Stock Predictor

CMPE 188, Spring 2019, Section 01

Tiago Benko dos Anjos, [Akash Sindhu](https://sjsu.instructure.com/groups/282063/users/4254468), Lorena Silva

*Computer Engineering Department, San Jose State University*

tiago.benkodosanjos@sjsu.edu

akashsindhu96@gmail.com

lorena.silvaq@gmail.com

**Abstract**— **Stock prices are predicted to determine the future value of companies’ stock or other financial instruments that are marketed on financial exchanges. To predict stock movements, algorithmic traders and high-frequency traders are applying Machine Learning. Using fundamental and technical analysis, and applying the existing algorithms such as moving average, linear regression, random forest regression, auto aroma, prophet, LSTM, this paper focuses on the attempt to develop a program that predicts stock prices and suggest an investment options based on daily stock price of a company. We have collected raw data from scratch using alpha vantage API and performed an analysis to see which algorithm performs better and how much better by visualizing graphs and plots.**

# 1 Introduction

Anticipating how the financial exchange will perform is a standout amongst the most troublesome activities. There are such a large number of factors associated with forecast like physical factor and physiological factors, rational and irrational behavior, etc. Due to these factors, markets are very volatile and are very difficult to predict accurately. Can we use AI and machine learning algorithms to be game changer in this field? Utilizing highlights like the most recent declarations about an association, their quarterly income results, and so forth, AI systems can possibly uncover patterns and bits of knowledge we didn't see previously, and these can be utilized to make unerringly exact forecasts. In this project we will work with historical stock data of a publicly listed company and try to predict future stock price of the company using many machine learning algorithms like Random forest regressor, moving average, linear regression, Auto-Regressive Integrated Moving Averages., prophet and Long Short Term Memory networks. Same data was used for all of the algorithms and calculated the RMSE and visualize the actual and predicted through the graphs. Daily raw data from scratch was collected for around 3000 companies. Alpha vantage API which is available to public for free. For the variety, we used GitHub API to get the sentiment analysis of the last 100 tweets. The remainder of this paper is structured as follows: In section 2 we will cover the data collection and data preparation, tools we used, version control. Section 3: we will cover different algorithms we used and their methodology, Section 4: we will discuss the results and visualize the results. Section 5: concludes the project and section 6: future prospects of the project.

# 2 Data collection

To ensure quality assurance and integrity of this research, collecting the most accurate data was essential for this project. In order to collect accurate historical stock data, Alpha Vantage was used. Alpha Vantage provides global real time and historical data for daily, weekly, monthly and intraday. For this particular stock market prediction, 2941 companies were collected dated from 2019-04-12 to the Initial Public Offering (IPO) date of the company. In order to automate the process, we created a python script that utilizes Alpha Vantage’s API to extract daily historical data for each company. Unfortunately, Alpha Vantage’s free tier API allows only 500 API calls per day/account. In order to overcome this obstacle, all the team members created an Alpha Vantage account with 500 accesses per day. The python script responsible for extracting stock data can handle multiple API accounts in order to speed the data collection (see Appendix A). The source code capable of handling multiple accounts was developed entirely from scratch and is publicly available at <https://github.com/tiagobka/HistoricalStockDataExtraction>. To be able to make sense of the obtained data, sentiment analysis was used as the main tool. In order to do this, a system was built to help extract attributes to the expression, in this case, Twitter’s sentiment analysis of the last 100

tweets where chosen to see if the news about the company were positive, negative or neutral to get a more accurate intuition before buying stock.

In the figure below, we can see our data composition that we collected from the alpha vantage API. In this project, the data downloaded did not had any missing values, noise, null values so we did not have to spend time replacing or predicting values.

|  |  |  |
| --- | --- | --- |
| **Category** | **Number** | **Percent** |
| **Basic Industries** | 63 | 2.14% |
| **Capital Goods** | 142 | 4.83% |
| **Consumer Durables** | 77 | 2.62% |
| **Consumer Non-Durables** | 89 | 3.03% |
| **Consumer Services** | 311 | 10.57% |
| **Energy** | 53 | 1.8% |
| **Finance** | 575 | 19.55% |
| **Health Care** | 632 | 21.49% |
| **Miscellaneous** | 534 | 18.16% |
| **Public Utilities** | 50 | 1.7% |
| **Technology** | 363 | 12.34% |
| **Transportation** | 52 | 1.77% |
| **Total** | 2941 |  |

**Figure**: Data Composition

# 3 Fundamental and Technical Analysis

Stock market analysis is divided into two parts – Fundamental Analysis and Technical Analysis.

## 3.1 FUNDAMENTAL ANALYSIS

For stocks and equity instruments, major investigation utilizes revenues, income, future development, return on value, net revenues, and other information to decide an organization's hidden esteem and potential for future development.

## 3.2 TECHNICAL ANALYSIS

It involves reading graphs and technical plots to understand and predict trends in stock markets.

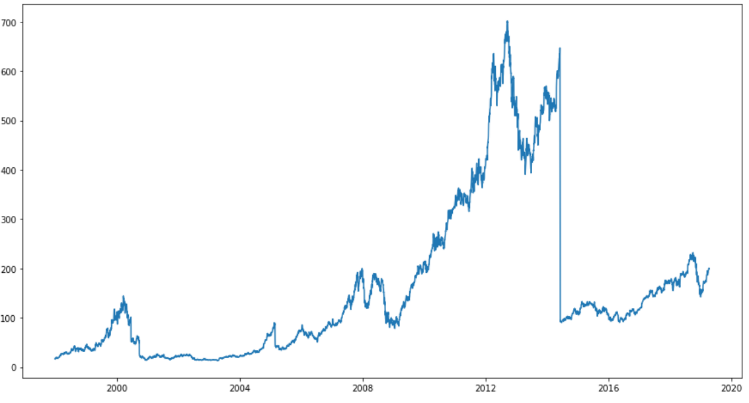
# 4 Algorithms

The dataset we used for our project is of Apple from the starting date of company to the 2019-04-12. We will focus on technical analysis in this project now onwards.

there are multiple columns in our dataset, date, open, high, low, close, volume.

* The columns *Open* and *Close* represent the starting and final price at which the stock is traded on a particular day.
* *High*, and *Low* represent the maximum, and minimum number of shares for the day.
* *Volume*is the amount of a security that was traded during a given period of time.

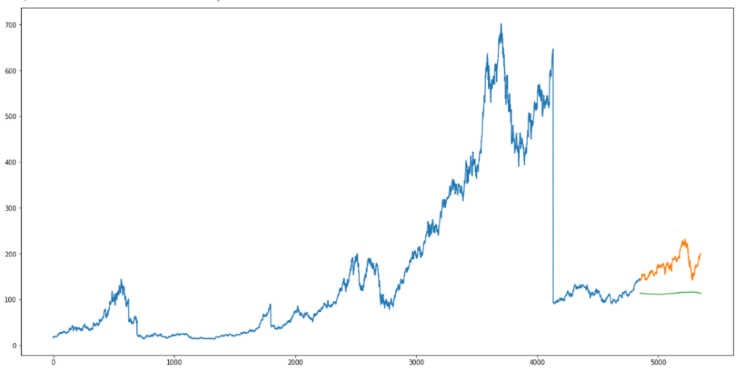
We will consider closing price as the target variable as the profit and loss is determined by this only. Other are not useful for us.

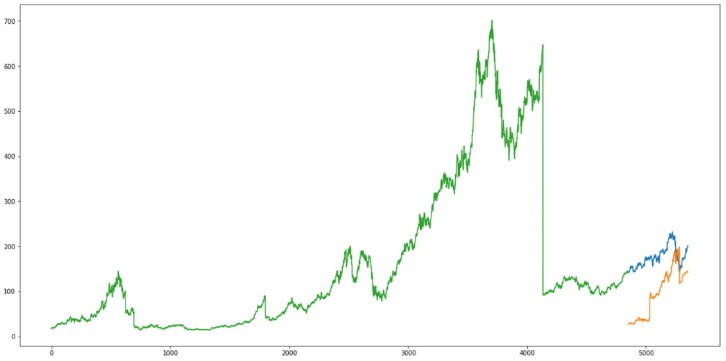
Let’s plot the target variable to see how our data looks like. From the figure 1 we can see that the stock price increase till 2014 and then sudden decreases and again increases.

**Figure 1**: Overall representation of our data

## 4.1 MOVING AVERAGE

Average is something with which we all are familiar. We use average in day to day life in from calculating average temperature, average speed, average grades so this is the good starting point. Instead of average, moving average is used here. In this technique, latest set of values for each prediction is used which means once the value is predicted, the oldest value is not taken into consideration, instead of that the whole set of values will move one step considering the most recent predicted value.



**Figure 2:** Moving average plot of predicted values along with actual values.

## 4.2 LINEAR REGRESSION

Linear regression returns the equation that determines the relationship between the independent and dependent variables.

/var/folders/j6/mk0t4_jj53n2wljx82kgkzmc0000gn/T/com.microsoft.Word/WebArchiveCopyPasteTempFiles/A8+2ZuG0IU6gAAAAAElFTkSuQmCC

Where Y = Dependent variable (DV)

X1, X2, Xn = Independent variable (IV)

b0 = intercept

b1, b2 = coefficients

N = No. of observations

In our project we don’t have independent variables. So, we considered time as the independent variable and did feature extraction to extract features like day, month, year, mon/fri etc. and then fit the model.  We used fastai.structured library to import add\_datepart which extract the features from date.

A close up of a map

Description automatically generated

**Figure 3:** Linear regression plot of predicted values along with actual values.

## 4.3 RANDOM FOREST

Ensemble methods have proven to be very successful on predicting the direction of stocks but in this project, we will test random forest regressor to predict accurate prices. We think that it will perform better than other algorithms like moving average, linear regression and AUTO ARIMA.

In the significant paper we discussed about the ensemble methods predicted much better than single classifier models when predicting the direction of stocks which we can see in our project.

**Figure 4:** Random forest regressor plot of

predicted values along with actual values.

## 4.4 AUTO ARIMA

Auto regression integrated moving average. It is one of the most popular time series forecasting algorithm. It is widely used to predict the direction of time series data. Model works on following assumptions –

1. The data series is stationary, which means that the mean and variance should not vary with time. A series can be made stationary by using log transformation or differencing the series.

2. The data provided as input must be a univariate series, since arima uses the past values to predict the future values.

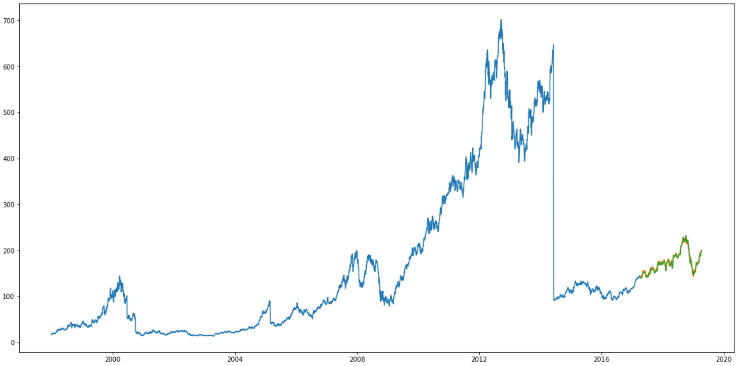
It has three components-

AR = Auto Regression

I = differencing term

MA = Moving average

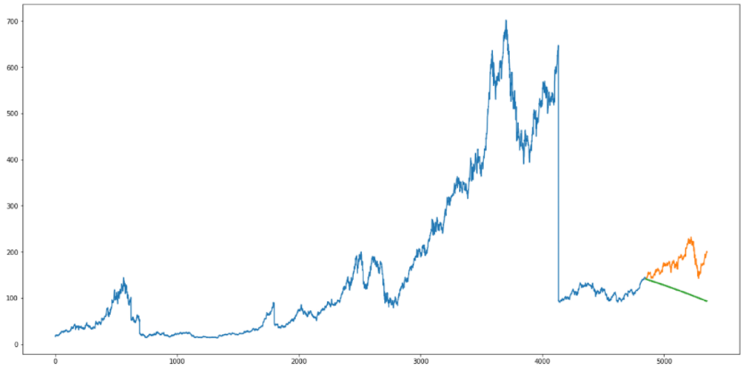
AR term refers to the past values used for forecasting the next value. The AR term is defined by the parameter ‘p’ in arima. The value of ‘p’ is determined using the PACF plot.



MA term is used to defines number of past forecast errors used to predict the future values. The parameter ‘q’ in arima represents the MA term. ACF plot is used to identify the correct ‘q’ value.

We have used auto arima, which means it will choose parameters automatically that will lead to least errors.

Order of differencing specifies the number of times the differencing operation is performed on series to make it stationary. Test like ADF and KPSS can be used to determine whether the series is stationary and help in identifying the d value.



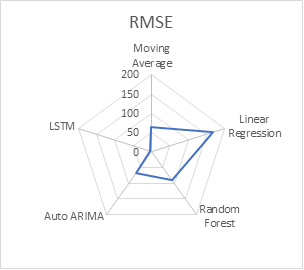
**Figure 5:**AUTO ARIMAplot of predicted values along with actual values.

## 4.5 LONG SHOR-TERM MEMORY

Long Short**-**Term Memory is a very efficient algorithm for sequence prediction.  Hence, it is very adequate to use it for finding patterns in stock market data.  LSTM algorithms have been proven to be more efficient than conventional Feed Forward Neural Networks (FFNN) and Recurrent Neural Networks (RNN) because LSTM have the ability to selectively remember patterns. Just like Neural Networks LSTM is composed of multiple cells. However, each cell in the LSTM is composed on three gates. The first gate is the Input gate, which controls whether the memory cell get updated. The forget gate controls if the memory cell should be reset. And the output gate controls if the content of the current cell should be visible. LSTM cells were specifically designed to tackle vanishing or exploding gradients observed in deep neural networks. To briefly explain, vanishing/ exploding gradients is a phenomenon that occurs when computing derivatives during the training of deep neural networks which leads to the loss of input information.

**Figure 6:** LSTM plot of predicted values along with actual values.

# 5 Results



**Figure 7**: RMSE comparison of all the models.

While splitting the training and test data, we cannot split randomly because it will break the time component. In time series, time is one of the most important aspect and we cannot split time randomly as it will destroy the property of time and model can be prone to overfitting. So, we choose to select last (most recent) two years data into validation and the rest of data into training.

Figure 7 shows that RMSE values of LSTM is the closest to 0 which means it should perform best. Random forest also performed better than the linear regression, which is actually true as ensemble methods should perform better than single classifier models, we saw in our significant paper. AUTO ARIMA and moving average have better RMSE values than Random forest, we cannot say that they performed better than random forest without seeing the plot against actual close values and predicted close values. In figure 2, 5 we can see that moving average and AUTO ARIMA did not predicted direction of stocks close and accurate to actual values as the random forest in Figure 4 had. In figure 5, we can see that AUTO ARIMA doesn’t focus on seasonal part, but it focused more on trends in the series.

In the Figure 6, we can see that after plotting the actual and predicted values against each other, LSTM performed very close to the actual close values.

The RMSE values we got for Moving average is 65.4649, Linear Regression: 168.21809, Random Forest: 90.25008​, AUTO ARIMA: 66.180755​, LSTM: 9.15621372.

# Conclusion

This project compared Moving average, linear regression, random forest, AUTO ARIMA and LSTM in predicting stock market direction and absolute prices. We were able to predict all for all the models and able to represent actual and predicted values in plots. Predicting the direction of stock prices is enough for most of the time for companies and individuals to get good intuition about the future stocks. This can be beneficial for short term prediction and money-making strategy.

# Future improvements and prospects

Future improvements for this project are very technical as there are a lot of things need to be done to make this project more accurate, easy to use, automate and link with Robinhood app for real time data analysis. We need to work more on sentimental analysis of news and have to figure out how can we use different websites to gather news and include that in our system. To link this with Robinhood app for real time analysis, we need to figure out how can we pass daily data into our model from the app. This project can be used for personal profits.

**References**

[1] https://github.com/rhnvrm/labeled-tweet-generator​

[2] https://twitter-sentiment-csv.herokuapp.com/​

[3] https://www.analyticsvidhya.com/blog/2018/1-0/predicting-stock-price-machine-learningnd-deep-learning-techniques-python/#comment-156405​

**APENDIX**

# -\*- coding: utf-8 -\*-

"""stock\_market\_prediction.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1r4QMfxHn\_8sNOKgIzmhAfJ0KiuGWdlqg

"""

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

#setting figure size

from matplotlib.pylab import rcParams

rcParams['figure.figsize'] = 20,10

# #for normalizing the data

# from sklearn.preprocessing import MinMaxScaler

# scaler = MinMaxScaler(feature\_range=(0,1))

# read the file

from google.colab import files

uploaded = files.upload()

import io

df = pd.read\_csv(io.BytesIO(uploaded['daily\_AAPL.csv']))

# Dataset is now stored in a Pandas Dataframe

df.head()

import datetime

df['Date'] = pd.to\_datetime(df.Date)

# df.Date = df['Date'].dt.strftime('%m-%d-%Y')

# setting index as date

# df['Date'] = pd.to\_datetime(df.Date, format='%Y-%m-%d')

df.index = df.Date

df.head()

# plot

plt.figure(figsize=(15,8))

plt.plot(df['Close'], label='Close Price History')

"""# Moving Average"""

# create dataframe with date and close

data = df.sort\_index(ascending = True, axis = 0)

new\_data = pd.DataFrame(index=range(0, len(df)), columns=['Date', 'Close'])

new\_data

for i in range(0, len(data)):

new\_data['Date'][i] = data['Date'][i]

new\_data['Close'][i] = data['Close'][i]

new\_data

# splitting the data into train and validation

train = new\_data[:4851]

valid = new\_data[4851:]

new\_data.shape, train.shape, valid.shape

print(str(train['Date'].min()) + '\n', str(train['Date'].max()) + '\n', str(valid.Date.min()) + '\n' + str(valid.Date.max()))

train['Date'].min(), train['Date'].max(), valid['Date'].min(), valid['Date'].max()

# create predictions for the validation set and check RMSE using actual scores

# make predictions

preds = []

for i in range(0, 503):

a = train['Close'][len(train) - 503 + i:].sum() + sum(preds)

b = a/503

preds.append(b)

# calculate rmse

rms = np.sqrt(np.mean(np.power((np.array(valid.Close) - preds), 2)))

rms

"""### Just checking the RMSE does not help us in understanding how the model performed. Let’s visualize this to get a more intuitive understanding. So here is a plot of the predicted values along with the actual values."""

# plot

valid['predictions'] = 0

valid['predictions'] = preds

plt.plot(train['Close'])

plt.plot(valid[['Close', 'predictions']])

"""### The RMSE value is 66 but the results are not very close to what we want, still we need to predict much better. Let's try some other method.

## Linear Regression

### For our problem statement, we do not have a set of independent variables. We have only the dates instead. Let us use the date column to extract features like – day, month, year, mon/fri etc. and then fit a linear regression model.

"""

# # setting index as date values

# df['Date'] = pd.to\_datetime(df.Date, format="%Y-%m-%d")

# df.index = df.Date

df

new\_data1 = pd.DataFrame(index=range(0, len(df)), columns=['Date', 'Close'])

for i in range(0,len(data)):

new\_data1['Date'][i] = data['Date'][i]

new\_data1['Close'][i] = data['Close'][i]

new\_data1

#create features

!pip install fastai==0.7.0

from fastai.structured import add\_datepart

add\_datepart(new\_data1, 'Date')

new\_data1.drop('Elapsed', axis=1, inplace=True) #elapsed will be the time stamp

new\_data1['mon\_fri'] = 0

for i in range(0,len(new\_data1)):

if (new\_data1['Dayofweek'][i] == 0 or new\_data1['Dayofweek'][i] == 4):

new\_data1['mon\_fri'][i] = 1

else:

new\_data1['mon\_fri'][i] = 0

# splitting the data into train and validation

train1 = new\_data1[:4851]

valid1 = new\_data1[4851:]

x\_train = train1.drop('Close', axis=1)

y\_train = train1['Close']

x\_valid = valid1.drop('Close', axis=1)

y\_valid = valid1['Close']

#implement linear regression

from sklearn.linear\_model import LinearRegression

model = LinearRegression()

model.fit(x\_train,y\_train)

#make predictions and find the rmse

preds = model.predict(x\_valid)

rms=np.sqrt(np.mean(np.power((np.array(y\_valid)-np.array(preds)),2)))

rms

"""### We can see that rmse of linear regression is higher than moving average, let's see the graph to understand better."""

#plot

valid1['Predictions'] = 0

valid1['Predictions'] = preds

# valid.index = new\_data1[1848:].index

# train.index = new\_data1[:1848].index

plt.plot(train1['Close'])

plt.plot(valid1[['Close', 'Predictions']])

"""### Logistic regression performed even wose than moving average, so let's try other method."""

new\_data1.head()

x\_train

x\_valid

"""## Random Forest"""

from sklearn.ensemble import RandomForestRegressor

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature\_range=(0,1))

# scaling data

x\_train\_scaled = scaler.fit\_transform(x\_train)

x\_train = pd.DataFrame(x\_train\_scaled)

x\_valid\_scaled = scaler.fit\_transform(x\_valid)

x\_valid = pd.DataFrame(x\_valid\_scaled)

regressor = RandomForestRegressor()

regressor.fit(x\_train, y\_train)

preds = regressor.predict(x\_valid)

#rmse

rms=np.sqrt(np.mean(np.power((np.array(y\_valid)-np.array(preds)),2)))

rms

# plot

valid1['predictions'] = 0

valid1['predictions'] = preds

plt.plot(valid1[['Close', 'predictions']])

plt.plot(train1['Close'])

"""### We can see that we didn't get good RMSE score and even we didn't predict good results in graph too. Let's try other method.

## Auto ARIMA

### ARMIA takes into account three parameters

### 1. p (past values used for forecasting the next value.)

### 2. q (past forecast errors used to predict the future values.)

### 3. d (order of differencing)

### Parameter tuning takes a lot of time so we selected AUTO ARIMA which will automatically select the best parameters (p,q,d) that provides the least errors.

"""

!pip install pyramid-arima

from pyramid.arima import auto\_arima

training = train['Close']

validation = valid['Close']

model = auto\_arima(training, start\_p=1, start\_q=1,max\_p=3, max\_q=3, m=12,start\_P=0,

seasonal=True,d=1, D=1, trace=True,error\_action='ignore',suppress\_warnings=True)

model.fit(training)

forecast = model.predict(n\_periods=503)

forecast = pd.DataFrame(forecast,index = valid.index,columns=['Prediction'])

rms=np.sqrt(np.mean(np.power((np.array(valid['Close'])-np.array(forecast['Prediction'])),2)))

rms

#plot

plt.plot(train['Close'])

plt.plot(valid['Close'])

plt.plot(forecast['Prediction'])

"""### As its evident from the plot, the model has captured a trend in the series, but does not focus on the seasonal part. In the next section, we will implement a time series model that takes both trend and seasonality of a series into account.

## LSTM (Long Short Term Memory)

"""

!pip install keras

#importing required libraries

from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential

from keras.layers import Dense, Dropout, LSTM

#creating dataframe

data = df.sort\_index(ascending=True, axis=0)

new\_data = pd.DataFrame(index=range(0,len(df)),columns=['Date', 'Close'])

for i in range(0,len(data)):

new\_data['Date'][i] = data['Date'][i]

new\_data['Close'][i] = data['Close'][i]

#setting index

new\_data.index = new\_data.Date

new\_data.drop('Date', axis=1, inplace=True)

#creating train and test sets

dataset = new\_data.values

train = dataset[0:4851,:]

valid = dataset[4851:,:]

#converting dataset into x\_train and y\_train

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_d = scaler.fit\_transform(dataset)

x\_train, y\_train = [], []

for i in range(100,len(train)):

x\_train.append(scaled\_d[i-100:i,0])

y\_train.append(scaled\_d[i,0])

x\_train, y\_train = np.array(x\_train), np.array(y\_train)

x\_train = np.reshape(x\_train, (x\_train.shape[0],x\_train.shape[1],1))

# create and fit the LSTM network

model = Sequential()

model.add(LSTM(units=50, return\_sequences=True, input\_shape=(x\_train.shape[1],1)))

model.add(LSTM(units=50))

model.add(Dense(1))

model.compile(loss='mean\_squared\_error', optimizer='adam')

model.fit(x\_train, y\_train, epochs=1, batch\_size=1, verbose=2)

#using past 60 values from the training data

inputs = new\_data[len(new\_data) - len(valid) - 100:].values

inputs = inputs.reshape(-1,1)

inputs = scaler.transform(inputs)

X\_test = []

for i in range(100,inputs.shape[0]):

X\_test.append(inputs[i-100:i,0])

X\_test = np.array(X\_test)

X\_test = np.reshape(X\_test, (X\_test.shape[0],X\_test.shape[1],1))

closing\_price = model.predict(X\_test)

closing\_price = scaler.inverse\_transform(closing\_price)

rms=np.sqrt(np.mean(np.power((valid-closing\_price),2)))

rms

#for plotting

train = new\_data[:4851]

valid = new\_data[4851:]

valid['Predictions'] = closing\_price

plt.plot(train['Close'])

plt.plot(valid[['Close','Predictions']])