Question 4:

Stock Market Prediction is the act of trying to determine the future value of company stock or other financial instrument traded on an exchange. Many methods can achieve this prediction, here we will apply two types of time series analysis which are Long Time Seris Memory (LSTM) and ARIMA models.

We will use the *AXISBANK.csv* dataset which contains AXISBANK India bank data spans from 1st January 2000 to 30th April 2021 in the Indian market.

Loading Data:

Loading data by command:

data=pandas.read_csv("AXISBANK.csv")

This dataset has shape (5306,15) columns but for predicting the future stock, by command :

data.columns

Output:

Index(['Date', 'Symbol', 'Series', 'Prev Close', 'Open', 'High', 'Low', 'Last', 'Close', 'VWAP', 'Volume', 'Turnover', 'Trades', 'Deliverable Volume','%Deliverble']):

For predicting the future stocks the date column will be the main index that all data will be referred to, and the close column will be the output label. The close value in the stock market depends on some vectors such as previously closed value, open value, high and low values, and the last value. Therefore, the output label just depends on 'Prev Close', 'Open', 'High', 'Low' and 'Last' columns, which means the other columns will be removed.

To achieve this for making date column is index column by this command:

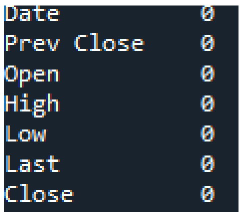
data.set_index("Date", drop=False, inplace=True)

and for just taking the important columns by using this command:

data=data.loc[:,['Date', 'Prev Close', 'Open', 'High', 'Low', 'Last', 'Close']]

Checking if there are any NaN values in the dataset by using this command:

data2.isnull().sum()
the output as shown in Figure 1:



```
Date          0
Prev Close    0
Open          0
High          0
Low           0
Last          0
Close         0
```

Figure 1: Finding NaN values

Creating train and test datasets:

1) **Splitting dataset:** For training and testing data, 75% of data used training data and 25% as testing data.

   For training data we took first 75% *'Date'*, *'Close'* columns of dataset as shown in command below:

   ```
   train_data = data2[:int(.75*len(data2))][['Date', 'Close']]
   ```

   For testing data we took the rest of dataset as shown in command below:

   ```
   train_test = data2[int(.75*len(data2)):][['Date', 'Close']]
   ```

2) **Scaling datasets:**
   For preserving the shape of dataset distribution we should scale data between (0,1)

   By creating *MinMaxScaler* scaler as shown in command line below:

   ```
   scaler = MinMaxScaler(feature_range=(0, 1))
   ```

   The train and test datasets are still in the type of DataFrame, before fitting them through created scaler we have to convert them to NumPy array type and reshape them as 1 dimension as shown in commands below:

   For training dataset:

   ```
   train_data_scaled = scaler.fit_transform(numpy.array(train_data['Close']).reshape(-1,1))
   ```

   For testing dataset:

   ```
   train_test_scaled = scaler.fit_transform(numpy.array(train_test['Close']).reshape(-1,1))
   ```

3) Re-shaping datasets:
   we converted the train and test data into a 2D array with x_train samples, 9 timestamps, and one feature at each step as shown in the function below:

   ```
   def get_x_y(data, N, offset):
       x, y = [], []
       for i in range(offset, len(data)):
           x.append(data[i-N:i])
           y.append(data[i])
       x = numpy.array(x)
       y = numpy.array(y)
       return x, y
   ```

Where data presents the train or test data, N is timestamping.

For training and test data final preparing we used the command below:

```
x_train, y_train = get_x_y(train_data_scaled, 9, 9)
x_test, y_test = get_x_y(train_test_scaled, 9, 9)
```

The final trainig data shapes: x_train = (3970, 9, 1) and y_train = (3970, 1)

,and for test data shapes :x_test = (1318, 9, 1) and y_test = (1318, 1).

LSTM Model:

Training Model:

For the training model, we have to create a sequential function that allows us to create models layer-by-layer. Creating sequential   model can be created by the following command:

```
model = Sequential()
```

This model main components are:

1) Input Layer:
   we used LSTM layer with the following arguments: 60 units is the dimensionality of the output space, return_sequences=True is necessary for stacking LSTM layers so the consequent LSTM layer has a two-dimensional sequence input, and input_shape is the shape of the training dataset, as shown below:

```
model.add(LSTM(units=60, return_sequences=True, input_shape=(x_train.shape[1],1)))
```

   Specifying 0.3 in the Dropout layer means that 30% of the layers will be dropped, as shown below:

```
model.add(Dropout(0.3))
```

2) Hidden Layer:
   For hideden layer we used 20 LSTM units with 30% dropout as shown in command lines :

```
model.add(LSTM(units=20))
model.add(Dropout(0.3))
```

3) Output Layer:

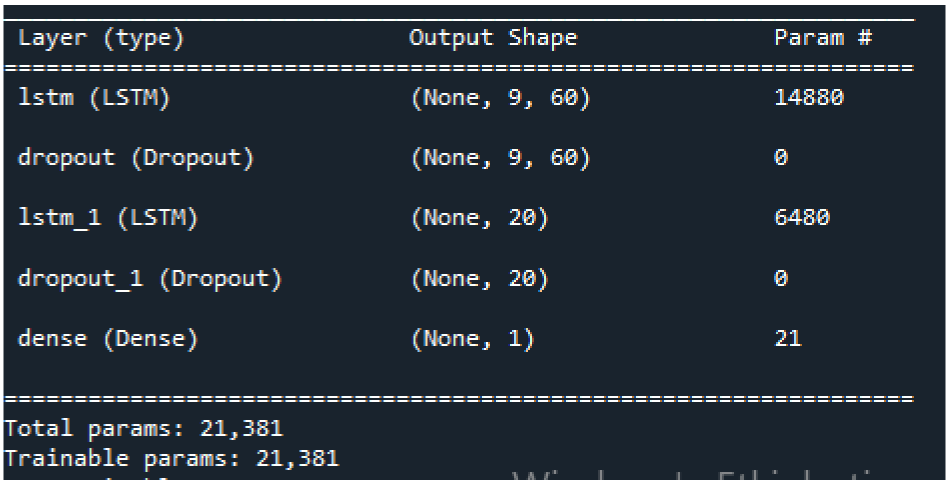
Because we trained the model in one feature, therefore, the output layer will just be one dense layer, as shown in the command below:

model.add(Dense(1))

To compile our model we use the *Adam* optimizer and set the loss as the *mean_squared_error*. After that, we fit the model to run for 30 epochs (the epochs are the number of times the learning algorithm will work through the entire training set) with a batch size of 1.

The model summary can be summarized by the output of the command below:

Model.summary()



```
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 9, 60)             14880

 dropout (Dropout)           (None, 9, 60)             0

 lstm_1 (LSTM)               (None, 20)                6480

 dropout_1 (Dropout)         (None, 20)                0

 dense (Dense)               (None, 1)                 21

=================================================================
Total params: 21,381
Trainable params: 21,381
```

Figure 2: LSTM model Summary.

For prediction the future stockes, we uses the *predict()* function for *x_test* data and the output Y_predicted values will roll back to their true values by using invers_transform() function as shown in command below:

Y_predicted= model.predict(x_test)

Y_predicted_inversed=scaler.inverse_transform(Y_predicted)

To know the real closing value, we have to rescale the output of predict function, because the input data scaled to (0,1) before the training process:

Y_true=scaler.inverse_transform(y_test)

The model predicting result will show in figure 3, where the blue line represents the true test dataset future values and the red line represent the data we predicted by the LSTM model:

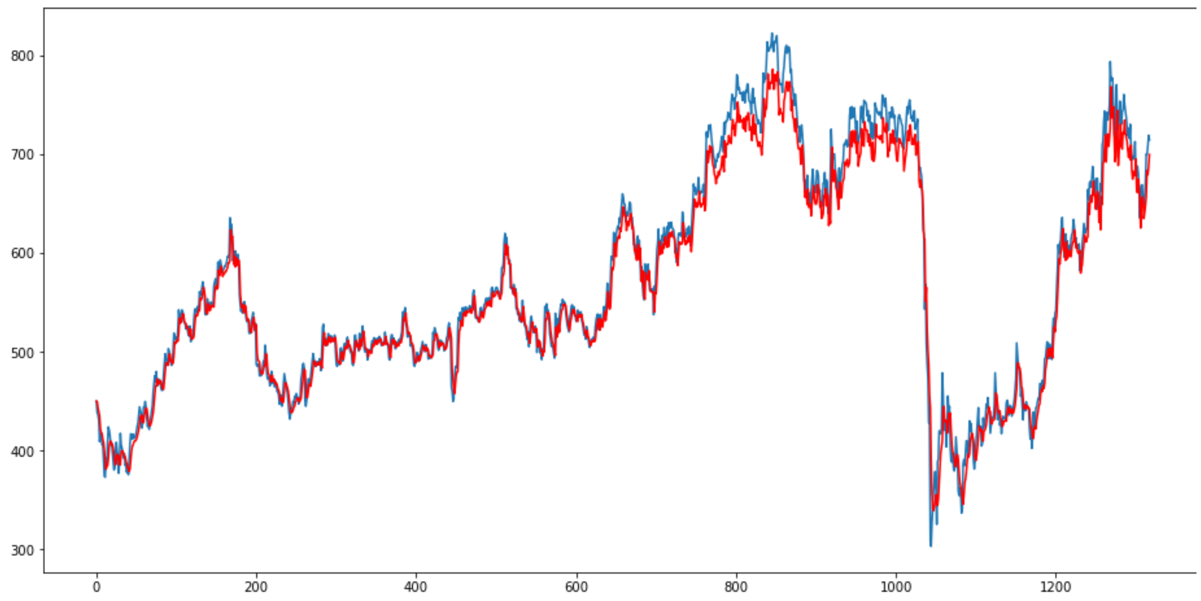


Figure 3: LSTM True Test and Predicted Values.

ARIMA Model:

In this model the input data will be that columns but before training, I will check the data available for training, the steps and explanation can be summarized as:

Loading data:

dataset = pd.read_csv('AXISBANK.csv')

dataset.shape

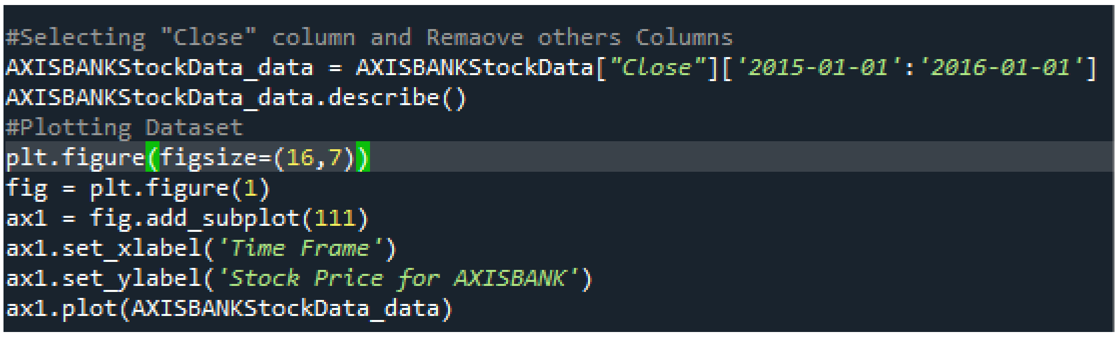
Data had been loaded with shape: ((5306, 15))

In the beginning, the date data will be reindexed to be the main index for the dataset, and for dropping NaN values. I used the command lines below to achieve it:

AXISBANKStockData = dataset.dropna()

AXISBANKStockData.index = pd.to_datetime(AXISBANKStockData.Date)

For better visualization and explanation and because the used dataset is too long, I selected the date starting with   2015-01-01 to 2016-01-01 for this data to be trained, at the same time, I removed unnecessary columns. This model will be trained according to close value and the date. The Command-line below will implement this and represent the dataset in Figure 4:



```
#Selecting "Close" column and Remaove others Columns
AXISBANKStockData_data = AXISBANKStockData["Close"]['2015-01-01':'2016-01-01']
AXISBANKStockData_data.describe()
#Plotting Dataset
plt.figure(figsize=(16,7))
fig = plt.figure(1)
ax1 = fig.add_subplot(111)
ax1.set_xlabel('Time Frame')
ax1.set_ylabel('Stock Price for AXISBANK')
ax1.plot(AXISBANKStockData_data)
```

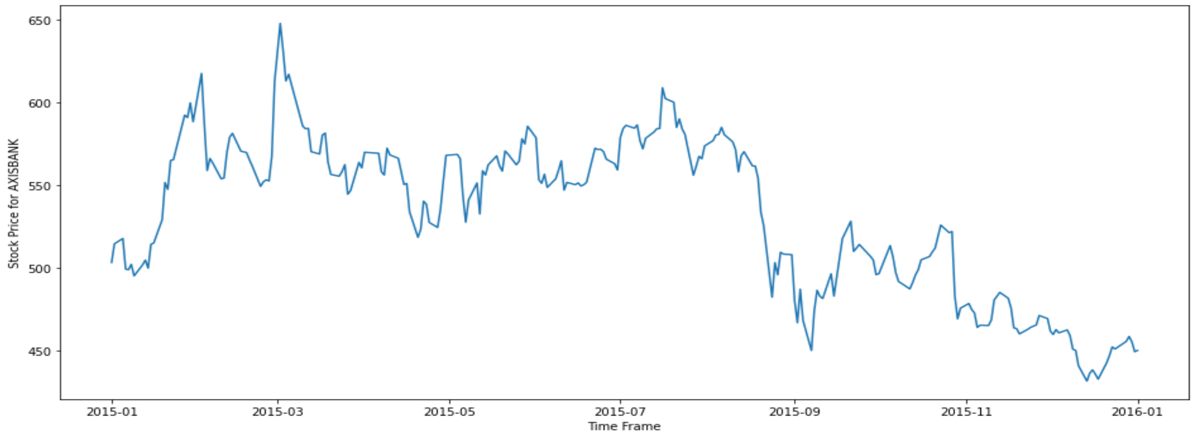
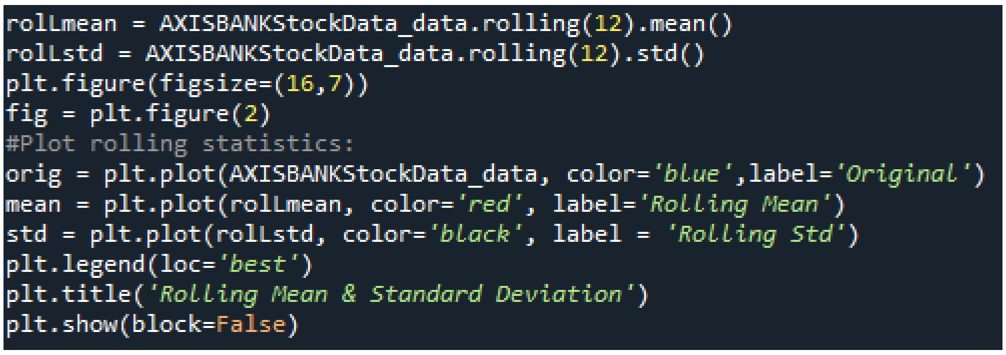


Figure 4: Dataset From 2015-01-01 to 2016-01-01

Before training, the dataset must be in a stationary state, which means the dataset must be at the same levels and doesn't change dramatically during the time. The Figure below shows the black line is the stationary levels of the dataset, it shows the data is at a stationary level. The command lines below, allowed me to Check the stationary status and display it.



```
rolLmean = AXISBANKStockData_data.rolling(12).mean()
rolLstd = AXISBANKStockData_data.rolling(12).std()
plt.figure(figsize=(16,7))
fig = plt.figure(2)
#Plot rolling statistics:
orig = plt.plot(AXISBANKStockData_data, color='blue',label='Original')
mean = plt.plot(rolLmean, color='red', label='Rolling Mean')
std = plt.plot(rolLstd, color='black', label = 'Rolling Std')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation')
plt.show(block=False)
```

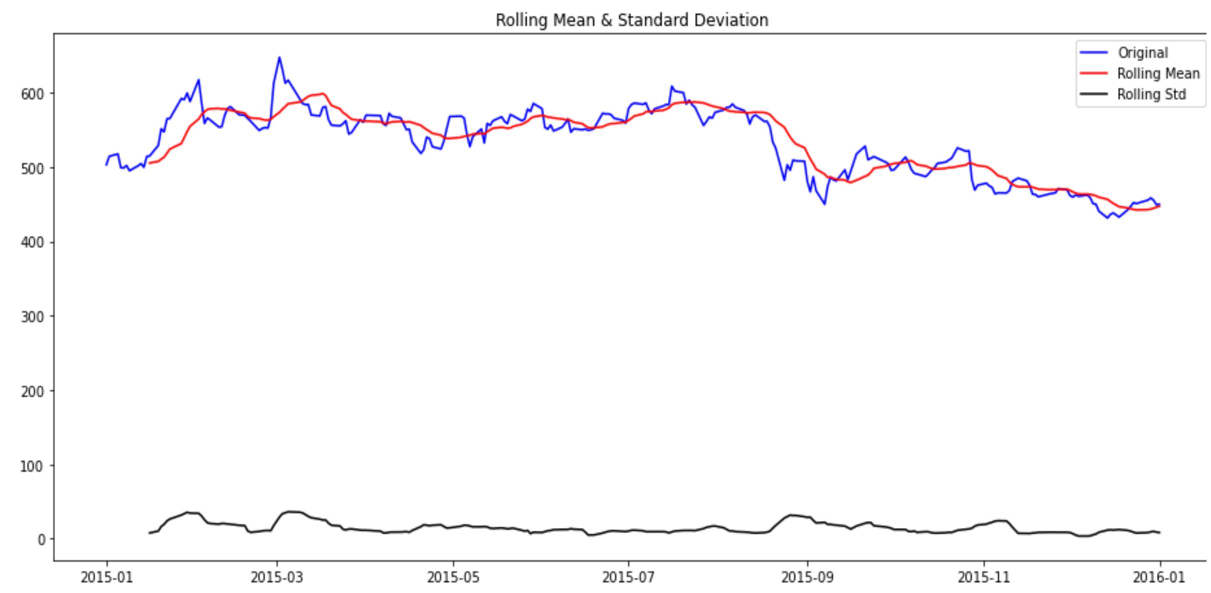
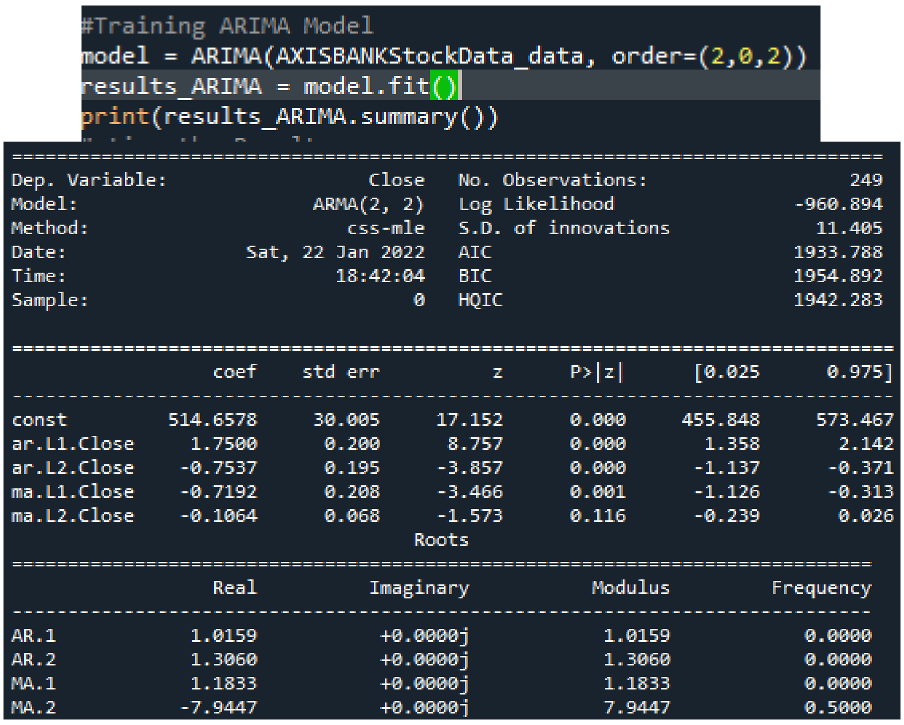


Figure 5: Dataset Rolling Statistics

After checking the serise stationery of the dataset, we will create an ARIMA model with order = (2,0,2).

The model creating, fitting, and summary can be achieved by commands lines and the output will be model summary in the figure.

Finally, model performance will be tested by predicting the data that the model trained by. The results are in figure 7.

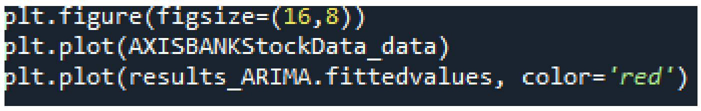


```
#Training ARIMA Model
model = ARIMA(AXISBANKStockData_data, order=(2,0,2))
results_ARIMA = model.fit()
print(results_ARIMA.summary())
```

```
==============================================================================
Dep. Variable:                  Close   No. Observations:                  249
Model:                     ARMA(2, 2)   Log Likelihood                -960.894
Method:                       css-mle   S.D. of innovations             11.405
Date:                Sat, 22 Jan 2022   AIC                           1933.788
Time:                        18:42:04   BIC                           1954.892
Sample:                             0   HQIC                          1942.283

===============================================================================
                  coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------
const         514.6578     30.005     17.152      0.000     455.848     573.467
ar.L1.Close     1.7500      0.200      8.757      0.000       1.358       2.142
ar.L2.Close    -0.7537      0.195     -3.857      0.000      -1.137      -0.371
ma.L1.Close    -0.7192      0.208     -3.466      0.001      -1.126      -0.313
ma.L2.Close    -0.1064      0.068     -1.573      0.116      -0.239       0.026
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0159           +0.0000j            1.0159            0.0000
AR.2            1.3060           +0.0000j            1.3060            0.0000
MA.1            1.1833           +0.0000j            1.1833            0.0000
MA.2           -7.9447           +0.0000j            7.9447            0.5000
```

Figure 6: ARIMA Model Summary



```
plt.figure(figsize=(16,8))
plt.plot(AXISBANKStockData_data)
plt.plot(results_ARIMA.fittedvalues, color='red')
```

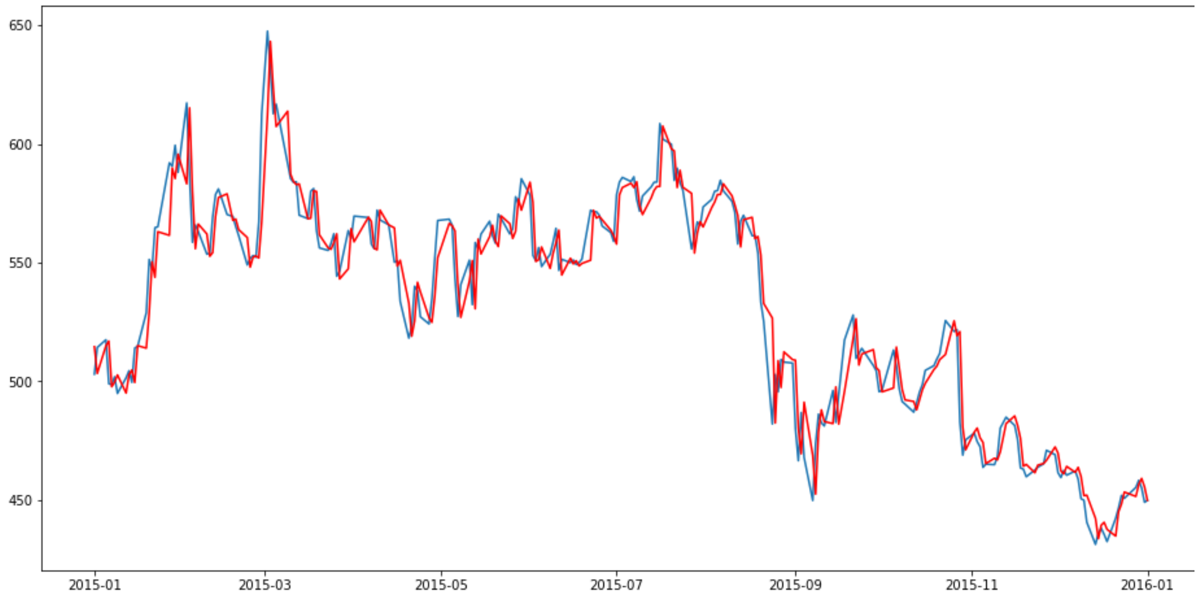


Figure 7: ARIMA Model Results