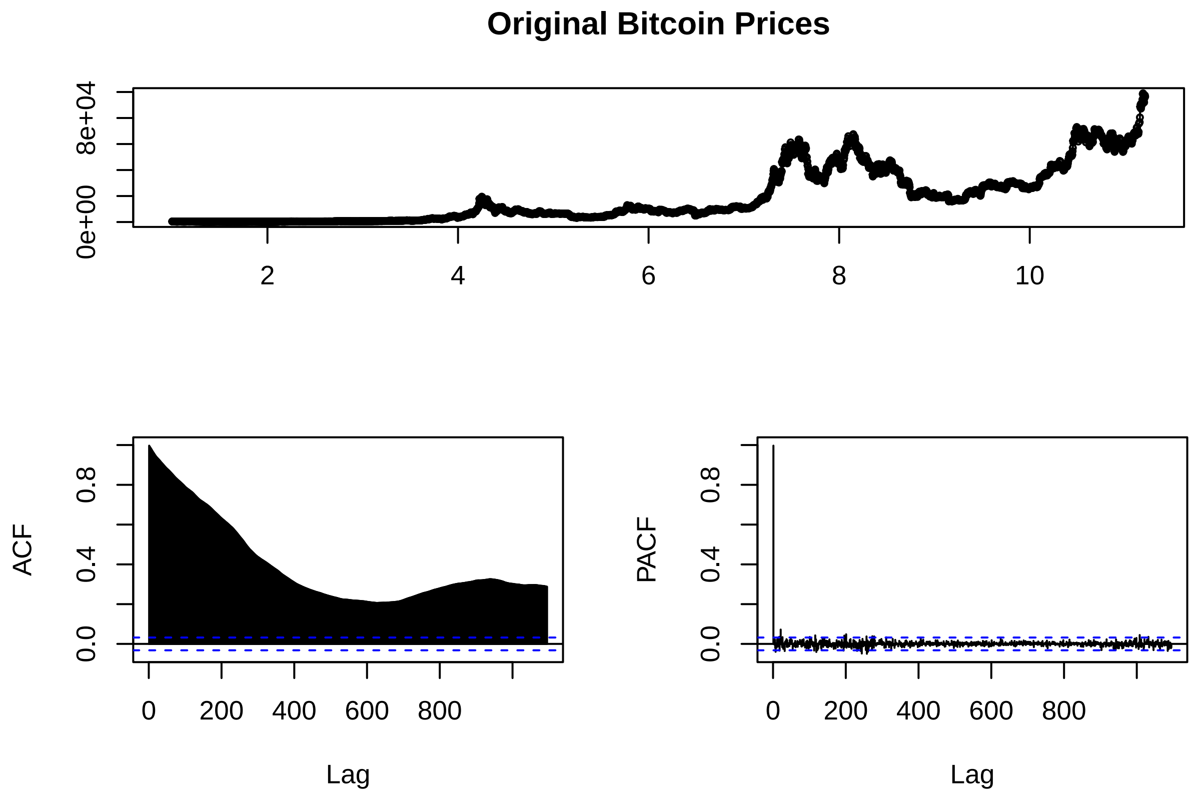
Data

For this project, instead of manually download data from the website, we're importing Bitcoin data directly using the `quantmod` package.

```r
# Fetch historical Bitcoin data from Yahoo Finance
getSymbols("BTC-USD", src = "yahoo", from = "2010-01-01", to = Sys.Date())
```

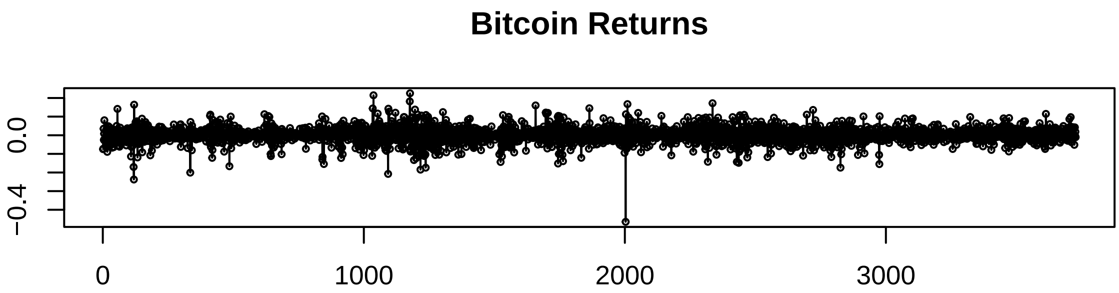
```
## [1] "BTC-USD"
```

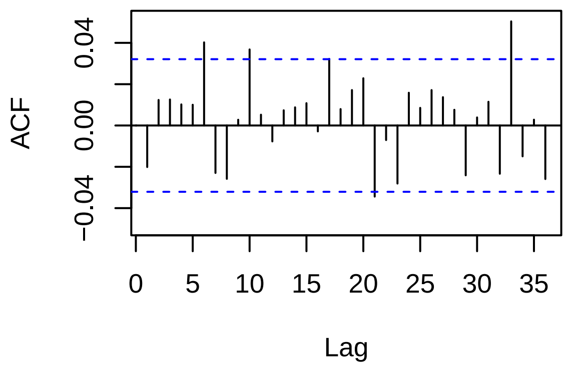
```r
# Use the closing price of Bitcoin and omit missing values
bitcoin_prices <- na.omit(Cl(`BTC-USD`))

# Convert to time series object with the daily frequency
bitcoin_ts <- ts(bitcoin_prices, frequency = 365)

# Plot the original Bitcoin Prices data
tsdisplay(bitcoin_ts, main = "Original Bitcoin Prices")
```

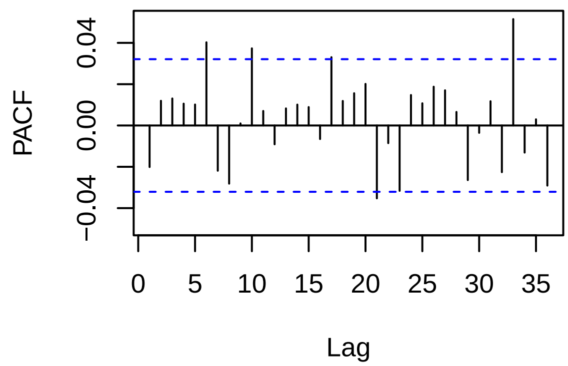


From the graph of bitcoin prices, we observed that the data has an obvious upward trend with some potential cycles. However, since most of the models prefer stationary data and our research questions is focus on the returns of the bitcoin prices in order to observe the dynamic of market, we are now taking the difference to get the returns and check for the staionarity.

```
bitcoin_returns <- diff(log(bitcoin_prices))
bitcoin_returns <- na.omit(bitcoin_returns)

# Plot returns
tsdisplay(bitcoin_returns, main = "Bitcoin Returns")
```







Compared with the Bitcoin prices graph, the Bitcoin returns are much more similar to the white noises; however, there are still some spikes in the ACF and PACF, so we need to use some further official methods - ADF test - to determine the stationarity of the Bitcoin Returns Data.

```r
# Test for stationarity of the returns
adf_test_returns <- adf.test(bitcoin_returns)
print(adf_test_returns)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  bitcoin_returns
## Dickey-Fuller = -14.662, Lag order = 15, p-value = 0.01
## alternative hypothesis: stationary
```

From the p-value of the Augmented Dickey-Fuller Test, we found a significant result, which allows us to reject the null hypothesis and conclude that the Bitcoin returns data is stationary.

After confirming the stationarity of the data, we are now splitting the data into training and testing sets to forecast using the same dataset for all models, allowing for a horizontal comparison.

```r
# Define the training set size as 80% of the total dataset
train_size <- floor(0.8 * length(bitcoin_returns))

# Split the data into training (first 80%) and testing (last 20%)
train_data <- bitcoin_returns[1:train_size]  # First 80%
```

```r
test_data <- bitcoin_returns[(train_size + 1):length(bitcoin_returns)]# Last 20%

# Define the forecasting horizon (length of the test set)
forecast_horizon <- length(test_data)

# # Combine all data (training + testing) into a single time series
all_data <- ts(c(train_data, test_data), start = start(bitcoin_returns),
               frequency = frequency(bitcoin_returns))

# Convert training data to a time series
train_data <- ts(train_data, start = start(all_data),
                 frequency = frequency(all_data))

# Convert testing data to a time series
test_data <- ts(test_data, start = time(all_data)[train_size + 1],
                frequency = frequency(all_data))
```

Now, the data has been split into 80% for training and the remianing 20% for testing. We're going to use `train_data` and `test_data` in order to build models and test for the models.

Results

ARIMA

For this step, we're going to build the arima model by using the training set and then forecast the testing set part, which has a forecast_horizon of 20% of the dataset.

```r
# Fit the ARIMA model to the training data
arima_model <- auto.arima(train_data)

# Print the ARIMA model summary
summary(arima_model)
```

```
## Series: train_data
## ARIMA(2,0,0) with non-zero mean
##
## Coefficients:
##           ar1     ar2    mean
##       -0.0194  0.0064  0.0012
## s.e.   0.0183  0.0183  0.0007
##
## sigma^2 = 0.001514:  log likelihood = 5449.16
## AIC=-10890.33   AICc=-10890.32   BIC=-10866.33
##
## Training set error measures:
##                        ME       RMSE        MAE  MPE MAPE      MASE
## Training set 5.572383e-07 0.03889444 0.02543681 -Inf  Inf 0.6694431
##                       ACF1
## Training set -8.676864e-05
```

From the summary of the arima model, we noticed that the model includes two autoregressive terms (AR) and no moving average terms (MA). Since the data is already stationary, no differencing should be applied.

The weak AR coefficients suggest that past values (lags) have limited influence on the current values. - ar1 = -0.0195: Indicates a weak, slightly negative influence of the first lag on the current value. - ar2 = 0.0064: Indicates an even weaker positive influence of the second lag.

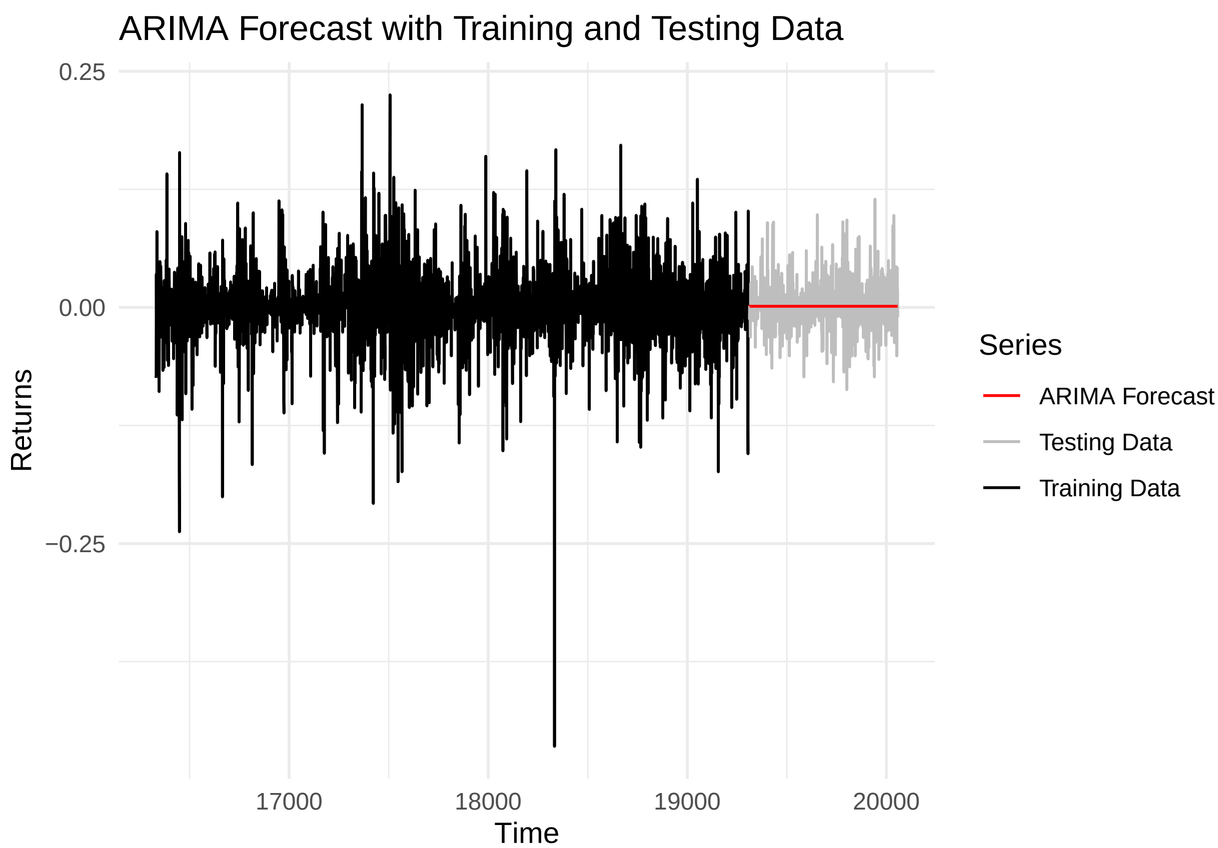
```
# Test for AIC and BIC
arima_aic <- AIC(arima_model)
arima_bic <- BIC(arima_model)
cat("The AIC of the arima model is: ",
    arima_aic, "; and the BIC is: ", arima_bic, ".")
```

```
## The AIC of the arima model is:  -10890.33 ; and the BIC is:  -10866.33 .
```

Since the AIC and BIC are necessary when choosing a final model, we're listing the AIC and BIC above for further comparsions.

After building the model, let's forecast it:

```
# Generate forecasts for the testing period
arima_forecast <- forecast(arima_model, h = forecast_horizon)

# Convert ARIMA forecast to a time series aligned with the testing set
arima_forecast <- ts(arima_forecast$mean,
                        start = time(test_data)[1],
                        frequency = frequency(all_data))

# Plot training data, testing data, and ARIMA forecast
autoplot(train_data, series = "Training Data") +
  autolayer(test_data, series = "Testing Data") +
  autolayer(arima_forecast, series = "ARIMA Forecast") +
  ggtitle("ARIMA Forecast with Training and Testing Data") +
  ylab("Returns") +
  xlab("Time") +
  theme_minimal() +
  scale_color_manual(values = c("red", "grey", "black")) +
  labs(color = "Series")
```



From the plots, the ARIMA forecast is a horizontal line. It is reasonable since the data is stationary and the validity is smaller compared to the training set validity.

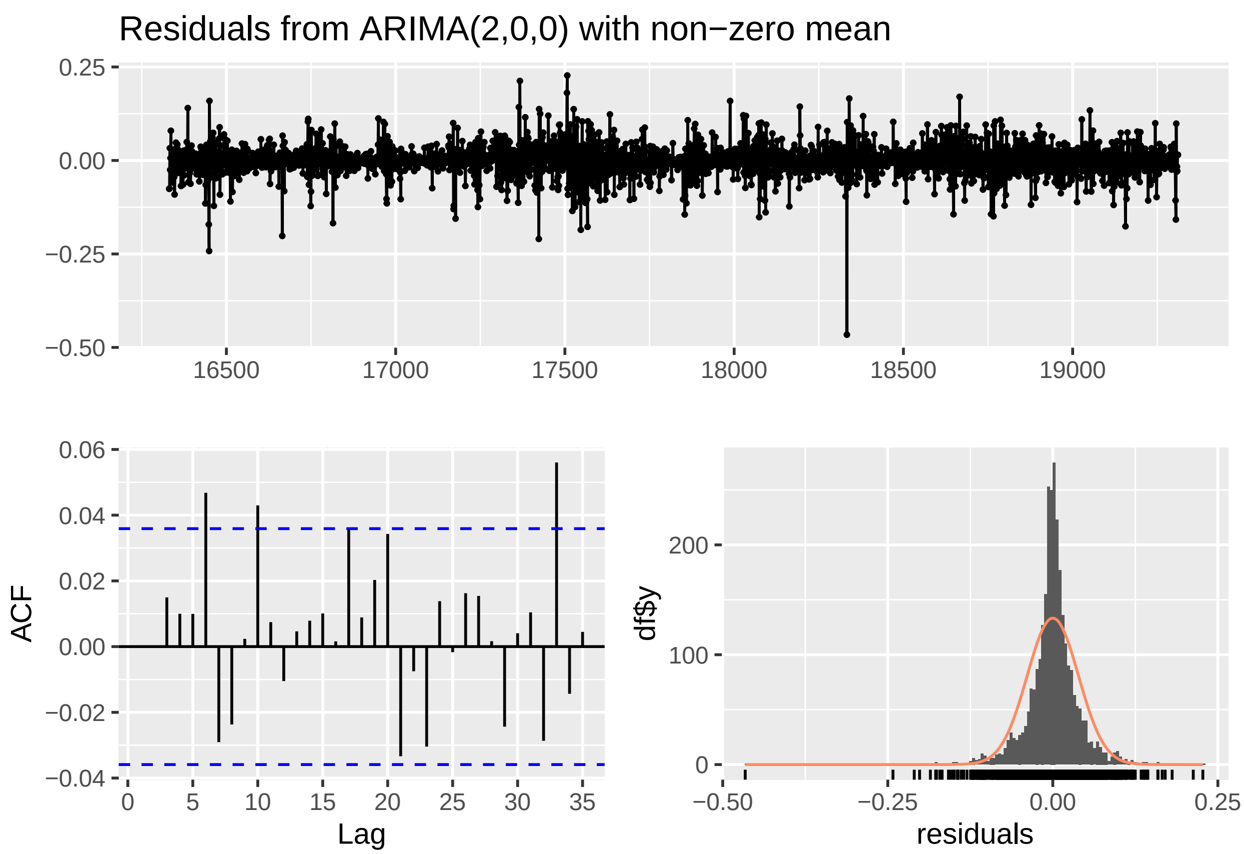
```
# Calculate accuracy metrics for ARIMA
arima_accuracy <- accuracy(arima_forecast, test_data)

# Display the accuracy metrics
print(arima_accuracy)
```

```
##                   ME       RMSE       MAE      MPE     MAPE        ACF1
## Test set 0.001126065 0.02497928 0.0173945 140.9899 174.1597 -0.02677828
##          Theil's U
## Test set  1.046421
```

The accuracy metrics for the ARIMA model reveal its performance characteristics. The small Mean Error (ME) of 0.00116 suggests a slight positive bias, while the Root Mean Squared Error (RMSE) of 0.02498 and Mean Absolute Error (MAE) of 0.01739 indicate relatively low forecast errors, with RMSE penalizing larger errors more heavily. The unusually high Mean Percentage Error (MPE) and Mean Absolute Percentage Error (MAPE) values (140.80% and 173.84%) are likely inflated by near-zero actual values in the test set. The minimal Autocorrelation of Residuals (ACF1) at -0.02708 is a positive sign, indicating no significant correlation in residuals. However, the Theil's U statistic of 1.046 suggests the model's performance is only slightly better than a naïve forecast.

Examining residuals is a critical step in time series analysis to validate model assumptions and ensure its adequacy. Residuals should ideally resemble white noise, meaning they are uncorrelated, normally distributed, and have constant variance over time. Any patterns or significant autocorrelation in residuals could indicate that the model fails to capture important structures in the data. To further assess the ARIMA model's validity, we will now evaluate its residuals through diagnostic plots and statistical tests.

```
# Check residual diagnostics for the ARIMA model
checkresiduals(arima_model)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(2,0,0) with non-zero mean
## Q* = 17.559, df = 8, p-value = 0.02479
##
## Model df: 2.   Total lags used: 10
```

Based on the Ljung-Box test, since the p-value (0.02639) is less than the 0.05 significance level, we reject the null hypothesis. This indicates that the residuals exhibit some level of autocorrelation and are not purely white noise. This could mean that the ARIMA(2,0,0) model fails to capture all patterns in the data, suggesting room for improvement.

From the Histogram of Residuals (Bottom Right), we observed a slight deviation from normality, with a sharp peak at the center and slightly heavier tails. This indicates that the model may struggle with extreme observations or volatility in the data.

After evaluating the ARIMA model, we will next explore the ETS (Error, Trend, Seasonality) model, which is well-suited for capturing exponential smoothing patterns in time series data. Unlike ARIMA, the ETS model explicitly handles trend and seasonality components. This makes it particularly effective for modeling data with clear level shifts or seasonal patterns, providing an alternative perspective on Bitcoin returns.

ETS (Reagan)

Holt-Winters (Reagan)

NNETAR (Jiaxuan)

Prophet (Jiaxuan)

Combination (Marc)

Conclusion & Future Works

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) The New S Language. Wadsworth & Brooks/Cole.