Stock Analysis and Trading Strategy Model

Final Project,
Fall Semester'20

Team Members :

Akash M Dubey
Nikhil Kashid
Purvang J Thakkar
Mayank Mehta
Siddhi Prabhu
Indupriya Kompi S
Shwetank

Under Supervision :
Professor Yizhen Zhao

Teaching Assistant :
Xinyue (Shirley) Cai

BACKGROUND & CONTEXT

BACKGROUND OF AMAZON STOCK

Amazon.com, Inc. engages in the provision of online retail shopping services.
It operates through the following business segments: North America, International, and Amazon Web Services (AWS).

Historical daily share price chart and data for Amazon since 1997 adjusted for splits. The latest closing stock price for Amazon as of December 08, 2020 is **3177.29**.

- The all-time high Amazon stock closing price was **3531.45** on **September 02, 2020**.
- The Amazon 52-week high stock price is **3552.25**, which is **11.8%** above the current share price.
- The Amazon 52-week low stock price is **1626.03**, which is **48.8%** below the current share price.
- The average Amazon stock price for the last 52 weeks is **2601.89**.

AMAZON STOCK DURING COVID-19

- **Amazon’s market capitalization is about 570$ Billion so far in 2020, and the company is worth 1.49$ Trillion trailing just behind Microsoft and Apple**
- **Amazon’s stock has increased by more than 70% this year and now trading at around $3,000 per share**
- **If you invested $5,000 in Amazon shares at the start of the year in 2020, you'd have $8,400 of the stock in your portfolio today.**

OBJECTIVES

•To assess the price dynamics of Amazon stock over the period of 6 years(2014-2019) using visualizations

•To study various candidate models and algorithms to get the Error Metrics (R-Squared and RMSE)

•Develop a trading strategy to make Amazon a valuable offering

DATA PRE PROCESSING FLOW

- Get the Data
- Data Preprocessing – Cleaning, Modification
- Exploratory Data Analysis
- Decide on the Candidate Models
- Design an Algorithm
- Performance Evaluation- RMSE and R-square
- Back – testing - Sharpe ratio
- Comparison of models
- Trading Strategy – buy or sell the stock at the right time

DATASET DESCRIPTION

- The initial dataset has 1508 rows from 01/01/2014 to 12/31/2019.
- Following is the description of the columns:
- Date: shows the date of the trading day.
- Open: Opening price of AMZN on that trading day.
- High: Highest price of AMZN on that trading day.
- Low: Lowest price of AMZN on that trading day.
- Close: Closing price of AMZN at the end of the trading day.
- Adj Close: It is the closing price after adjustments for all applicable splits and dividend distributions.
- Volume: The total number of shares traded on the trading day.
- Apart from this Fama-French data has also been used.

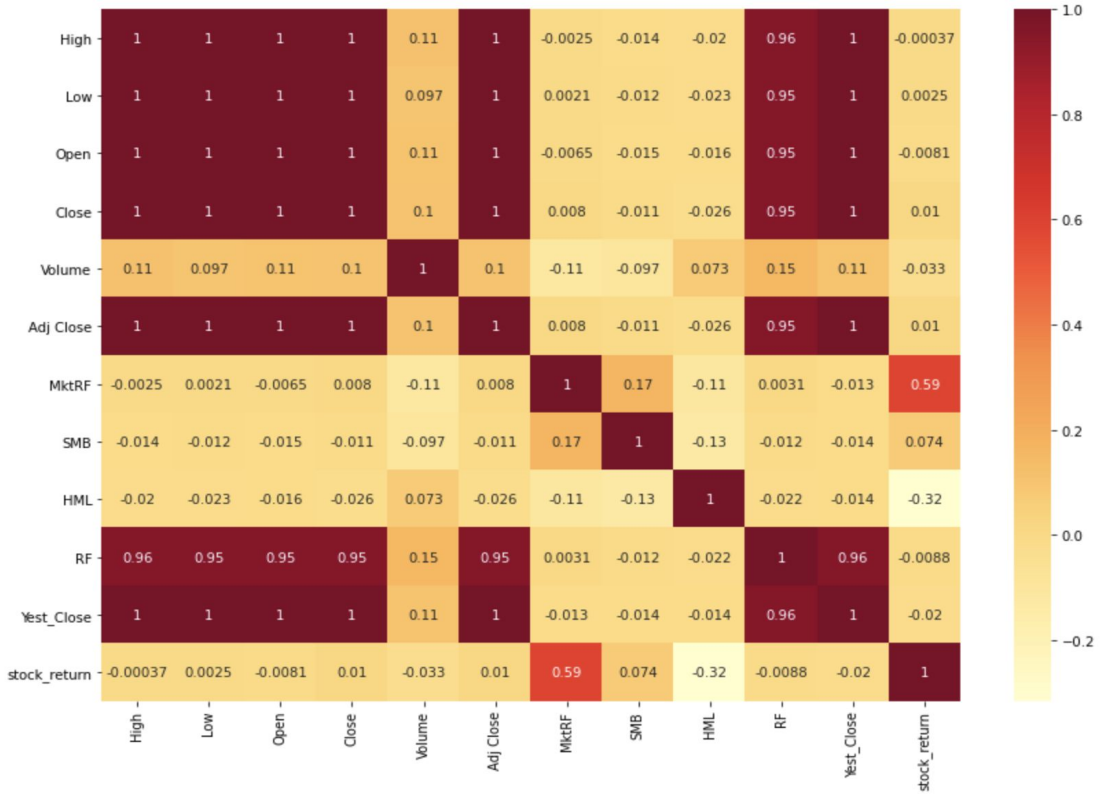
```
df_amzn
```

	Date	High	Low	Open	Close	Volume	Adj Close	MktRF	SMB	HML	RF	Yest_Close	stock_return
1	2014-01-03	402.709991	396.220001	398.290009	396.440002	2210200	396.440002	0.03	0.36	0.04	0.000	397.970001	-0.003845
2	2014-01-06	397.000000	388.420013	395.850006	393.630005	3170600	393.630005	-0.34	-0.58	0.28	0.000	396.440002	-0.007088
3	2014-01-07	398.470001	394.290009	395.040009	398.029999	1916000	398.029999	0.68	0.39	-0.39	0.000	393.630005	0.011178
4	2014-01-08	403.000000	396.040009	398.470001	401.920013	2316500	401.920013	0.04	0.01	-0.11	0.000	398.029999	0.009773
5	2014-01-09	406.890015	398.440002	403.709991	401.010010	2103000	401.010010	0.02	0.19	-0.41	0.000	401.920013	-0.002264
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1504	2019-12-23	1793.000000	1784.510010	1788.260010	1793.000000	2136400	1793.000000	0.10	0.21	-0.30	0.007	1786.500000	0.003638
1505	2019-12-24	1795.569946	1787.579956	1793.810059	1789.209961	881300	1789.209961	0.01	0.37	-0.02	0.007	1793.000000	-0.002114
1506	2019-12-26	1870.459961	1799.500000	1801.010010	1868.770020	6005400	1868.770020	0.48	-0.53	-0.01	0.007	1789.209961	0.044467
1507	2019-12-27	1901.400024	1866.010010	1882.920044	1869.800049	6186600	1869.800049	-0.10	-0.52	-0.07	0.007	1868.770020	0.000551
1508	2019-12-30	1884.000000	1840.619995	1874.000000	1846.890015	3674700	1846.890015	-0.57	0.17	0.60	0.007	1869.800049	-0.012253

1508 rows × 13 columns

DATA PREPROCESSING

- Following steps have been taken to use the data in the models and the algorithms:
- Columns from Fama – French 3 factor and ADS have been added to the initial Amazon dataset
- New columns; yest_close, stock_return and log_return have been calculated and added to the dataset
- Correlation Heatmap has been plotted to do the feature selection



Exploratory Data Analysis-EDA

Exploratory Data Analysis- EDA

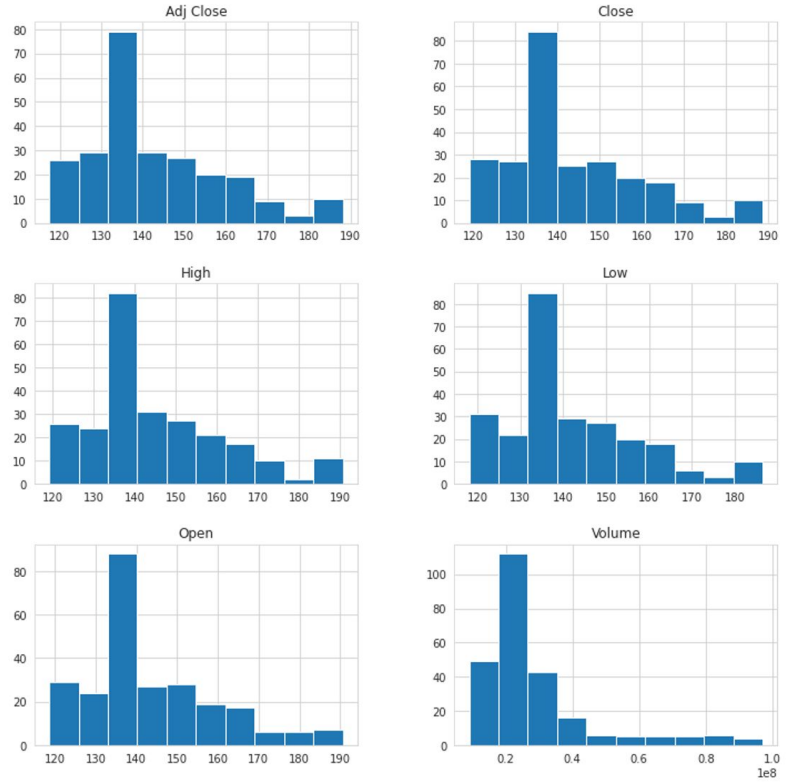
```
In [39]:
1 # Described AMAZON stock data
2 AMZN.describe()
```

Out[39]:

	High	Low	Open	Close	Volume	Adj Close	MA for 10 days	MA for 20 days	MA for 50 days	Daily Return
count	254.000000	254.000000	254.000000	254.000000	2.540000e+02	254.000000	245.000000	235.000000	205.000000	253.000000
mean	2634.152908	2561.751068	2599.165042	2599.083186	4.906697e+06	2599.083186	2604.364981	2609.415329	2620.969868	0.002581
std	577.053332	557.622006	571.520409	566.007158	1.996315e+06	566.007158	554.854824	543.505218	509.528599	0.024250
min	1750.000000	1626.030029	1641.510010	1676.609985	8.813000e+05	1676.609985	1772.487988	1821.103992	1920.880796	-0.079221
25%	2019.652527	1958.927521	1982.322510	1984.089966	3.450200e+06	1984.089966	2001.190002	2037.312494	2052.676995	-0.009969
50%	2623.954956	2530.614990	2602.354980	2586.770020	4.519400e+06	2586.770020	2584.126001	2590.415002	2651.184609	0.003056

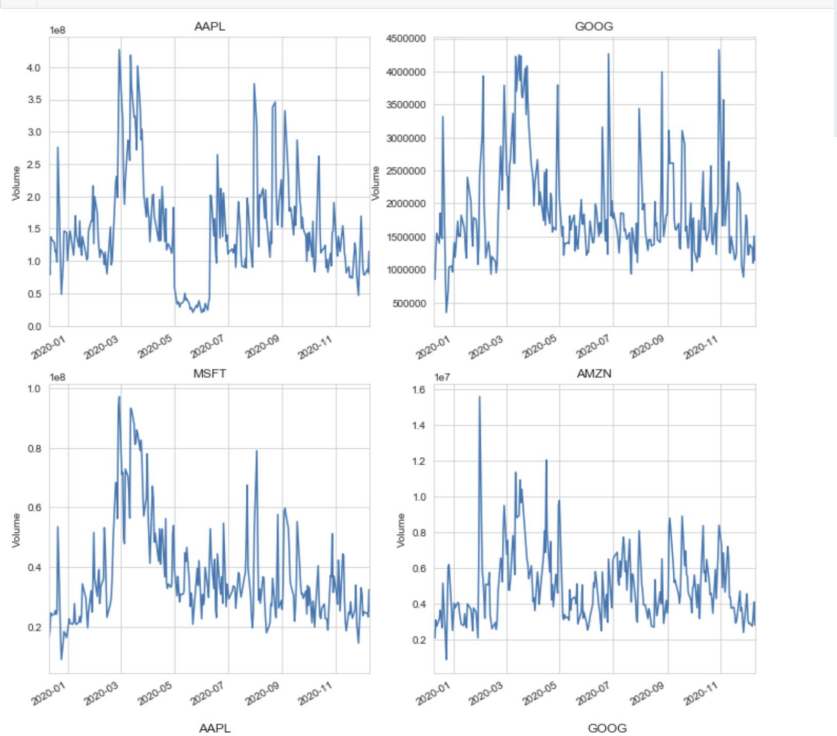
```
2 AMZN.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 254 entries, 2019-12-10 to 2020-12-10
Data columns (total 11 columns):
High              254 non-null float64
Low               254 non-null float64
Open              254 non-null float64
Close             254 non-null float64
Volume            254 non-null int64
Adj Close         254 non-null float64
company_name      254 non-null object
MA for 10 days    245 non-null float64
MA for 20 days    235 non-null float64
MA for 50 days    205 non-null float64
Daily Return      253 non-null float64
dtypes: float64(9), int64(1), object(1)
memory usage: 23.8+ KB
```

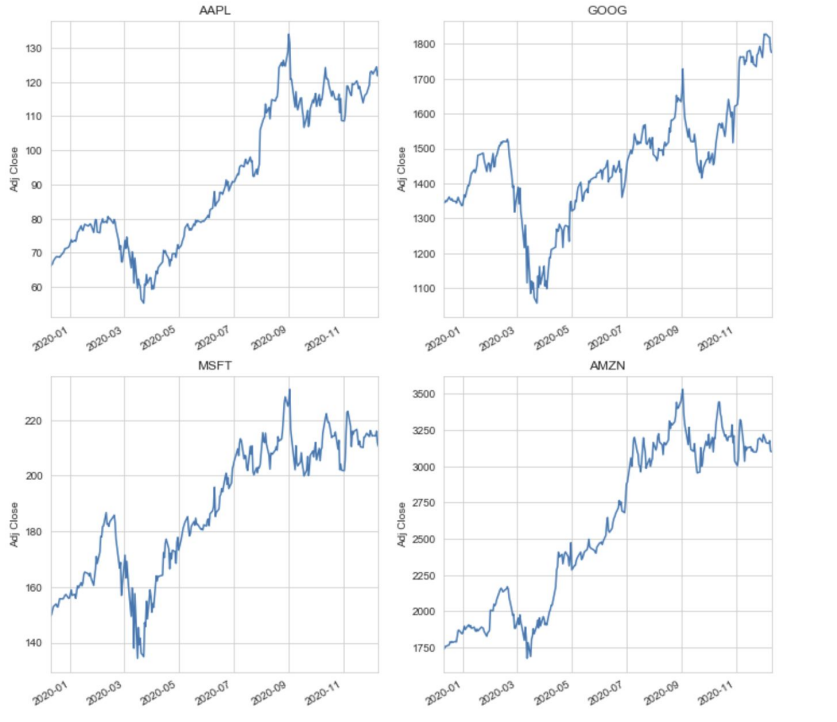


Above is the Histogram of Amazon.

★ Volume

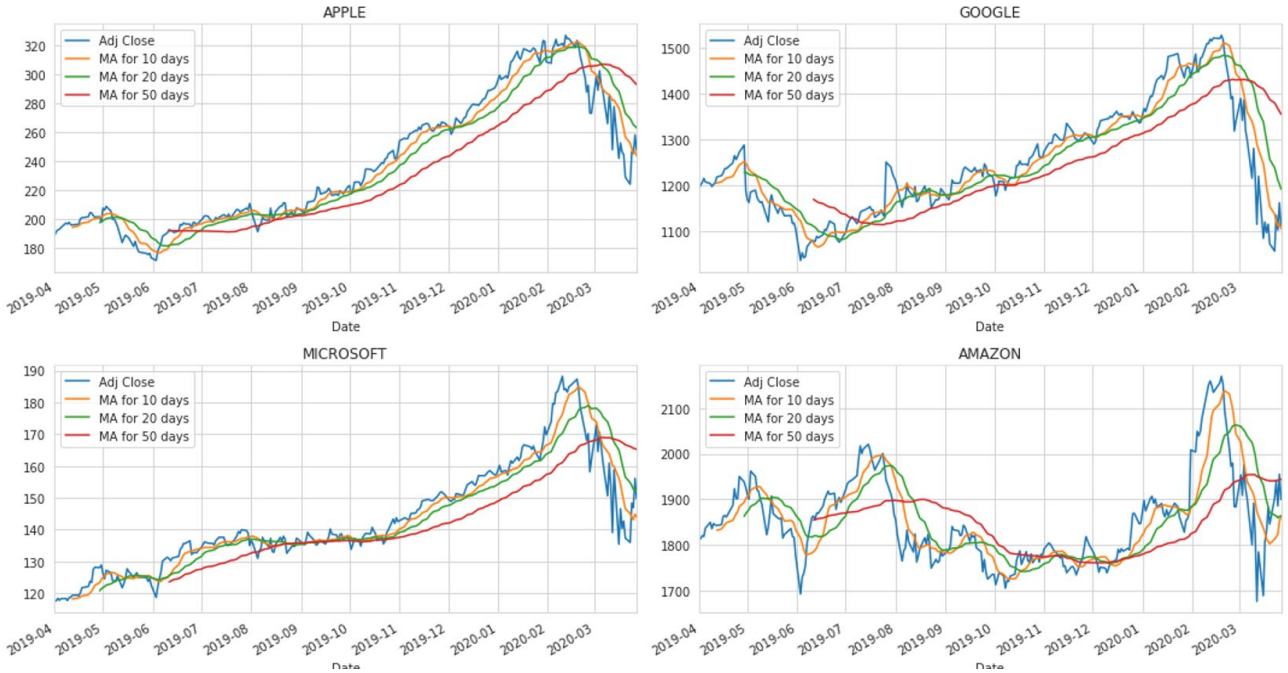


★ Adj Close

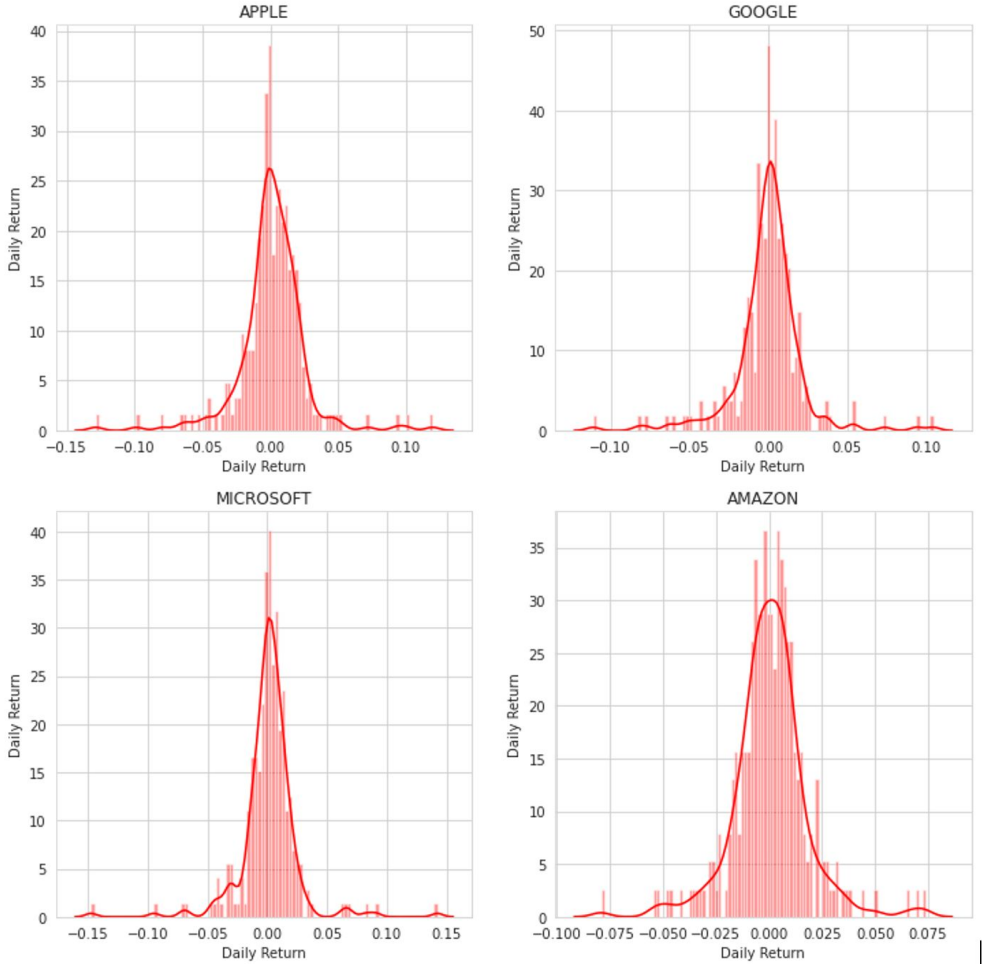


Exploratory Data Analysis- EDA

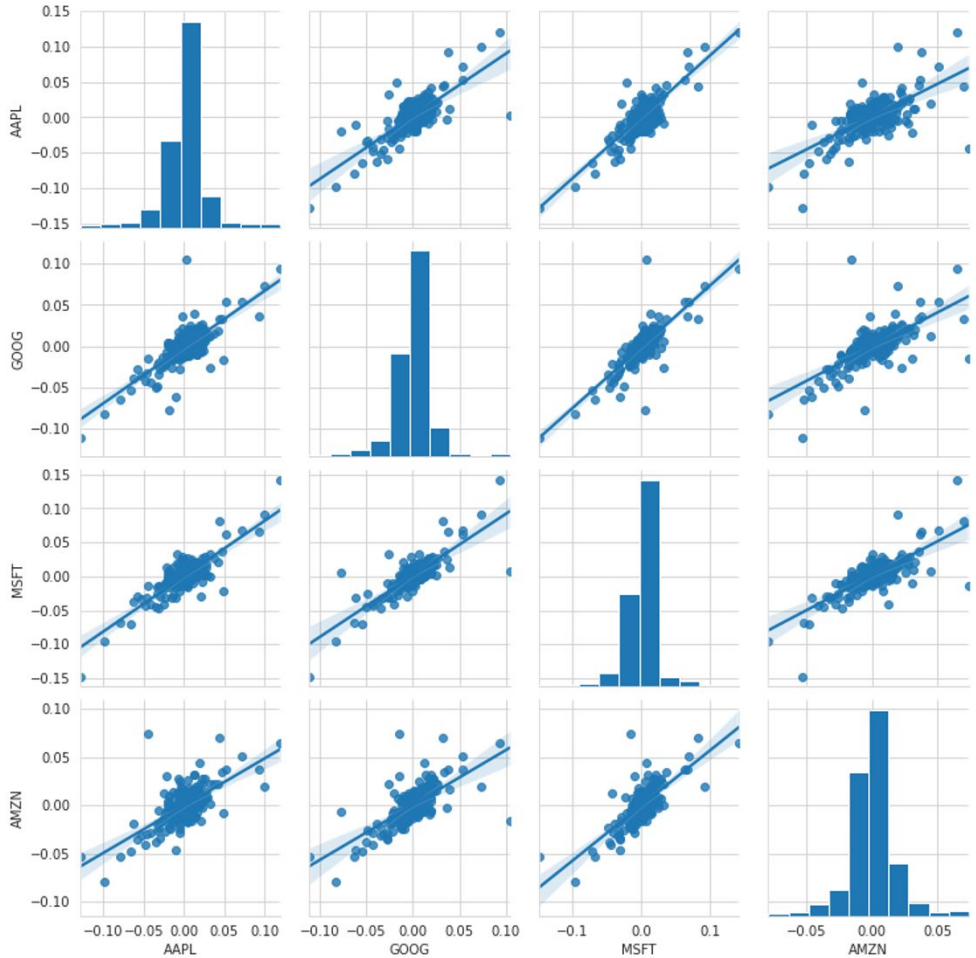
Below is the subplot of tech companies with moving average of 10 days, 20 days and 50 days.



Below we have plotted the same percent change of daily returns but using distplot() from seaborn package to get a better view. It gives a quick view at a univariate distribution of a stock.

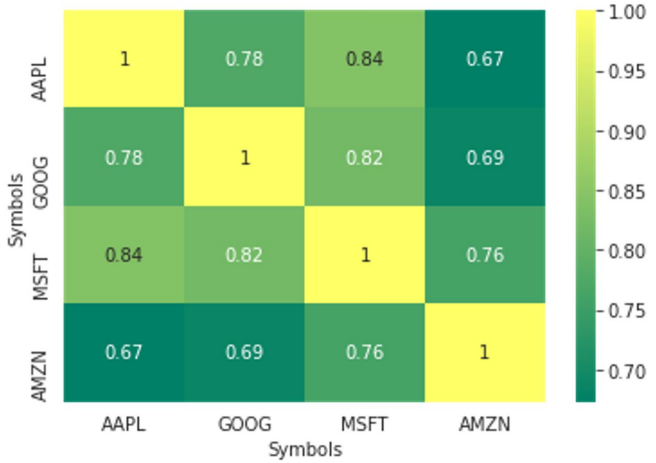


Below we've used pairplot() for comparison between all 4 companies. It shows how each tock is related to each other. We can say that every tech company related to each other. If stock price of one company increases then it will affect other companies stocks.



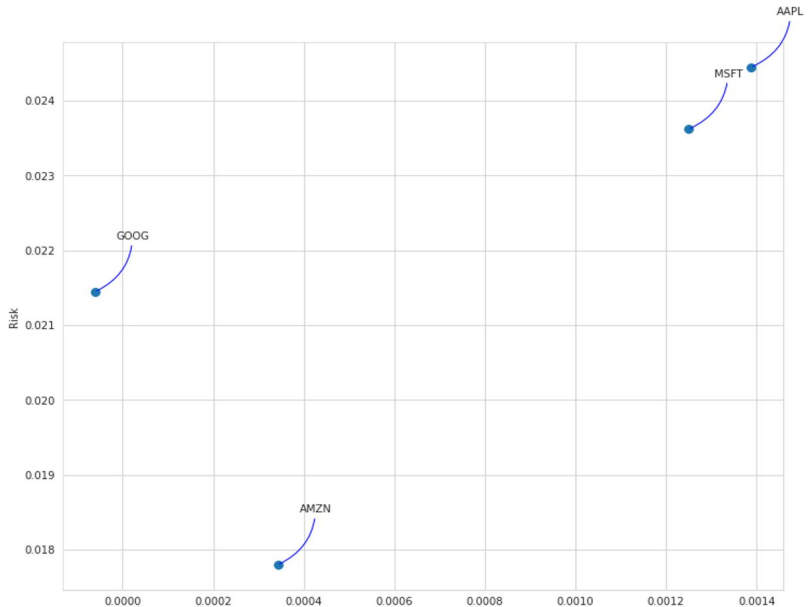
Exploratory Data Analysis

Below we've used a heatmap() from seaborn package to see the correlation between daily returns of the stocks from numerical point of view.

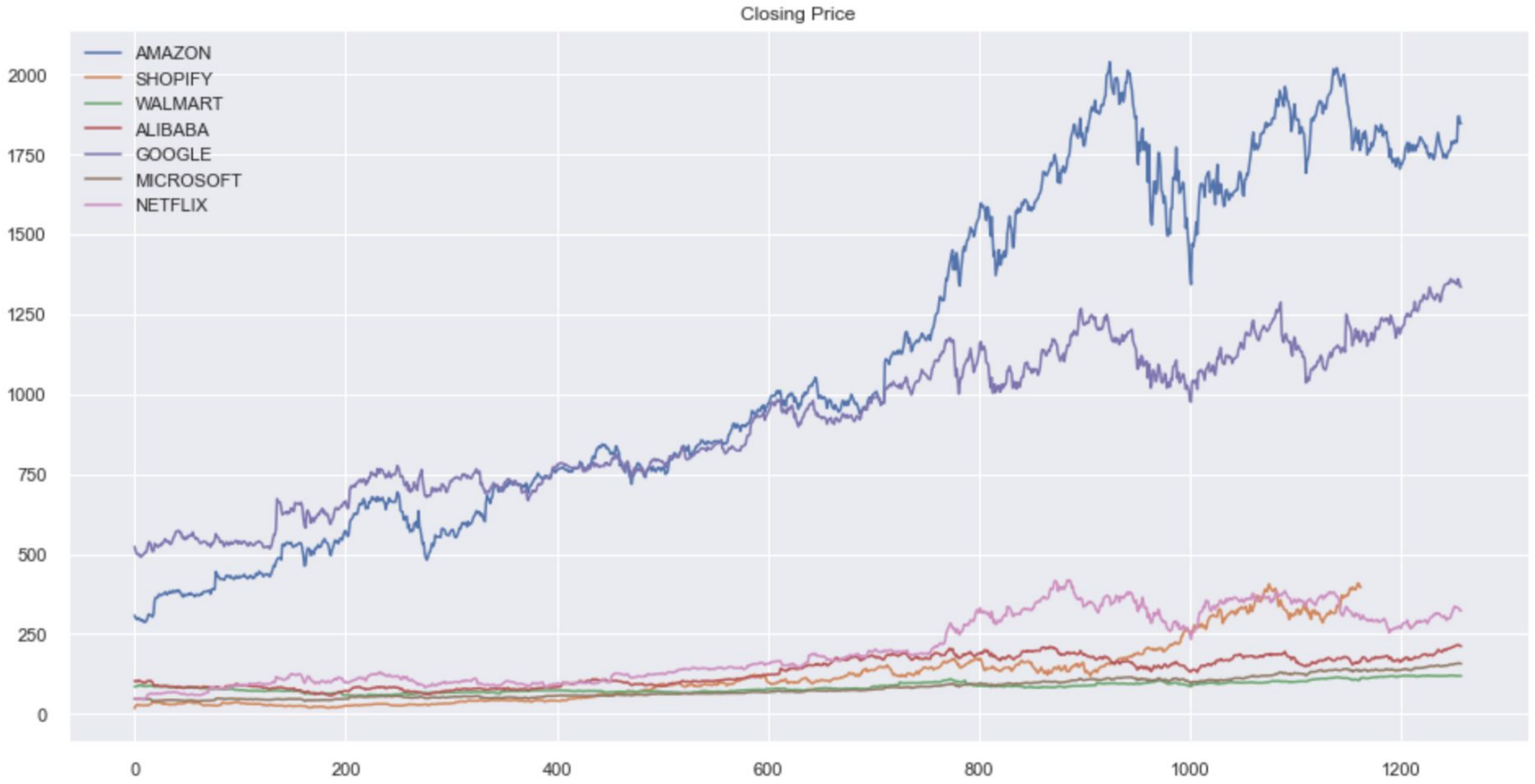


In conclusion, on the basis of returns we've made a graph of Expected returns V/S Risk. On the x-axis is Expected Returns and on the y-axis is Risk. We calculated the mean and standard deviation of the returns. So, from the chart it can be interpreted that

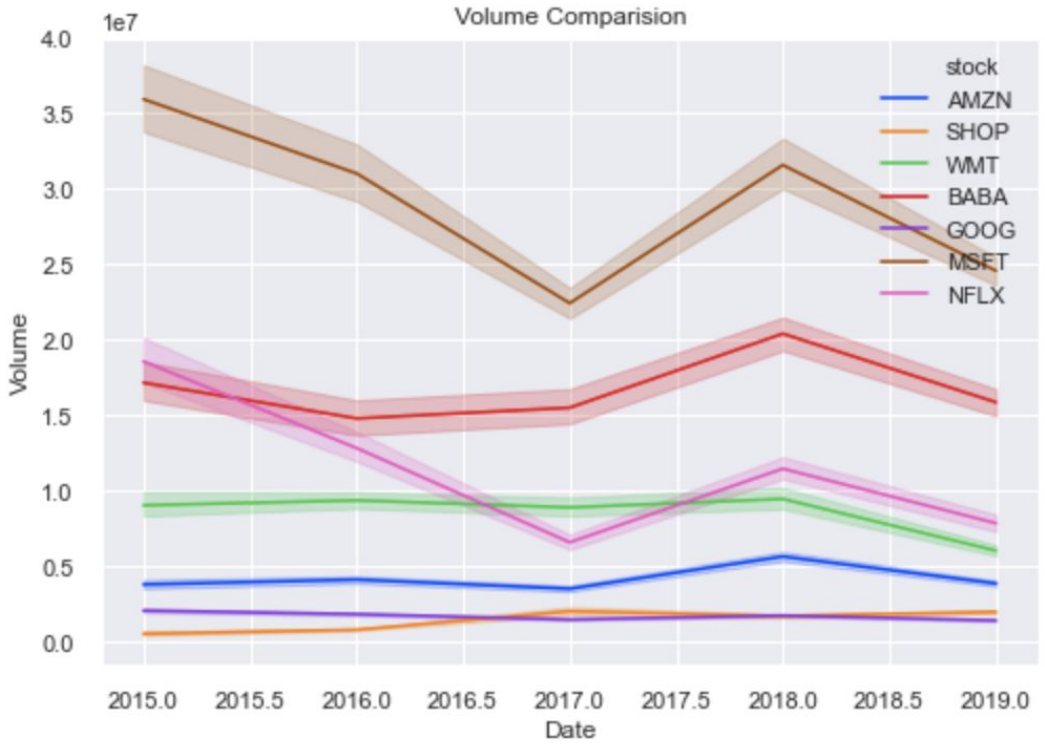
- Google will give less returns but its more riskier
- Amazon will give more returns and is less riskier
- Microsoft will give more returns but its risk is greater
- Apple will give the maximum returns compared to others but it's also the most riskier stock.



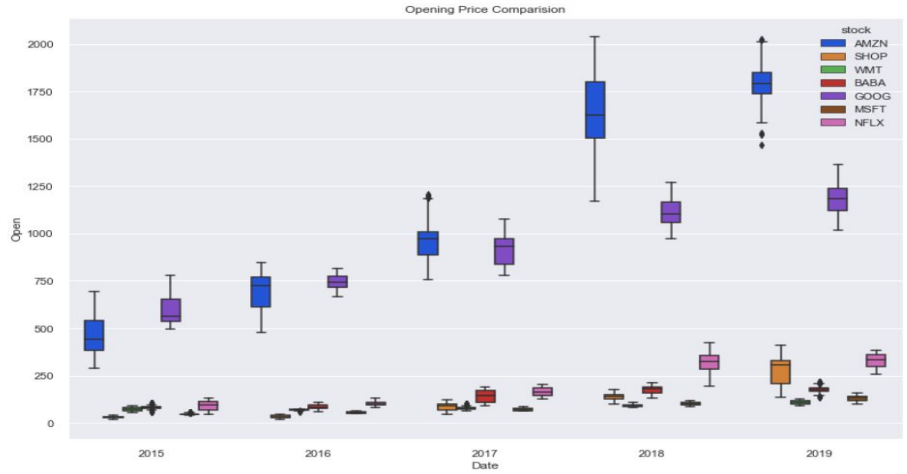
Closing Price Comparison of Stocks with Competitors

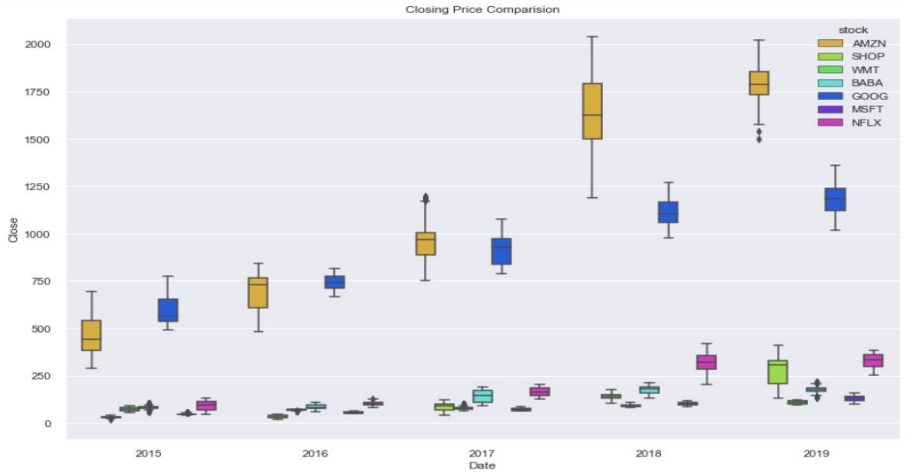


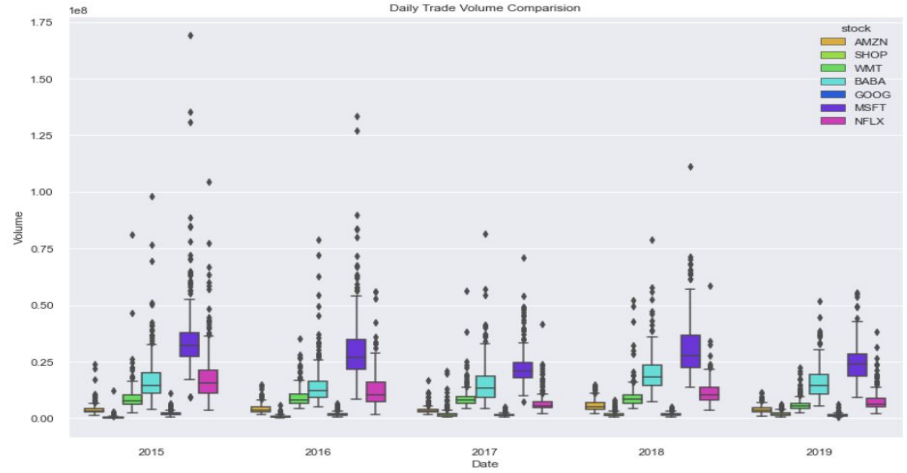
Comparison based on Trading Volume and Highest Price of Stocks

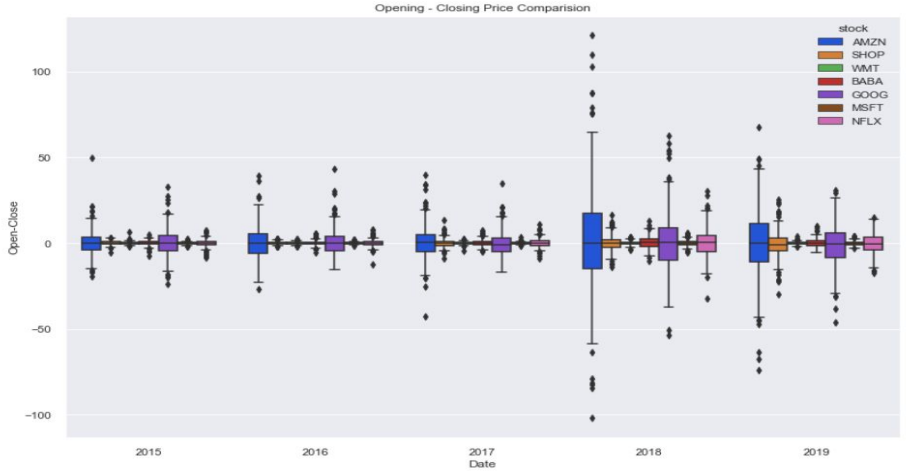


Comparison: BOXPLOT









INDICATORS

- Bollinger Bands
- Generating Trading signals using Moving Average and Exponential Moving Average Crossover Strategy

Indicators: Bollinger Bands

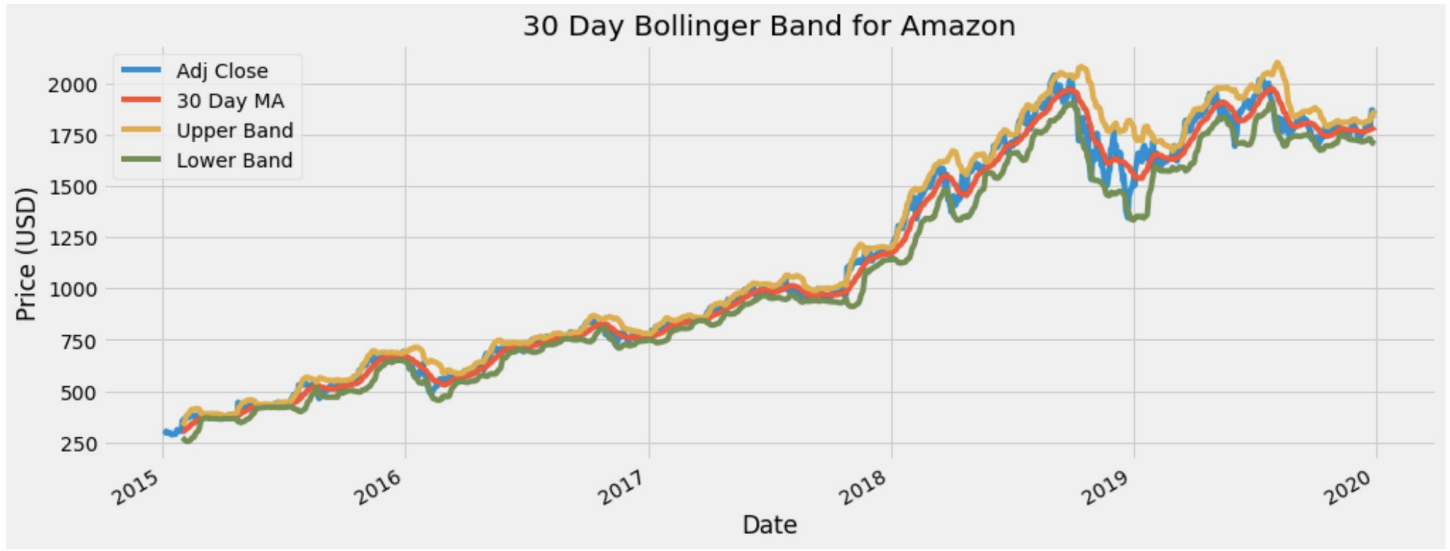
Bollinger Bands represent a key technical trading tool for financial traders. Bollinger bands are plotted by two (2) standard deviations (a measure of volatility) away from the moving average of a price. Bollinger Bands allow traders to monitor and take advantage of shifts in price volatilities.

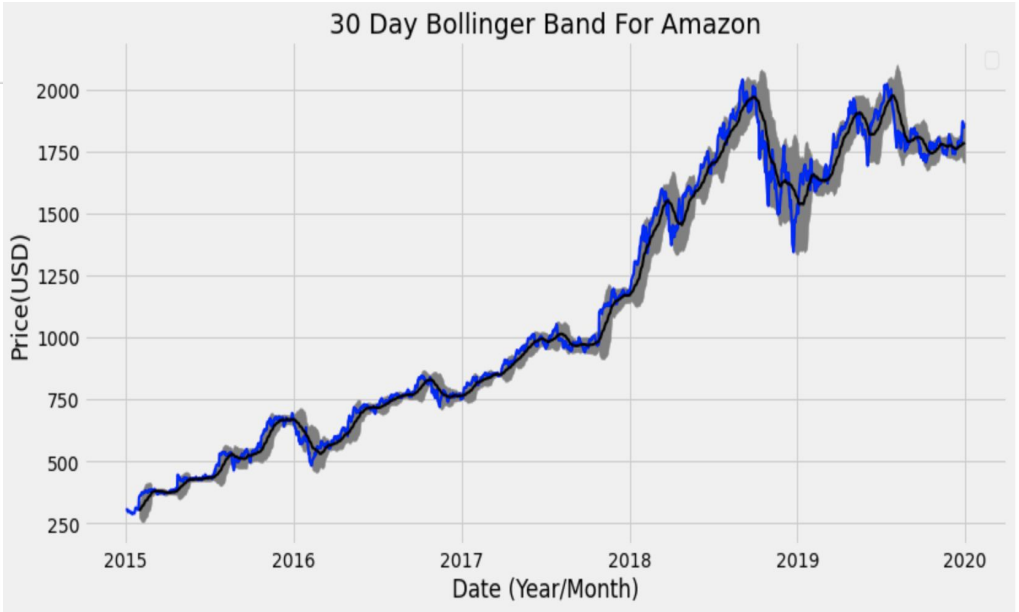
Main Components of a Bollinger Bands:

Upper Band: The upper band is simply two standard deviations above the moving average of a stock's price.

Middle Band: The middle band is simply the moving average of the stock's price.

Lower Band: Two standard deviations below the moving average is the lower band.





Moving Average Crossover Strategy

A moving average, also called as rolling average or running average is a used to analyze the time-series data by calculating a series of averages of the different subsets of full dataset.

Moving Averages are used to analyze the time-series data by calculati

Simple Moving Average Formula:

The simple moving average =

(sum of the an asset price over the past n periods)

/ (number of periods).

Moving Average Crossover Strategy - EMA

Exponential moving averages give more weight to the most recent periods and introduces lag.

Exponential Moving Average Formula:

EMA [today] = (α x Price [today]) + ((1 — α) x EMA [yesterday])

Where:

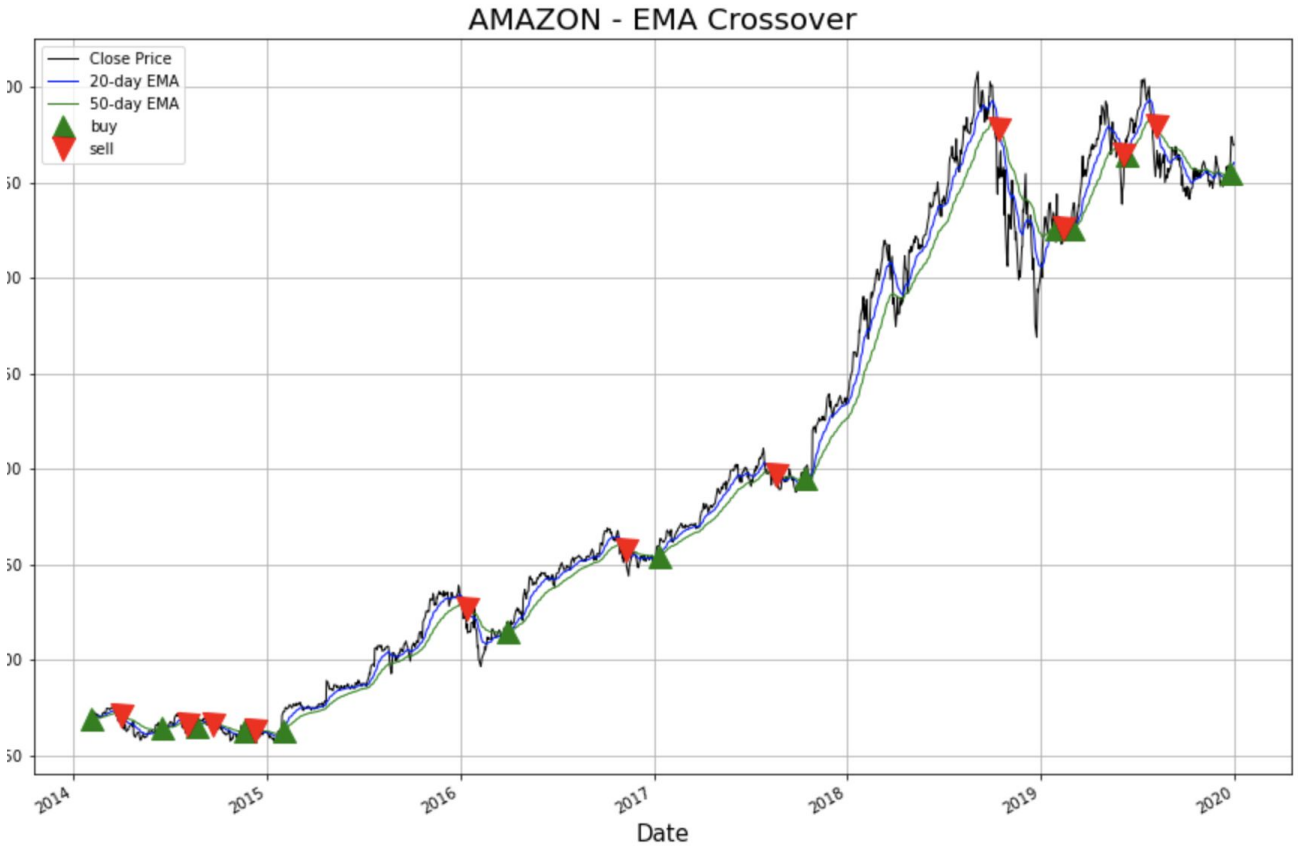
α = 2/(N + 1)

N = the length of the window (moving average period)

EMA [today] = the current EMA value

Price [today] = the current closing price

EMA [yesterday] = the previous EMA value



CANDIDATE MODELS

- KALMAN FILTER
- ELASTIC NET
- GARCH
- MIDAS
- LSTM
- RIDGE REGRESSION
- MOVING AVERAGE
- RANDOM FOREST
- MODEL COMBINATION (AutoRegressor, FamaFrench3 Model, Moving Average) using Random Forest Algorithm

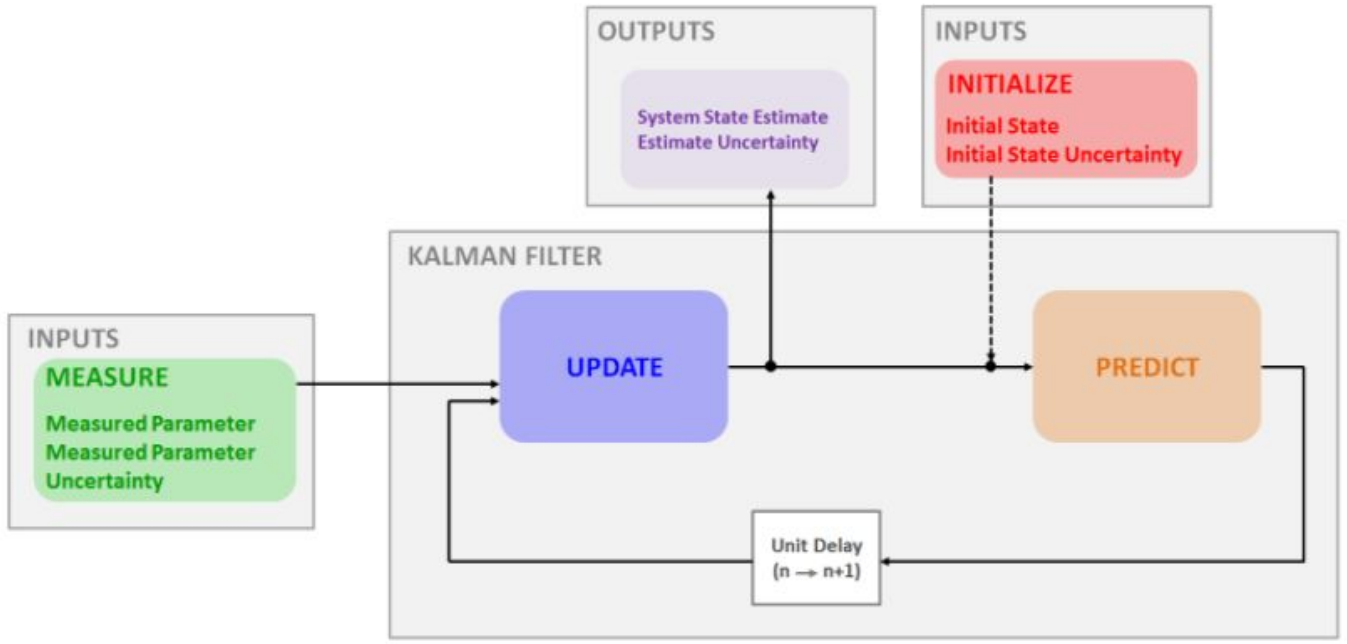
KALMAN FILTER

Kalman Filter

- An algorithm that provides estimates of some unknown variables given the measurements observed over time
- The Kalman filter process has two steps: the prediction step, where the next state of the system is predicted given the previous measurements, and the update step, where the current state of the system is estimated given the measurement at that time
- A predictor-corrector type estimator that is optimal in the sense that it minimizes the estimated error covariance when some presumed conditions are met

Kalman Filter

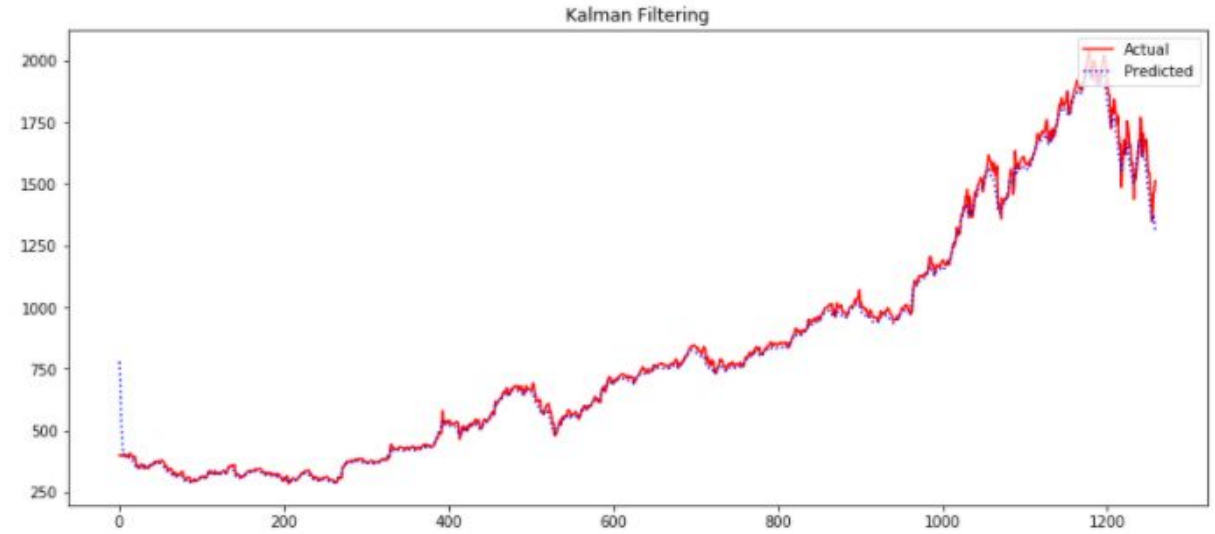
```
def Kalman_Filter(Y):
    S = Y.shape[0]
    S = S + 1
    "Initialize Params:"
    Z = param0[0]
    T = param0[1]
    H = param0[2]
    Q = param0[3]
    # "Kalman Filter Starts:"
    u_predict = np.zeros(S)
    u_update = np.zeros(S)
    P_predict = np.zeros(S)
    P_update = np.zeros(S)
    v = np.zeros(S)
    F = np.zeros(S)
    KF_Dens = np.zeros(S)
    for s in range(1,S):
        if s == 1:
            P_update[s] = 1000
            P_predict[s] =  T*P_update[1]*np.transpose(T)+Q
        else:
            F[s]= Z*P_predict[s-1]*np.transpose(Z)+H
            v[s] = Y[s-1] - Z*u_predict[s-1]
            u_update[s] = u_predict[s-1]+P_predict[s-1]*np.transpose(Z)*(1/F[s])*v[s]
            u_predict[s] = T*u_predict[s];
            P_update[s] = P_predict[s-1]-P_predict[s-1]*np.transpose(Z)*(1/F[s])*Z*P_predict[s-1]
            P_predict[s] = T*P_update[s]*np.transpose(T)+Q
            Likelihood = np.sum(KF_Dens[1:-1])
            return Likelihood
```



```
Y = amzn['Open']
T = Y.shape[0]
mu = 1196;


param0 = np.array([0.3, 0.9, 0.8, 1.1])
param_star = minimize(Kalman_Filter, param0, method='BFGS', options={'xtol': 1e-8, 'disp': True})
u = Kalman_Smoother(param_star.x, Y)

timevec = np.linspace(1,T,T)


fig= plt.figure(figsize=(14,6))
plt.plot(timevec, Y,'r-', label='Actual')
plt.plot(timevec, u,'b:', label='Predicted')
plt.legend(loc='upper right')
plt.title("Kalman Filtering")
plt.show()
```

```
C:\Users\Siddhi\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: OptimizeWarning: Unknown solver options: xtol
  import sys
```

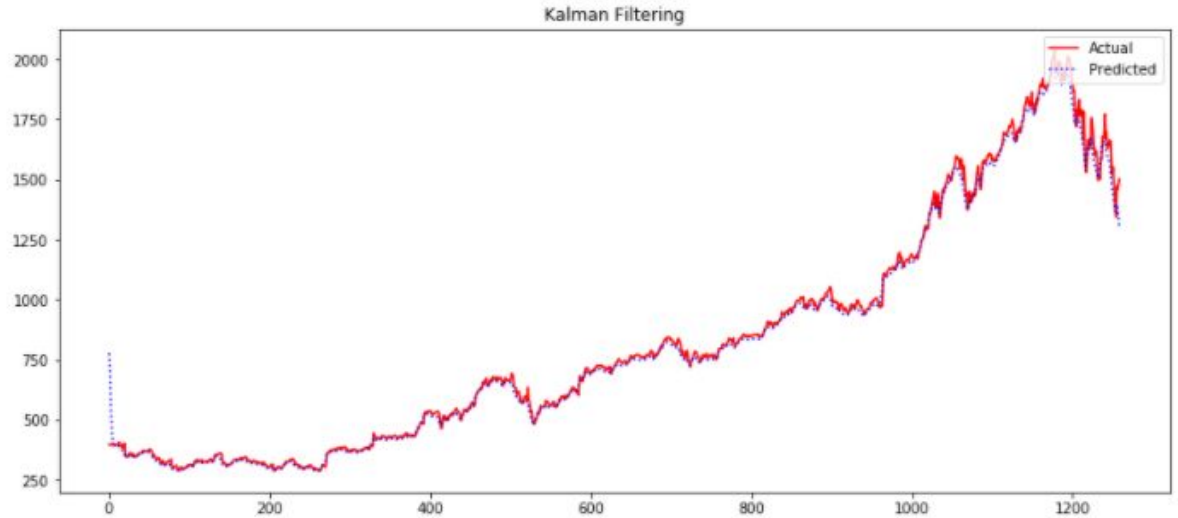
```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 0
         Function evaluations: 6
         Gradient evaluations: 1
```




```
Y = amzn['Close']
T = Y.shape[0]
mu = 1196;


param0 = np.array([0.3, 0.9, 0.8, 1.1])
param_star = minimize(Kalman_Filter, param0, method='BFGS', options={'xtol': 1e-8, 'disp': True})
u = Kalman_Smoother(param_star.x, Y)

timevec = np.linspace(1,T,T)


fig= plt.figure(figsize=(14,6))
plt.plot(timevec, Y,'r-', label='Actual')
plt.plot(timevec, u,'b:', label='Predicted')
plt.legend(loc='upper right')
plt.title("Kalman Filtering")
plt.show()
```

```
C:\Users\Siddhi\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: OptimizeWarning: Unknown solver options: xtol
  import sys
```

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 0
         Function evaluations: 6
         Gradient evaluations: 1
```



Long Short Day Trading Strategy

```
amount = 10000
signal = 0
Amount = []
balance = 0
action = []
portfolio = 0
Portfolio = []
stocks = 0
Stocks = []

for i in range(len(results)):
    if results['Predicted'][i] > results['Actual'][i-1]:
        action.append('Buy at Open')
        stocks = int(amount/results['Open'][i])
        balance = int(amount%results['Close'][i])
        portfolio = stocks * results ['Open'][i]
        print(i,'Buy at Open',round(portfolio,2),stocks,round(balance,2))


        action.append('Sell at End')
        portfolio = stocks * results['Close'][i]
        signal = 0
        stocks = 0
        amount = balance + portfolio
        portfolio = 0
        balance = 0
        print(i,'Sell at Close',round(amount,2),balance)
        Amount.append(amount)

    else:
        action.append('Sell at Open')
        stocks = int(amount/results['Open'][i])
        balance = int(amount%results['Close'][i])
        portfolio = stocks * results ['Open'][i]
        print(i,'Sell at Open',round(portfolio,2),'-',stocks,round(balance,2))


        action.append('Buy at Close')
        portfolio = stocks * results['Close'][i]
        signal = 0
        stocks = 0
        amount = balance + portfolio
        portfolio = 0
        balance = 0
        print(i,'Buy Back at Close',round(amount,2),balance)
        Amount.append(amount)
    print('\n')
```

- if predicted > yesterdays close, buy and sell at end of day
- if predicted < yesterdays close, sell and buy at end of day

```
0 Sell at Open 9970.0 - 25 50
0 Buy Back at Close 9999.25 0


1 Buy at Open 9957.25 25 88
1 Sell at Close 9999.0 0


2 Buy at Open 9896.25 25 158
2 Sell at Close 9998.75 0


3 Buy at Open 9876.0 25 48
3 Sell at Close 9998.75 0


4 Buy at Open 9961.75 25 352
4 Sell at Close 10400.0 0
```

```
mean_returns = results['Returns'].mean()
sd = results['Returns'].std()
print(mean_returns,sd)
Market_RF = 0.0464


Sharpe_Ratio = np.sqrt(878)*(mean_returns)/sd
Sharpe_Ratio
```

```
0.001127517787779276 0.06723535950493167

0.49690449870956616
```

```
from sklearn.metrics import mean_squared_error
from math import sqrt

rms = sqrt(mean_squared_error(results['Actual'], results['Predicted']))
rms
```

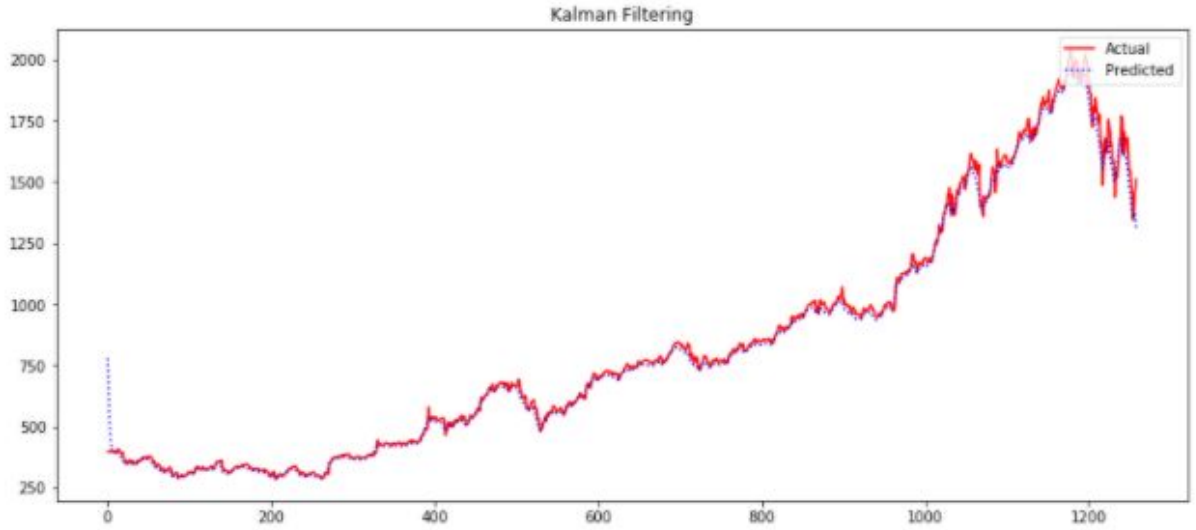
```
: 28.36511171691858
```

Comparison with Shopify stock - Open Price

```
Y = amzn['Open']
T = Y.shape[0]
mu = 1196;


param0 = np.array([0.3, 0.9, 0.8, 1.1])
param_star = minimize(Kalman_Filter, param0, method='BFGS', options={'xtol': 1e-8, 'disp': True})
u = Kalman_Smoother(param_star.x, Y)

timevec = np.linspace(1,T,T)


fig= plt.figure(figsize=(14,6))
plt.plot(timevec, Y,'r-', label='Actual')
plt.plot(timevec, u,'b:', label='Predicted')
plt.legend(loc='upper right')
plt.title("Kalman Filtering")
plt.show()
```

```
C:\Users\Siddhi\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: OptimizeWarning: Unknown solver options: xtol
  import sys
```

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 0
         Function evaluations: 6
         Gradient evaluations: 1
```




```
Y = shop['Open']
T = Y.shape[0]
mu = 1196;


param0 = np.array([0.3, 0.9, 0.8, 1.1])
param_star = minimize(Kalman_Filter, param0, method='BFGS', options={'xtol': 1e-8, 'disp': True})
u = Kalman_Smoother(param_star.x, Y)

timevec = np.linspace(1,T,T)


fig= plt.figure(figsize=(14,6))
plt.plot(timevec, Y,'r-', label='Actual')
plt.plot(timevec, u,'b:', label='Predicted')
plt.legend(loc='upper right')
plt.title("Kalman Filtering- Shopify Stock- Open")
plt.show()
```

```
C:\Users\Siddhi\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: OptimizeWarning: Unknown solver options: xtol
  import sys
```

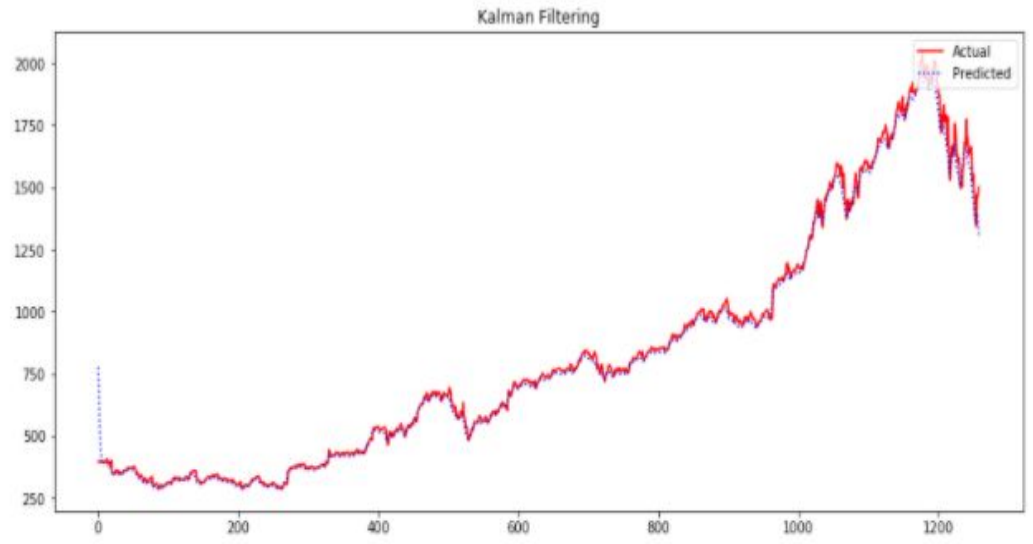
```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 0
         Function evaluations: 6
         Gradient evaluations: 1
```

Comparison with Shopify stock - Close Price

```
Y = amzn['Close']
T = Y.shape[0]
mu = 1196;


param0 = np.array([0.3, 0.9, 0.8, 1.1])
param_star = minimize(Kalman_Filter, param0, method='BFGS', options={'xtol': 1e-8, 'disp': True})
u = Kalman_Smoother(param_star.x, Y)

timevec = np.linspace(1,T,T)


fig= plt.figure(figsize=(14,6))
plt.plot(timevec, Y,'r-', label='Actual')
plt.plot(timevec, u,'b:', label='Predicted')
plt.legend(loc='upper right')
plt.title("Kalman Filtering")
plt.show()
```

```
C:\Users\Siddhi\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: OptimizeWarning: Unknown solver options: xtol
  import sys
```

```
Optimization terminated successfully.
        Current function value: 0.000000
        Iterations: 0
        Function evaluations: 6
        Gradient evaluations: 1
```




```
Y = shop['Close']
T = Y.shape[0]
mu = 1196;


param0 = np.array([0.3, 0.9, 0.8, 1.1])
param_star = minimize(Kalman_Filter, param0, method='BFGS', options={'xtol': 1e-8, 'disp': True})
u = Kalman_Smoother(param_star.x, Y)

timevec = np.linspace(1,T,T)


fig= plt.figure(figsize=(14,6))
plt.plot(timevec, Y,'r-', label='Actual')
plt.plot(timevec, u,'b:', label='Predicted')
plt.legend(loc='upper right')
plt.title("Kalman Filtering- Shopify Stock - Close")
plt.show()
```

```
C:\Users\Siddhi\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: OptimizeWarning: Unknown solver options: xtol
  import sys
```

```
Optimization terminated successfully.
        Current function value: 0.000000
        Iterations: 0
        Function evaluations: 6
        Gradient evaluations: 1
```

Comparison with Shopify stock - RMSE value

RMSE Value of Amazon

```
from sklearn.metrics import mean_squared_error
from math import sqrt

rms = sqrt(mean_squared_error(results['Actual'], results['Predicted']))
rms
```

```
3]: 28.36511171691858
```

RMSE Value of Shopify

```
rms = sqrt(mean_squared_error(dif['Actual'],dif['Predicted']))
rms
```

```
15]: 3.5412598855655055
```

Kalman Filter Model with Bollinger Bands and Macd

- Bollinger Bands present a framework for determining whether prices are high or low on a relative basis

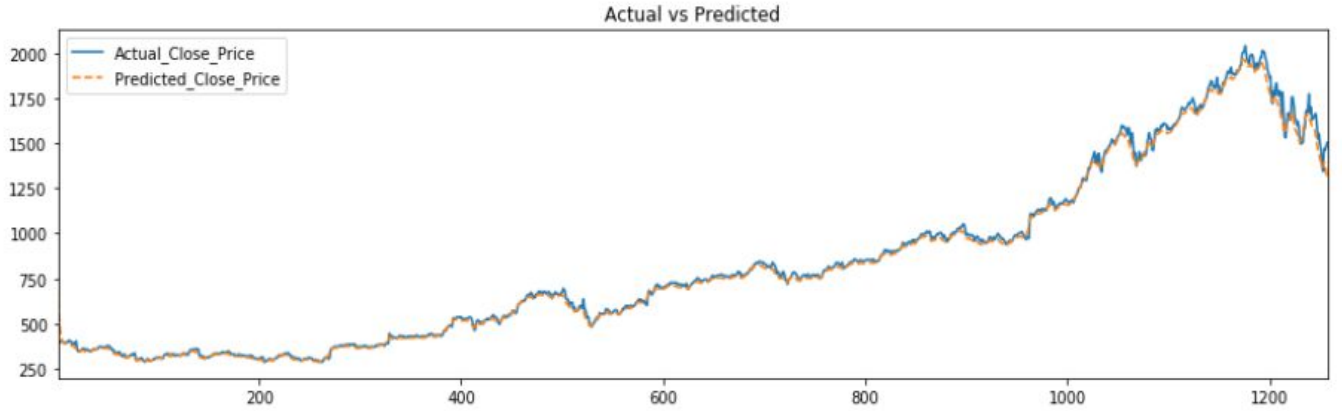
- Bollinger Bands are curves drawn in and around the price structure usually consisting of a moving average (the middle band), an upper band, and a lower band that answer the question as to whether prices are high or low on a relative basis

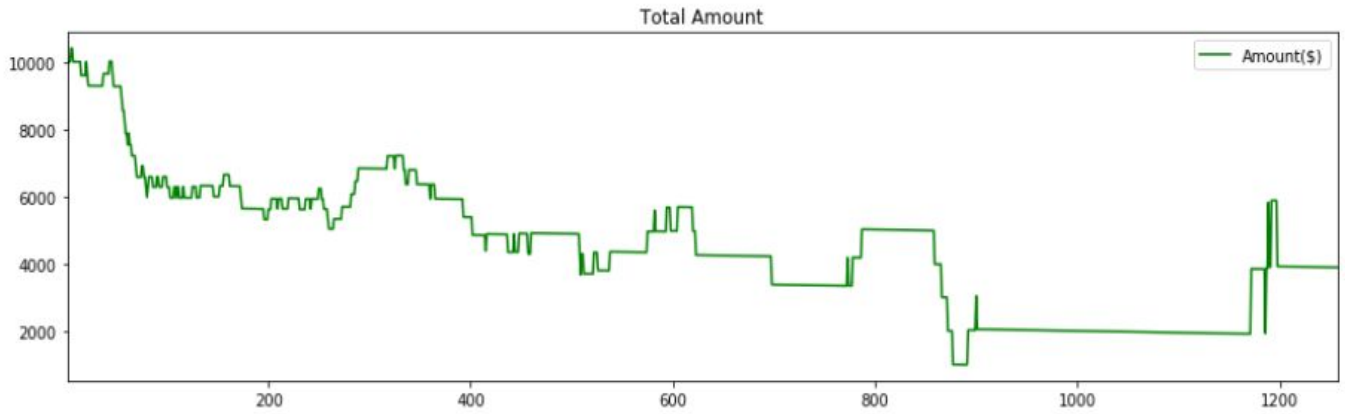
- Moving average convergence divergence (**MACD**) is a trend-following momentum indicator that shows the relationship between two moving averages of a security's price

Long Short Day trading

```
Strategy_Result[['Actual_Close_Price','Predicted_Close_Price']].loc[0:].plot(figsize=(15,4),
                                                                             style=['-','--'],
                                                                             title = 'Actual vs Predicted')


Strategy_Result[['Amount($)']].loc[0:].plot(figsize=(15,4),
                                            style=['-g'],
                                            title = 'Total Amount')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d2e105d388>
```





- if predicted > yesterdays close, buy and sell at end of day
- if predicted < yesterdays close, sell and buy at end of day

```
Profit = (Amount[-1])/10000
print('Initial_Investment : 10000$')
print('Final Amount:',Amount[-1],'$')
print ('Profit_Percent:',Profit*100,'%')
```

```
Initial_Investment : 10000$
Final Amount: 3899.0400390625 $
Profit_Percent: 38.990400390625 %
```

- Sharpe Ratio

```
Strategy_Result['Returns'] = Strategy_Result['Amount($)'].pct_change()
```

```
mean_returns = Strategy_Result['Returns'].mean()
sd = Strategy_Result['Returns'].std()
print(mean_returns,sd)
Market_RF = 0.0464


Sharpe_Ratio = np.sqrt(877)*(mean_returns)/sd
Sharpe_Ratio
```

```
0.0011284359328211418 0.06726213328360986
```

```
3]: 0.4968280031452998
```

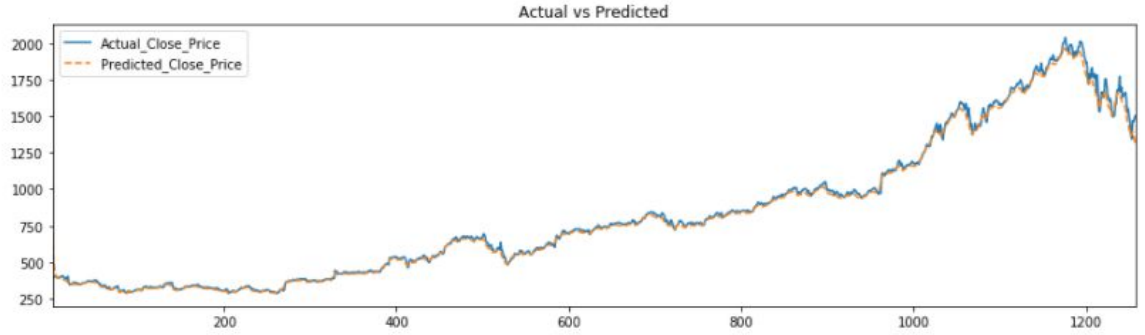
Buy & Hold Trading Strategy

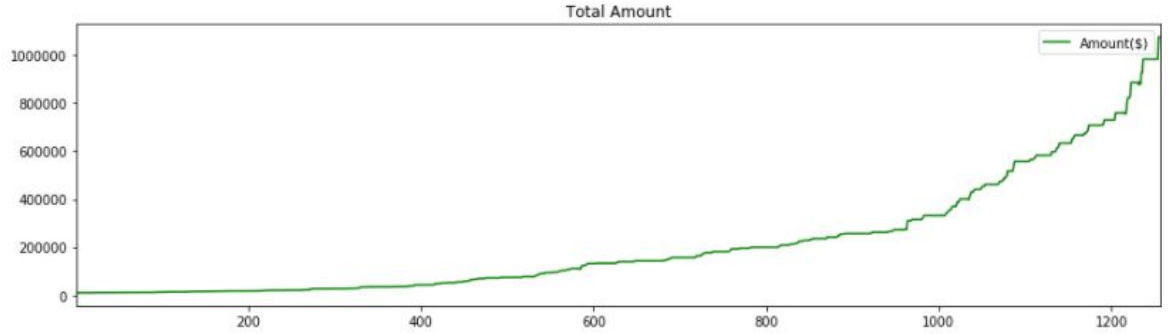
Position trading

- Buy if we have next days predicted_value greater than todays close value and hold if already bought
- Sell if we have next days predicted_value lesser than todays close value and dont buy until rule 1

```
Strategy_Result[['Actual_Close_Price','Predicted_Close_Price']].loc[0:].plot(figsize=(15,4),
                                                                             style=['-','--'],
                                                                             title = 'Actual vs Predicted')


Strategy_Result[['Amount($)']].loc[0:].plot(figsize=(15,4),
                                            style=['-g'],
                                            title = 'Total Amount')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d2deca7488>
```






```
Profit = (Amount[-1])/10000
print('Initial_Investment : 10000$')
print('Final Amount:',Amount[-1],'$')
print ('Profit_Percent:',Profit*100,'%')
```

```
Initial_Investment : 10000$
Final Amount: 1075656.0 $
Profit_Percent: 10756.56 %
```

- Sharpe Ratio

```
Strategy_Result['Returns'] = Strategy_Result['Amount($)'].pct_change()
```

```
mean_returns = Strategy_Result['Returns'].mean()
sd = Strategy_Result['Returns'].std()
print(mean_returns,sd)
Market_RF = 0.0464


Sharpe_Ratio = np.sqrt(877)*(mean_returns)/sd
Sharpe_Ratio
```

```
0.003801043380943083 0.012059451562487394
```

```
9.334156225557479
```

Buy & Hold Trading Strategy with Bollinger Bands

```
long_MA = results['Actual'].rolling(window=20).mean()
results['Long_Moving_Avg'] = long_MA
```

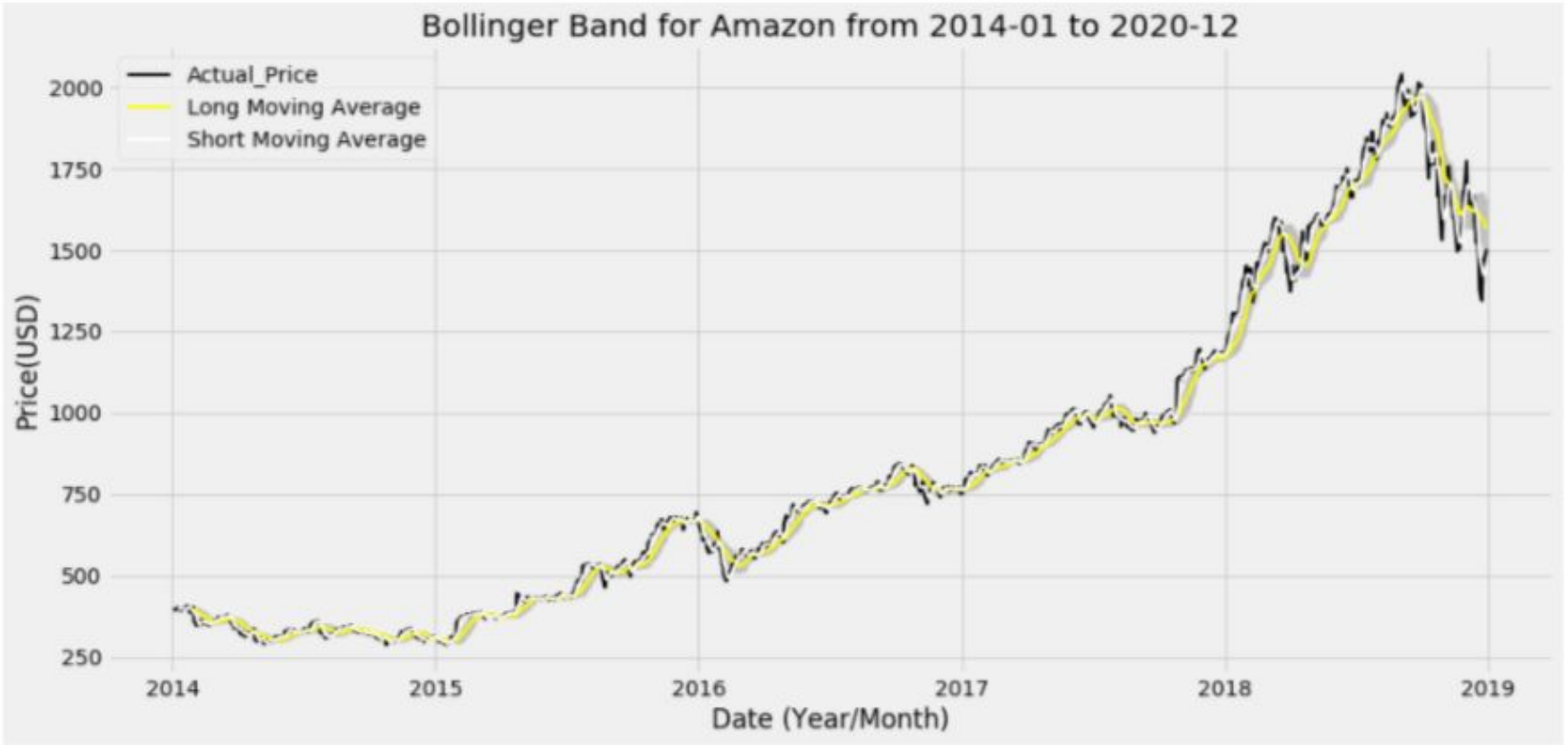
```
short_MA  = results['Actual'].rolling(window=5).mean()
results['Short_Moving_Avg'] = short_MA
```

```
long_std= results['Actual'].rolling(window=20).std()
results['Long_std'] = long_std
```

```
long_upper_band = ((long_MA) + (long_std)*0.75)
results['Long_Upper_Band'] = long_upper_band
```

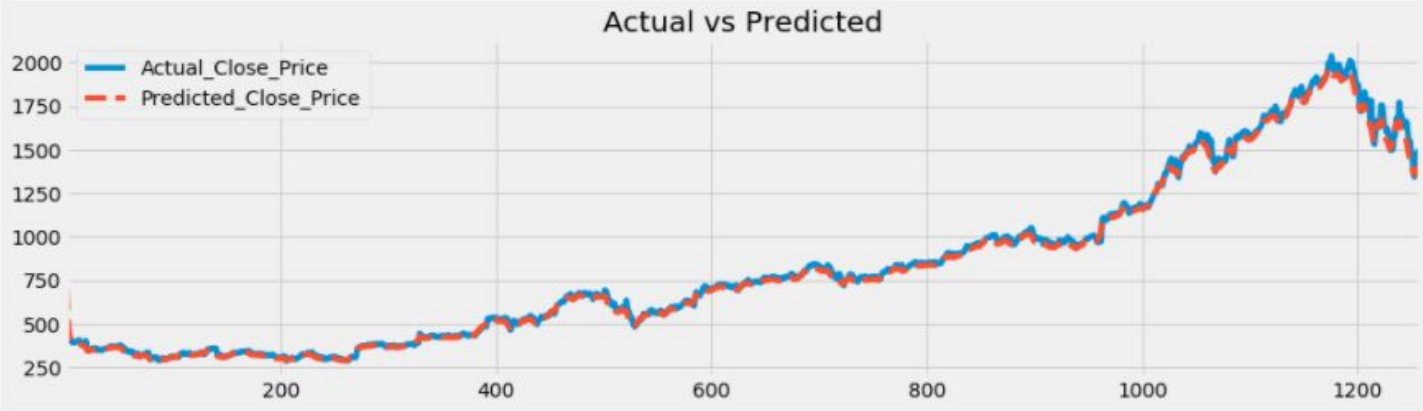
```
long_lower_band = ((long_MA) - (long_std)*0.75)
results['Long_Lower_Band'] = long_lower_band
```

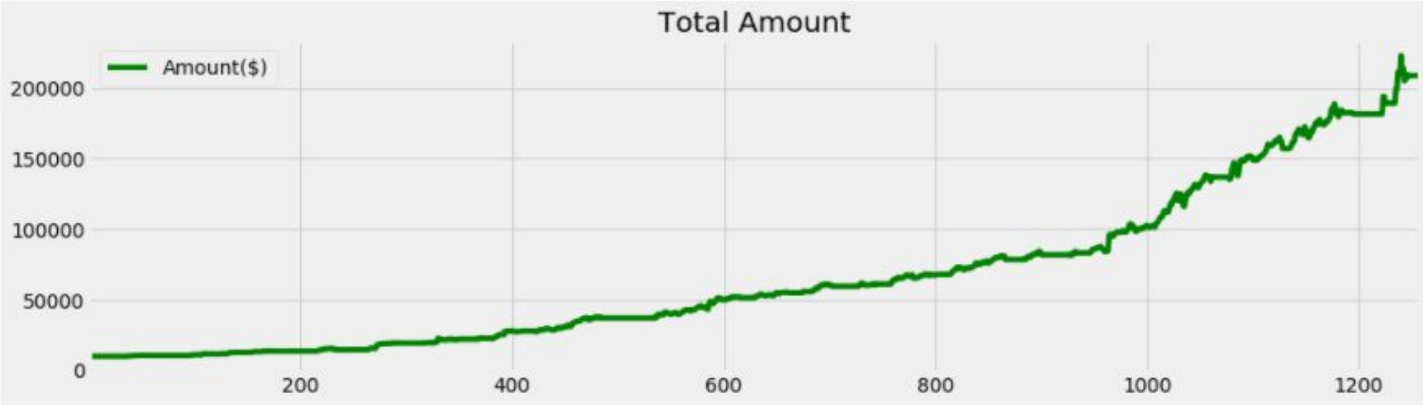
```
plt.style.use('fivethirtyeight')
fig = plt.figure(figsize=(15,7))
ax = fig.add_subplot(111)
x_axis = results.index
ax.fill_between(x_axis, results['Long_Upper_Band'], results['Long_Lower_Band'], color='silver')
ax.plot(x_axis, results['Actual'], color='black', lw=2, label = 'Actual_Price')
ax.plot(x_axis, results['Long_Moving_Avg'], color='yellow', lw=2, label = 'Long Moving Average')
ax.plot(x_axis, results['Short_Moving_Avg'], color='white', lw=2, label = 'Short Moving Average')

ax.set_title('Bollinger Band for Amazon from 2014-01 to 2020-12')
ax.set_xlabel('Date (Year/Month)')
ax.set_ylabel('Price(USD)')
ax.legend()
plt.show()
```



```
Strategy_Result[['Actual_Close_Price','Predicted_Close_Price']].loc[0:].plot(figsize=(15,4),
                                                                             style=['-','--'],
                                                                             title = 'Actual vs Predicted')


Strategy_Result[['Amount($)']].loc[0:].plot(figsize=(15,4),
                                            style=['-g'],
                                            title = 'Total Amount')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d2e1076548>
```






```
Profit = (Amount[-1])/10000
print('Initial_Investment : 10000$')
print('Final Amount:',Amount[-1],'$')
print ('Profit_Percent:',Profit*100,'%')
```

```
Initial_Investment : 10000$
Final Amount: 208744.0 $
Profit_Percent: 2087.44 %
```

```
Profit = (Amount[-1])/10000
print('Initial_Investment : 10000$')
print('Final Amount:',Amount[-1],'$')
print ('Profit_Percent:',Profit*100,'%')
```

```
Initial_Investment : 10000$
Final Amount: 208744.0 $
Profit_Percent: 2087.44 %
```

Sharpe Ratio

```
Strategy_Result['Returns'] = Strategy_Result['Amount($)'].pct_change()
```

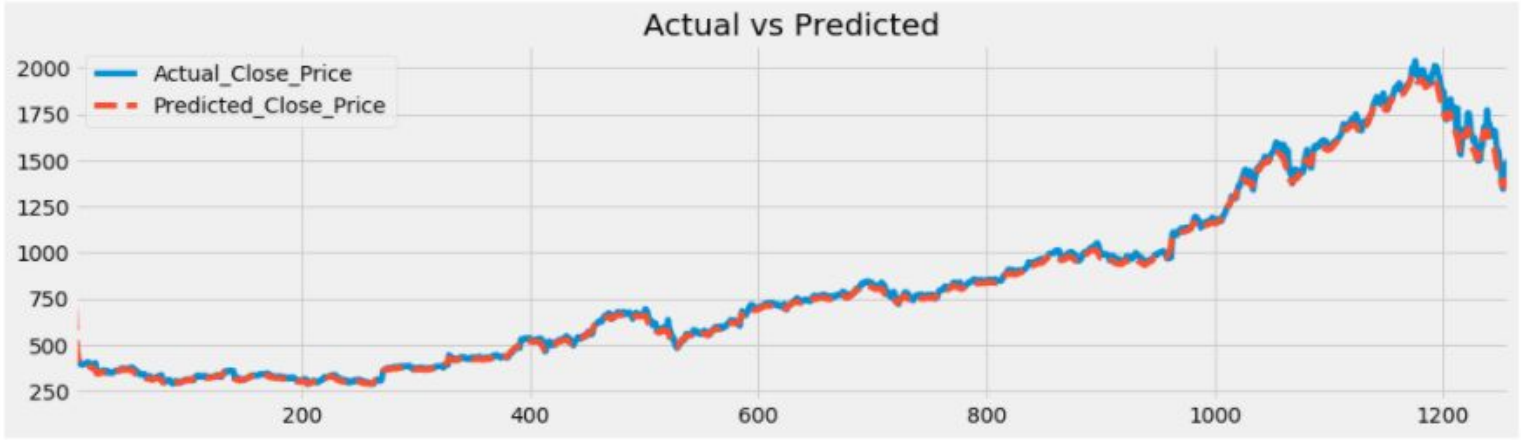
```
mean_returns = Strategy_Result['Returns'].mean()
sd = Strategy_Result['Returns'].std()
print(mean_returns,sd)
Market_RF = 0.0464


Sharpe_Ratio = np.sqrt(877)*(mean_returns)/sd
Sharpe_Ratio
```

```
0.002508106131459621 0.013314383147163463

5.5785927246438645
```

Buy & Hold Trading Strategy with Macd

```
Strategy_Result[['Actual_Close_Price','Predicted_Close_Price']].loc[0:].plot(figsize=(15,4),
                                                                             style=['-','--'],
                                                                             title = 'Actual vs Predicted')


Strategy_Result[['Amount($)']].loc[0:].plot(figsize=(15,4),
                                            style=['-g'],
                                            title = 'Total Amount')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d2deb11848>
```




```
Profit = (Amount[-1])/10000
print('Initial_Investment : 10000$')
print('Final Amount:',Amount[-1],'$')
print ('Profit_Percent:',Profit*100,'%')
```

```
Initial_Investment : 10000$
Final Amount: 13300.0 $
Profit_Percent: 133.0 %
```

```
Strategy_Result['Returns'] = Strategy_Result['Amount($)'].pct_change()
```

```
mean_returns = Strategy_Result['Returns'].mean()
sd = Strategy_Result['Returns'].std()
print(mean_returns,sd)
Market_RF = 0.0464


Sharpe_Ratio = np.sqrt(877)*(mean_returns)/sd
Sharpe_Ratio
```

```
0.0003075506419130009 0.012741104098963412

0.7148408629800209
```

Model Evaluation

- rmse

```
from sklearn.metrics import mean_squared_error
```

```
rms = sqrt(mean_squared_error(Strategy_Result['Actual_Close_Price'], Strategy_Result['Predicted_Close_Price']))
```

```
sqrt(rms)
```

[58]: 5.126630824290353

- r-square

```
from sklearn.metrics import r2_score

r2_score(Strategy_Result['Actual_Close_Price'], Strategy_Result['Predicted_Close_Price'])
```

[59]: 0.9969615467137605

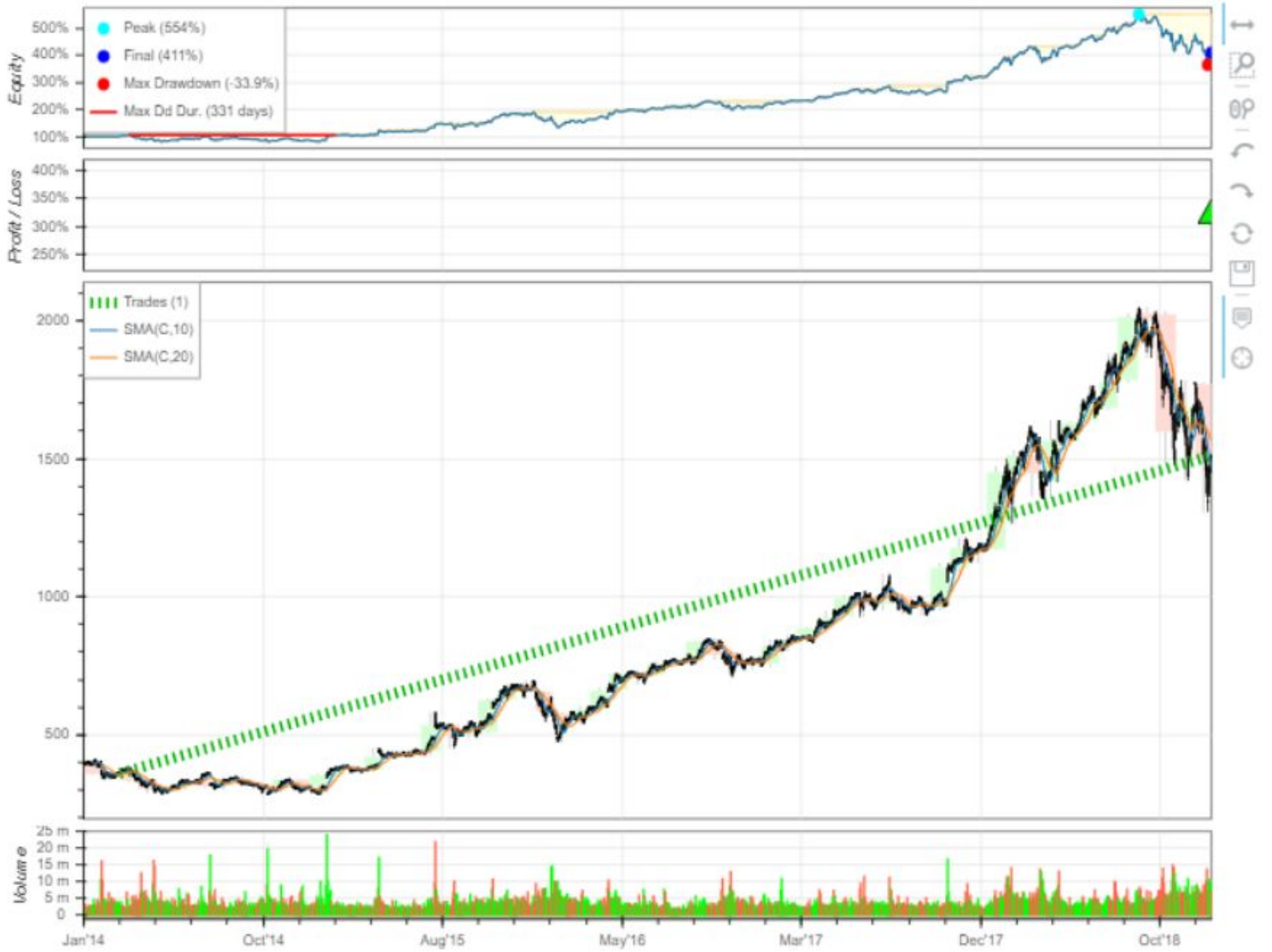
BackTesting

```python
from backtesting import Strategy
from backtesting.lib import crossover


class SmaCross(Strategy):

    # Define the two MA lags as *class variables*
    # for later optimization
    n1 = 10
    n2 = 20

    def init(self):
        # Precompute two moving averages
        self.sma1 = self.I(SMA, self.data.Close, self.n1)
        self.sma2 = self.I(SMA, self.data.Close, self.n2)

    def next(self):
        # If sma1 crosses above sma2, buy the asset
        if crossover(self.sma1, self.sma2):
            self.buy()

        # Else, if sma1 crosses below sma2, sell it
        elif crossover(self.sma2, self.sma1):
            self.sell()
```

```python
from backtesting import Backtest

bt = Backtest(df_amzn, SmaCross, cash=10000, commission=.002)
bt.run()
```

```
Start                     2014-01-02 00:00:00
End                       2018-12-31 00:00:00
Duration                   1824 days 00:00:00
Exposure Time [%]                    96.8203
Equity Final [$]                     41086.2
Equity Peak [$]                      55361.4
Return [%]                           310.862
Buy & Hold Return [%]                277.408
Return (Ann.) [%]                    32.7194
Volatility (Ann.) [%]                40.2231
Sharpe Ratio                        0.813447
Sortino Ratio                        1.68659
Calmar Ratio                        0.964541
Max. Drawdown [%]                   -33.9223
Avg. Drawdown [%]                   -3.87916
Max. Drawdown Duration      332 days 00:00:00
Avg. Drawdown Duration       24 days 00:00:00
# Trades                                   1
Win Rate [%]                             100
Best Trade [%]                         320.3
Worst Trade [%]                        320.3
Avg. Trade [%]                         320.3
Max. Trade Duration        1764 days 00:00:00
Avg. Trade Duration        1764 days 00:00:00
Profit Factor                             NaN
Expectancy [%]                            NaN
SQN                                       NaN
_strategy                           SmaCross
_equity_curve                             ...
_trades                      Size  EntryBa...
dtype: object
```

```python
bt.plot()
```

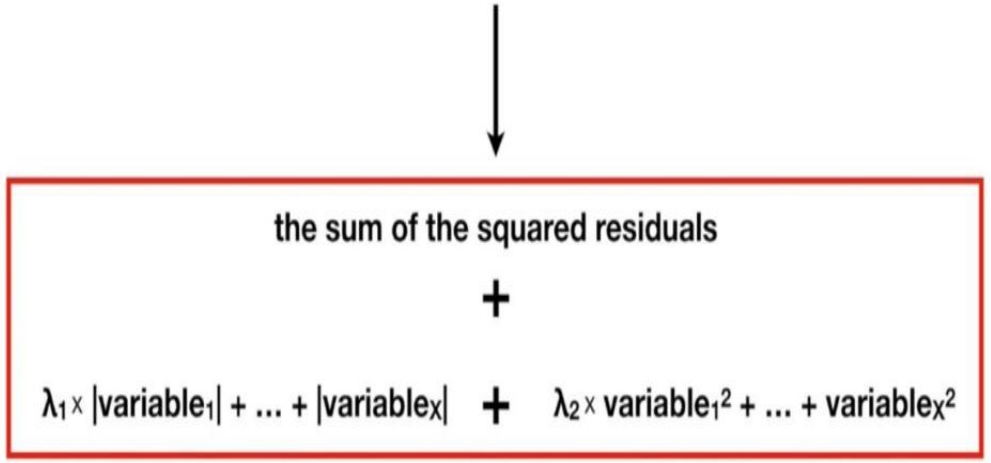


ELASTIC NET AND MONTE CARLO

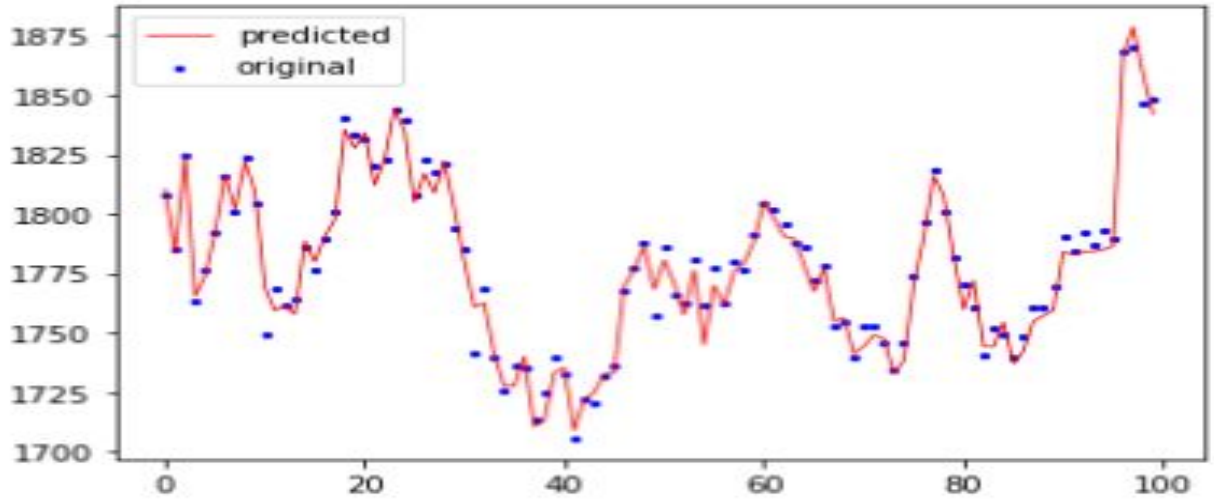
Elastic Net

```
alphas = [0.0001, 0.001, 0.01, 0.1, 0.3, 0.5, 0.7, 1]

for a in alphas:
    model = ElasticNet(alpha=a).fit(x,y)
    score = model.score(x, y)
    pred_y = model.predict(x)
    mse = mean_squared_error(y, pred_y)
    print("Alpha:{0:.4f}, R2:{1:.2f}, MSE:{2:.2f}, RMSE:{3:.2f}"
       .format(a, score, mse, np.sqrt(mse)))

elastic=ElasticNet(alpha=0.01).fit(xtrain, ytrain)
ypred = elastic.predict(xtest)
score = elastic.score(xtest, ytest)
mse = mean_squared_error(ytest, ypred)
print("R2:{0:.3f}, MSE:{1:.2f}, RMSE:{2:.2f}"
      .format(score, mse, np.sqrt(mse)))

x_ax = range(len(xtest))
plt.scatter(x_ax, ytest, s=5, color="blue", label="original")
plt.plot(x_ax, ypred, lw=0.8, color="red", label="predicted")
plt.legend()
plt.show()


# --- ElasticNetCV ----
elastic_cv=ElasticNetCV(alphas=alphas, cv=5)
model = elastic_cv.fit(xtrain, ytrain)
print(model.alpha_)
print(model.intercept_)

ypred = model.predict(xtest)
score = model.score(xtest, ytest)
mse = mean_squared_error(ytest, ypred)
print("R2:{0:.3f}, MSE:{1:.2f}, RMSE:{2:.2f}"
      .format(score, mse, np.sqrt(mse)))
```

Altogether, **Elastic Net Regression** combines the strengths of **Lasso** and **Ridge Regression**.



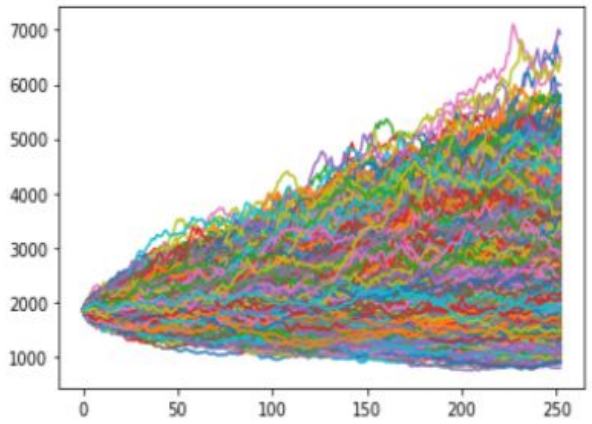

```
0.0001
5.132541525660713
R2:0.970, MSE:35.91, RMSE:5.99
```



Monte Carlo

- A class of techniques for randomly sampling a probability distribution
- All the possible outcomes of your decisions and assess the impact of risk, allowing for better decision making under uncertainty
- We are using a Monte Carlo simulation to look at the potential evolution of Amazon prices over time
- If we want to buy a Amazon stock, we may like to try to look into the future and attempt to predict what kind of returns we can expect with what kind of probability

```python
#Define Variables
S = amzn['Adj Close'][-1] #starting stock price (i.e. last available real stock price)
T = 252 #Number of trading days
mu = 0.2926 #Return
vol = 0.2979 #Volatility
#set up empty list to hold our ending values for each simulated price series
result = []
#choose number of runs to simulate - I have chosen 10,000
for i in range(10000):
    #create list of daily returns using random normal distribution
    daily_returns=np.random.normal(mu/T,vol/math.sqrt(T),T)+1

    #set starting price and create price series generated by above random daily returns
    price_list = [S]

    for x in daily_returns:
        price_list.append(price_list[-1]*x)

    #plot data from each individual run which we will plot at the end
    plt.plot(price_list)

    #append the ending value of each simulated run to the empty list we created at the beginning
    result.append(price_list[-1])

#show the plot of multiple price series created above
plt.show()

#create histogram of ending stock values for our mutliple simulations
plt.hist(result,bins=50)
plt.show()
```



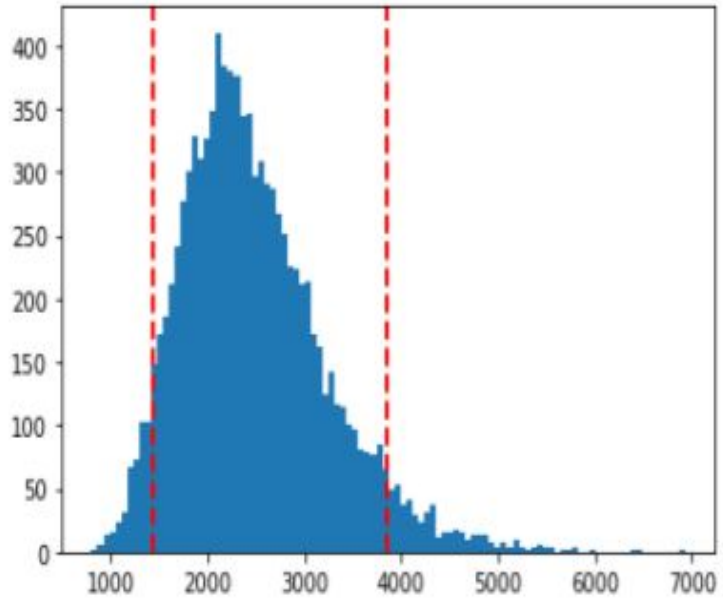

```python
#use numpy mean function to calculate the mean of the result
print(round(np.mean(result),2))
```

```
2478.78
```

```python
print("5% quantile =",np.percentile(result,5))
print("95% quantile =",np.percentile(result,95))
```

```
5% quantile = 1446.6098576904105
95% quantile = 3861.3311842152234
```

```python
plt.hist(result,bins=100)
plt.axvline(np.percentile(result,5), color='r', linestyle='dashed', linewidth=2)
plt.axvline(np.percentile(result,95), color='r', linestyle='dashed', linewidth=2)
plt.show()
```



GARCH

GARCH Model - analysis, explanation

GARCH is a statistical **model** that can be used to **analyze** a number of different types of financial data, for instance, macroeconomic data. Financial institutions typically use this **model** to estimate the volatility of returns for **stocks**, bonds, and market indices.

§The generalized autoregressive conditional heteroskedasticity (GARCH) process is an econometric term developed in 1982 by Robert F. Engle, an economist and 2003 winner of the Nobel Memorial Prize for Economics, to describe an approach to estimate volatility in financial markets.

§The general process for a GARCH model involves three steps. The first is to estimate a best-fitting autoregressive model. The second is to compute autocorrelations of the error term. The third step is to test for significance.

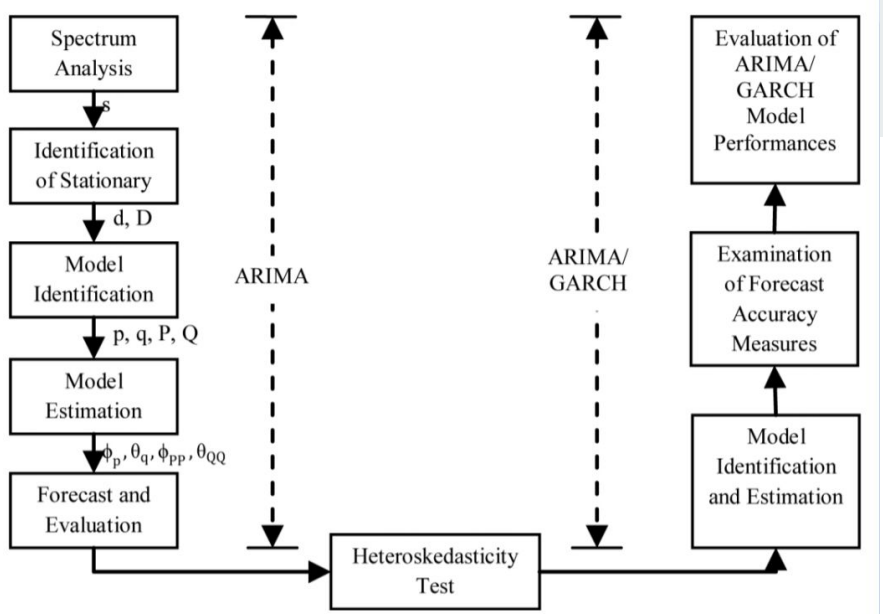
§An ARCH (AutoRegressive Conditionally Heteroskedasticity) model is a model for the variance of a time series. ARCH models are used to describe a changing, possibly volatile variance.

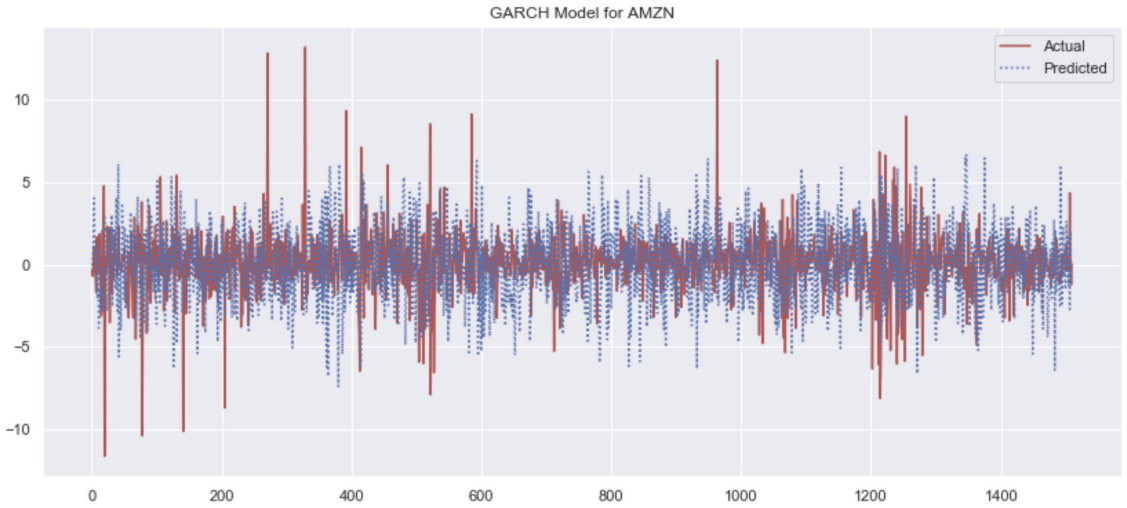
§If an Autoregressive Moving Average (ARMA) model is used for error variance, the model is a Generalized AutoRegressive Conditionally Heteroskedasticity (GARCH) model.

§$r_t = \mu + \sigma_t \varepsilon_t$

Where ε_t is the standard normal and is the σ_t variance.

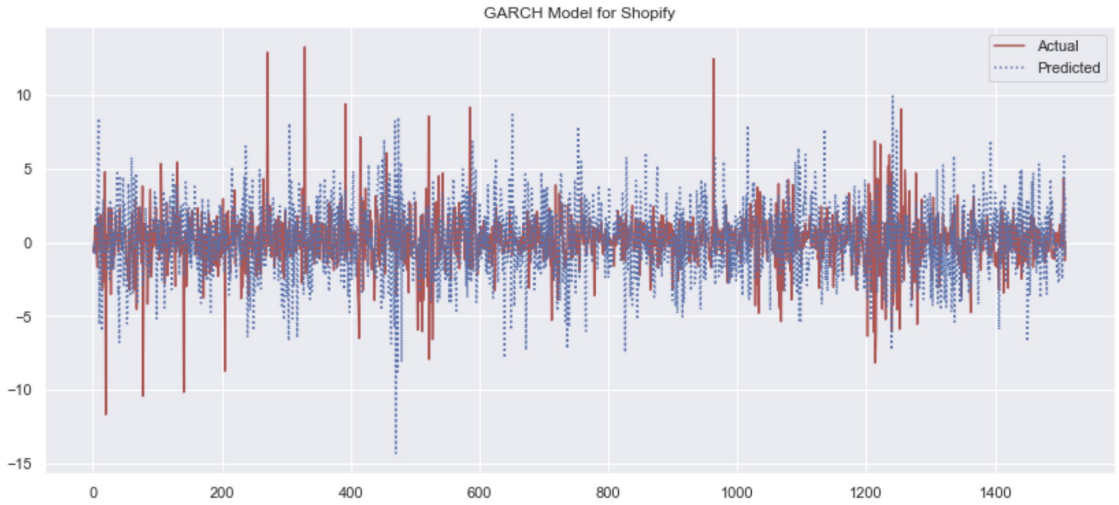
GARCH Model : AMZ , SHOP






```
rms = sqrt(mean_squared_error(dif['Actual'],dif['Predicted']))
rms
```

3.092914546632051




```
rms_shopify = sqrt(mean_squared_error(dif['Actual'],dif['Predicted']))
rms_shopify
```

3.2061166344295797

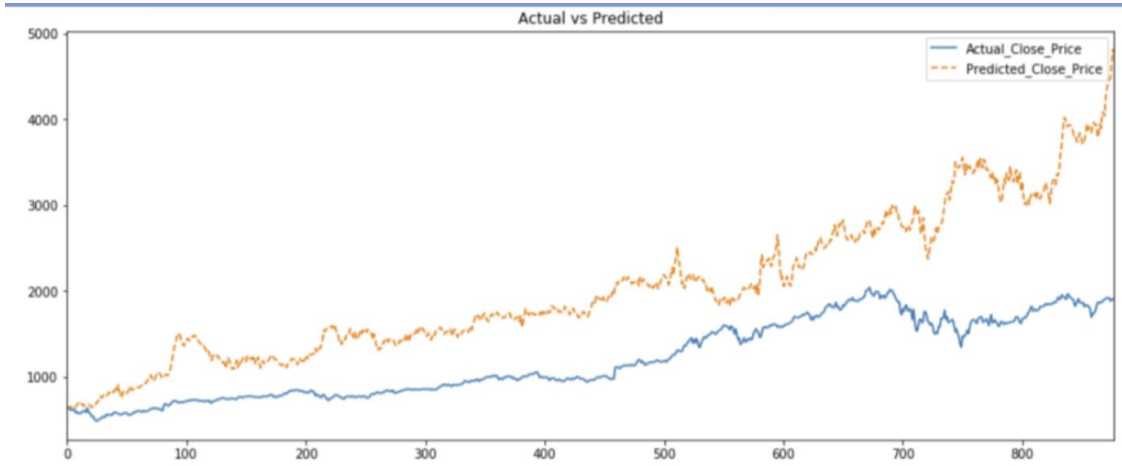
GARCH Model

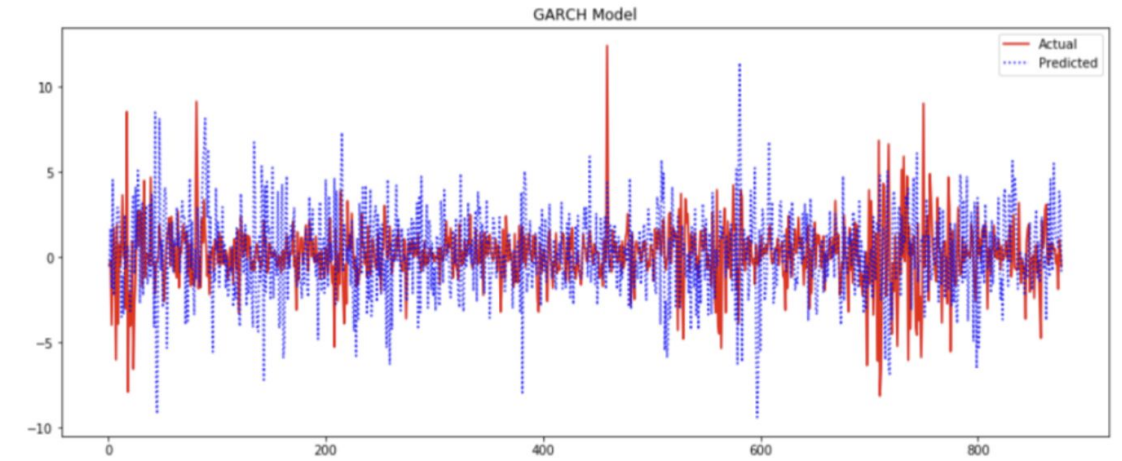
Volatility Measure against prediction

Checking Return volatility with actual and predicted values

Checking Return Volatility with Actual and Predicted Values¶

```
Predicted[['Actual_Close_Price','Predicted_Close_Price']].loc[0:].plot(figsize=(15,6), style=['-','--'],
                                                                       title = 'Actual vs Predicted')
fig= plt.figure(figsize=(15,6))
plt.plot(timevec, Y,'r-', label='Actual')
plt.plot(timevec, Y_GARCH,'b:', label='Predicted')
plt.legend(loc='upper right')
plt.title("GARCH Model")
plt.show()
```





Swing Trading Strategy for Garch

Buy if we have next day's predicted value greater than today's close value and hold if already bought

eg: day 1 : 100
day 2 : 200
…..BUY because value of stock is increasing profit , thus BOUGHT - HOLD

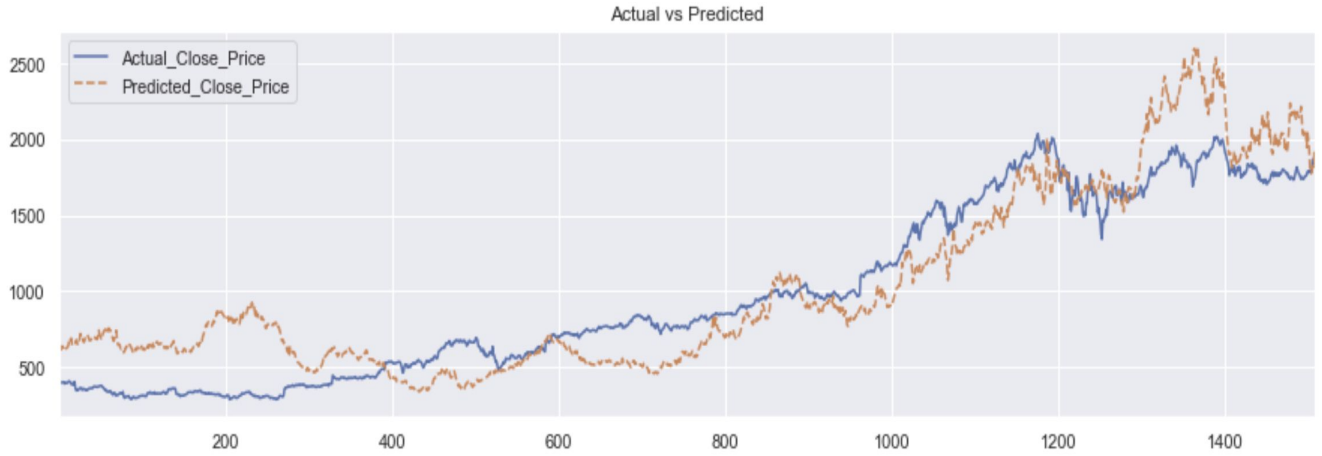
Sell if we have next days predicted value lesser t close value and don't buy until rule 1

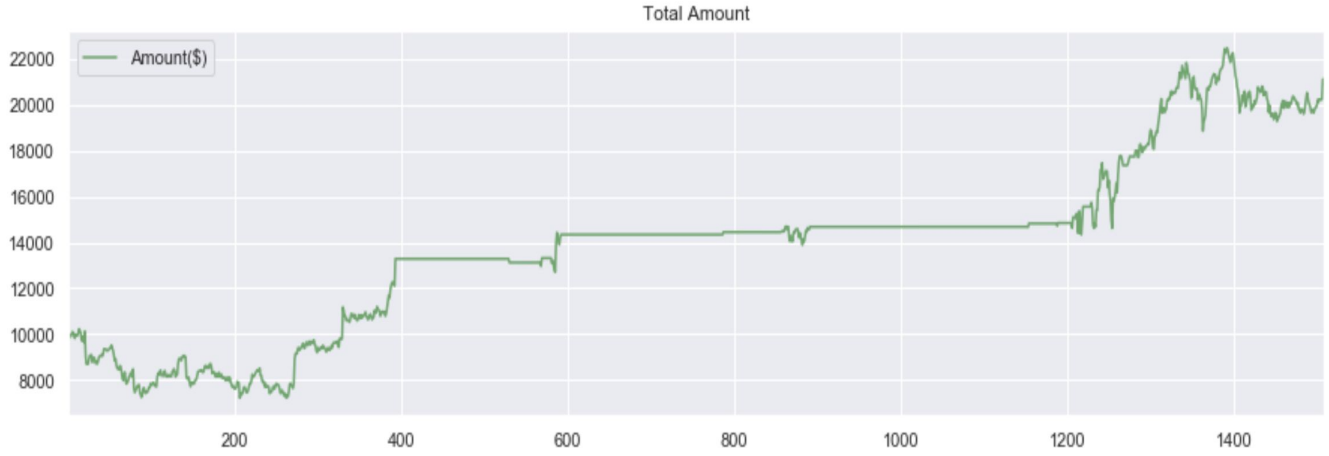
eg: day 1: 100
day 2 : 80
…..SELL because value of stock will decrease avoid loss, thus Price prediction already Lower

```
signal = 0
amount = 10000
Amount = []
balance = 0
action = []
portfolio = 0
Portfolio = []
stocks = 0
Stocks = []


for i in range(len(Predicted)-1):
    if Predicted['Predicted_Close_Price'][i+1] > Predicted['Actual_Close_Price'][i]:
        if signal == 0:
            action.append('Buy')
            stocks = int(amount / Predicted['Actual_Close_Price'][i])
            balance = int(amount % Predicted['Actual_Close_Price'][i])
            portfolio = stocks * Predicted['Actual_Close_Price'][i]
            signal = 1
            amount = portfolio + balance
            print('Stock:',Predicted['Actual_Close_Price'][i] ,'Action:',action[i],'Portfolio:',round(portfolio,2),'Stocks:'
            Portfolio.append(round(portfolio,5))
            Amount.append(round(amount,0))
            Stocks.append(stocks)
        else:
            action.append('Bought--Holding')
            portfolio = stocks * Predicted['Actual_Close_Price'][i]
            amount = portfolio + balance
            print('Stock:',Predicted['Actual_Close_Price'][i],'Action:',action[i],'Portfolio:',round(portfolio,2),'Stocks:',
            Portfolio.append(round(portfolio,5))
            Amount.append(round(amount,0))
            Stocks.append(stocks)

    elif Predicted['Predicted_Close_Price'][i+1] < Predicted['Actual_Close_Price'][i]:
        if signal == 1:
            action.append('Sell')
            portfolio = stocks * Predicted['Actual_Close_Price'][i]

            signal = 0
            stocks = 0
            amount = balance + portfolio
            portfolio = 0
            balance = 0
            print('Stock:',Predicted['Actual_Close_Price'][i],'Action:',action[i],'Portfolio:',round(portfolio,2),'Stocks:',
            Portfolio.append(round(portfolio,5))
            Amount.append(round(amount,0))
            Stocks.append(stocks)
        else:
            action.append('Price-Prediction-Already-Lower')
            print('Stock:',Predicted['Actual_Close_Price'][i],'Action:',action[i],'Portfolio:',round(portfolio,2),'Stocks:',
            Portfolio.append(round(portfolio,5))
            Amount.append(round(amount,0))
            Stocks.append(stocks)
```

Swing Trading Strategy for Garch (Sell the Model)

YESS!! we earned profit





Daily Strategy Movements

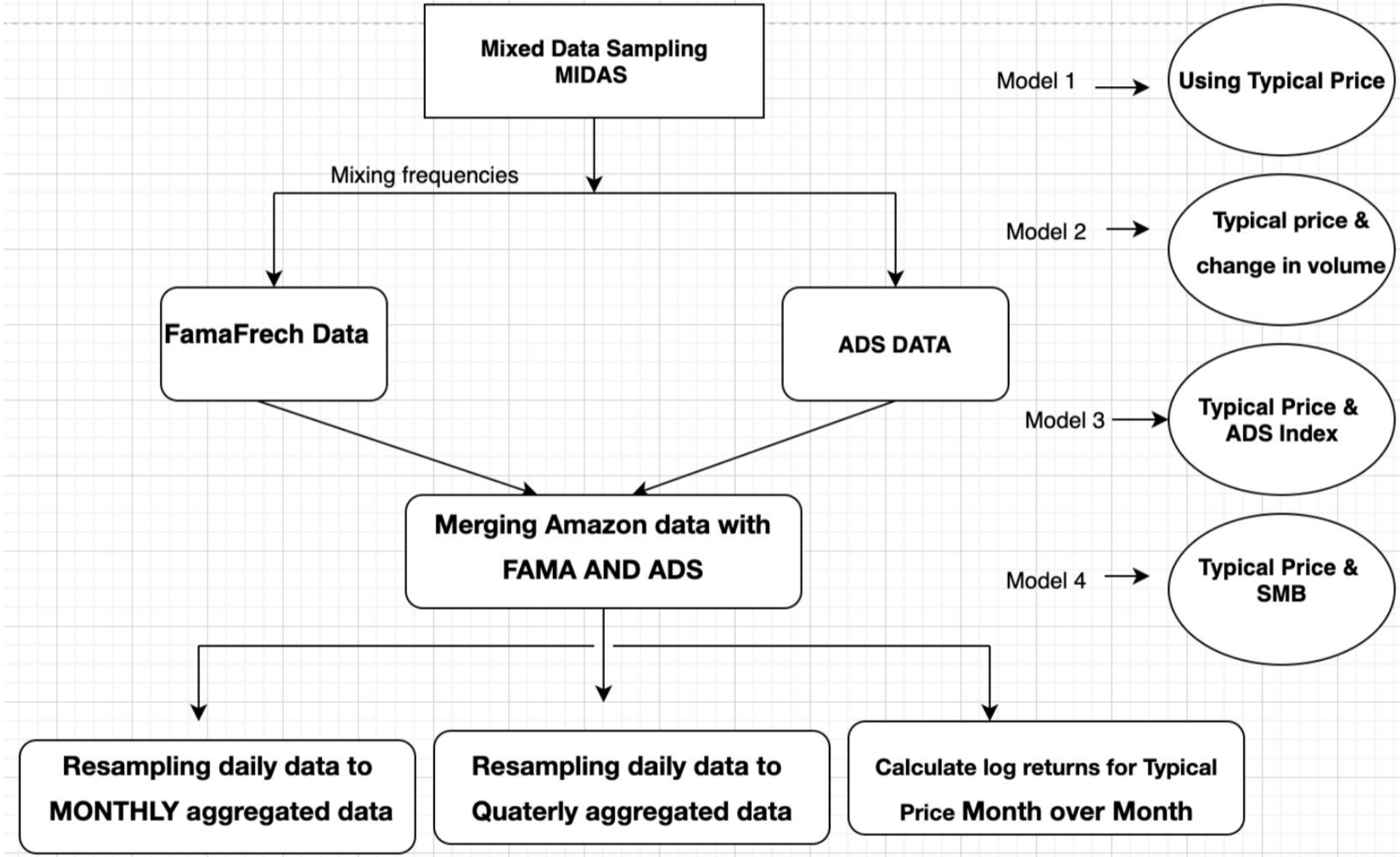
```
Stock: 397.9700012207031 Action: Buy Portfolio: 9949.25 Stocks: 25 Balance_init: 50 total($) 9999.25
Stock: 396.44000244140625 Action: Bought--Holding Portfolio: 9911.0 Stocks: 25 Balance_init: 50 total($) 9961.0
Stock: 393.6300048828125 Action: Bought--Holding Portfolio: 9840.75 Stocks: 25 Balance_init: 50 total($) 9890.75
Stock: 398.0299987792969 Action: Bought--Holding Portfolio: 9950.75 Stocks: 25 Balance_init: 50 total($) 10000.75
Stock: 401.9200134277344 Action: Bought--Holding Portfolio: 10048.0 Stocks: 25 Balance_init: 50 total($) 10098.0
```

```
Stock: 488.2699890136719 Action: Bought--Holding Portfolio: 12206.75 Stocks: 25 Balance_init: 50 total($) 12256.75
Stock: 482.17999267578125 Action: Bought--Holding Portfolio: 12054.5 Stocks: 25 Balance_init: 50 total($) 12104.5
Stock: 529.4199829101562 Action: Sell Portfolio: 0 Stocks: 0 Balance_init: 0 total($) 13285.5
Stock: 531.4099731445312 Action: Price-Prediction-Already-Lower Portfolio: 0 Stocks: 0 Balance_init: 0 total($) 13285.5
Stock: 526.0300292968750 Action: Price-Prediction-Already-Lower Portfolio: 0 Stocks: 0 Balance_init: 0 total($) 13285.5
```

Profit made with this strategy

```
Initial_Investment : 10000$
Final Amount: 20850.0 $
Profit_Percent: 208.5 %
```

MIDAS

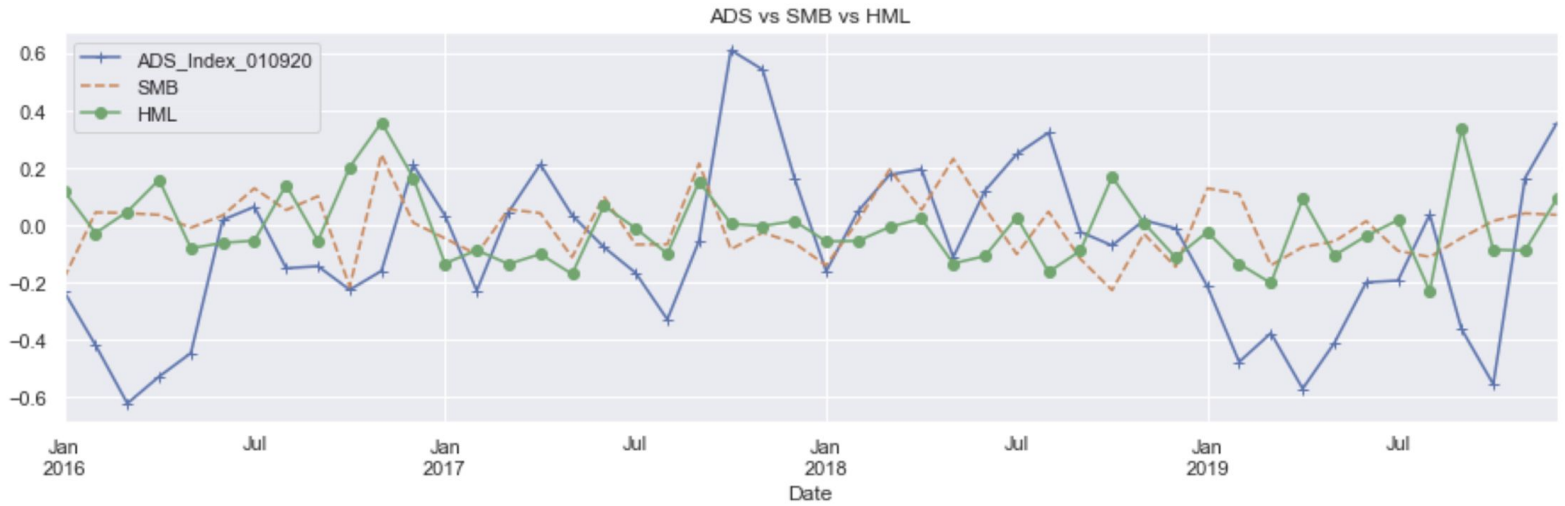

Mixed Data Sampling
MIDAS
Mixing frequencies
FamaFrech Data
ADS DATA
Merging Amazon data with
FAMA AND ADS
Resampling daily data to
MONTHLY aggregated data
Resampling daily data to
Quaterly aggregated data
Calculate log returns for Typical
Price Month over Month
Model 1
Using Typical Price
Model 2
Typical price &
change in volume
Model 3
Typical Price &
ADS Index
Model 4
Typical Price &
SMB

MIDAS -Performance Analysis

Line Charts to check ADS and Fama Factor Trends with Amazon Typical Price

In [52]:
```
month_agg[['ADS_Index_010920','SMB','HML']].loc['2016-01-04':].plot(figsize=(15,4), style=['-+','--','-o'],
                                                                    title = 'ADS vs SMB vs HML')
```

Out[52]: <matplotlib.axes._subplots.AxesSubplot at 0x228fbf20088>



We can see Amazon Trend is most similar to SMB index, so we would use these two as high frequency variables to predict AMZN quarterly returns

MIDAS - Trial Submodels

Model 1 : Using Typical Price

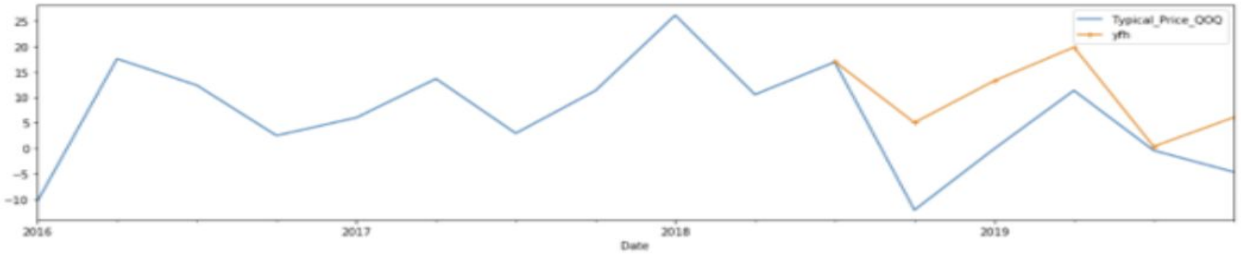
Model 2 : Considering Typical Price and Change in Volume

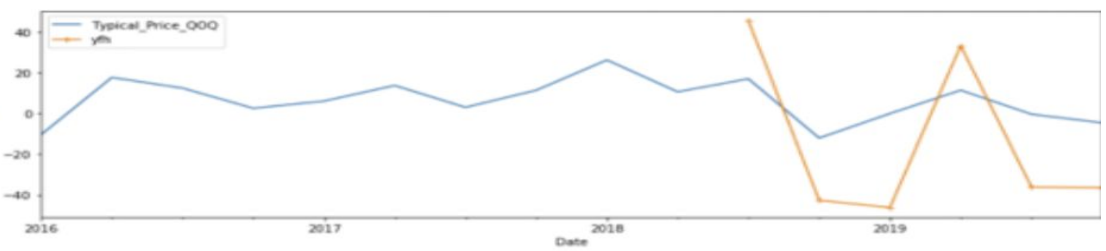
Model 3 : Considering Typical Price and ADS index model

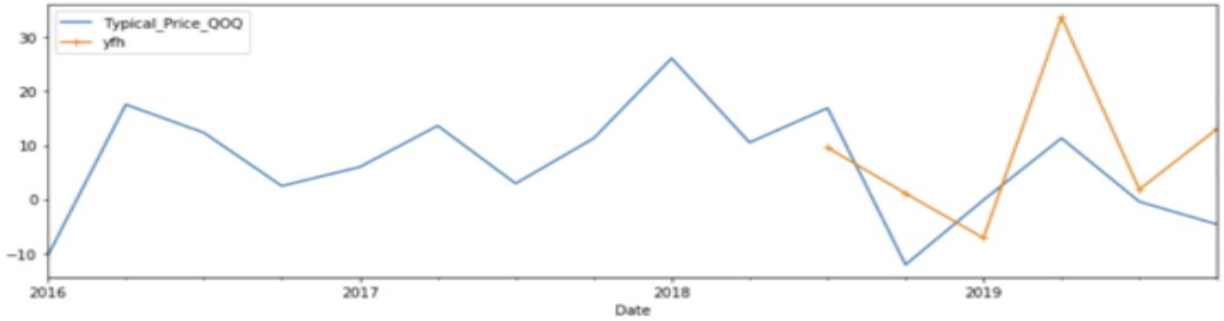
Model 4 : Using Typical Price and SMB

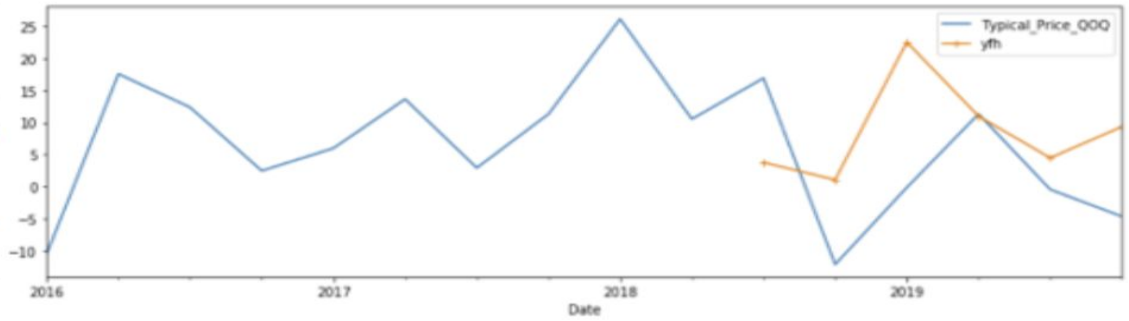
Model 5 : Checking for Optical Horizons in Model

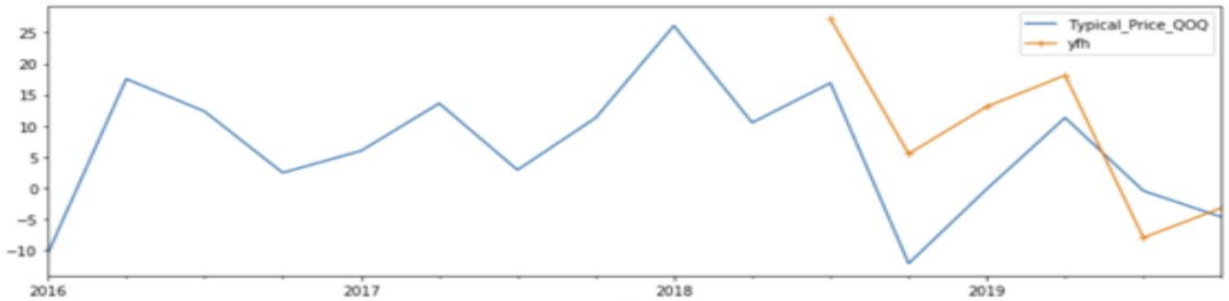
Model 6: Retaining it to 3 Horizon- Model 2

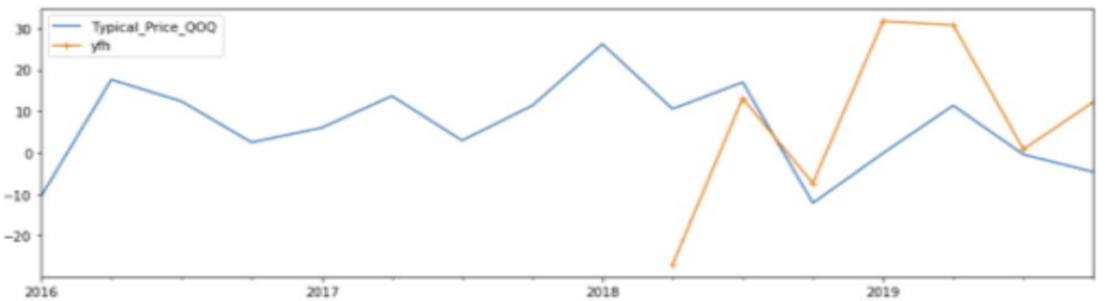












Swing Trading Strategy for MIDAS

```
amount = 10000
signal = 0
Amount = []
balance = 0
action = []
portfolio = 0
Portfolio = []
stocks = 0
Stocks = []
for i in range(len(Trading_df)-1):
    if Trading_df['Predicted_Typical_Price'][i+1] > Trading_df['Actual_Typical_Price'][i]:
        if signal == 0:
            action.append('Buy')
            stocks = int(amount / Trading_df['Actual_Typical_Price'][i])
            balance = int(amount % Trading_df['Actual_Typical_Price'][i])
            portfolio = stocks * Trading_df['Actual_Typical_Price'][i]
            signal = 1
            amount = portfolio + balance
            print('Stock:',Trading_df['Actual_Typical_Price'][i] ,'Action:',action[i],'Portfolio:',round(portfolio,2),'Stocks:',
            Portfolio.append(round(portfolio,5))
            Amount.append(round(amount,0))
            Stocks.append(stocks)
        else:
            action.append('Bought--Holding')
            portfolio = stocks * Trading_df['Actual_Typical_Price'][i]
            amount = portfolio + balance
            print('Stock:',Trading_df['Actual_Typical_Price'][i],'Action:',action[i],'Portfolio:',round(portfolio,2),'Stocks:',
            Portfolio.append(round(portfolio,5))
            Amount.append(round(amount,0))
            Stocks.append(stocks)

    elif Trading_df['Predicted_Typical_Price'][i+1] < Trading_df['Actual_Typical_Price'][i]:
        if signal == 1:
            action.append('Sell')
            portfolio = stocks * Trading_df['Actual_Typical_Price'][i]

            signal = 0
            stocks = 0
            amount = balance + portfolio
            portfolio = 0
            balance = 0
            print('Stock:',Trading_df['Actual_Typical_Price'][i],'Action:',action[i],'Portfolio:',round(portfolio,2),'Stocks:',
            Portfolio.append(round(portfolio,5))
            Amount.append(round(amount,0))
            Stocks.append(stocks)
        else:
            action.append('Price-Prediction-Already-Lower')
            print('Stock:',Trading_df['Actual_Typical_Price'][i],'Action:',action[i],'Portfolio:',round(portfolio,2),'Stocks:',
            Portfolio.append(round(portfolio,5))
            Amount.append(round(amount,0))
```

```
Stock: 1587.0684903462736 Action: Price-Prediction-Already-Lower Portfolio: 0 Stocks: 0 Balance_init: 0 total($) 10000
Stock: 1879.5161914320856 Action: Price-Prediction-Already-Lower Portfolio: 0 Stocks: 0 Balance_init: 0 total($) 10000
Stock: 1664.876507471478 Action: Price-Prediction-Already-Lower Portfolio: 0 Stocks: 0 Balance_init: 0 total($) 10000
Stock: 1662.7561261953554 Action: Buy Portfolio: 9976.54 Stocks: 6 Balance_init: 23 total($) 9999.54
Stock: 1862.1950761098706 Action: Bought--Holding Portfolio: 11173.17 Stocks: 6 Balance_init: 23 total($) 11196.17
Stock: 1853.943959554037 Action: Bought--Holding Portfolio: 11123.66 Stocks: 6 Balance_init: 23 total($) 11146.66
```

Date	Actual_Typical_Price	Predicted_Typical_Price
2018-06-30	1587.068490	1587.068490
2018-09-30	1879.516191	1212.266391
2018-12-31	1664.876507	1381.274349
2019-03-31	1662.756126	1283.178671
2019-06-30	1862.195076	1760.641880
2019-09-30	1853.943960	2392.955363
2019-12-31	1769.570102	2412.518988

LSTM

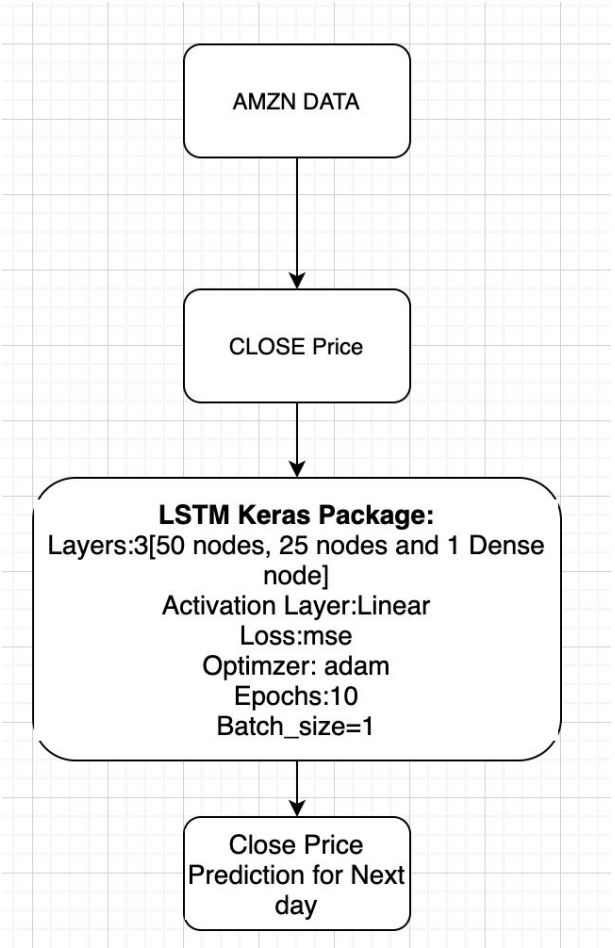
Long Short Term Memory

Definition: Long short-term memory is an artificial recurrent neural network architecture used in the field of deep learning.

Unlike standard feedforward neural networks, LSTM has feedback connections.

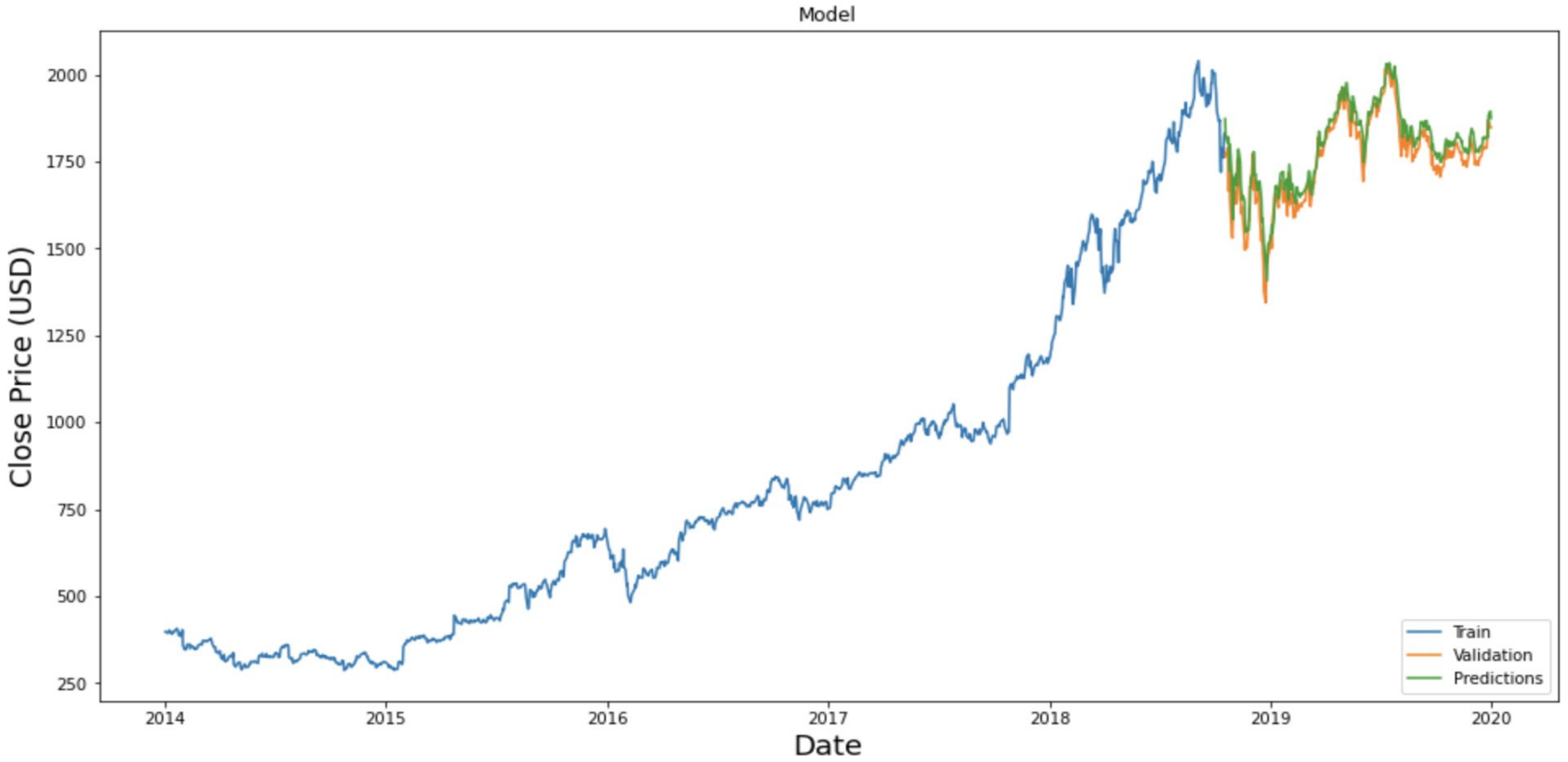
It can not only process single data points, but also entire sequences of data.

LSTMs are very powerful in sequence prediction problems because they are able to store the past information, this is important as we are predicting the future of the stock based on the previous closing price stock history.



Plot for LSTM Model Output

```
RMSE = {} 46.76465205537687
R-Squared Score = {} 0.836778766554912
```



OLS REGRESSION WITH FAMA FRENCH3

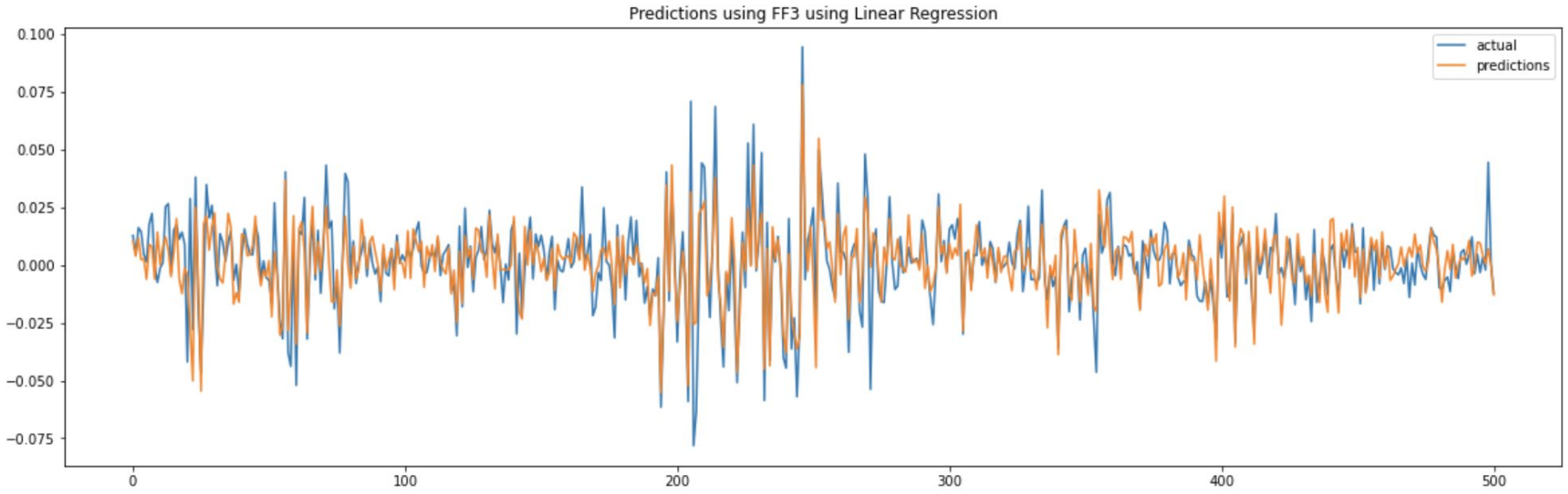
OLS Regression with Fama French3

Ordinary least squares (OLS) regression is a statistical method of analysis that estimates the relationship between one or more independent variables and a dependent variable.

The Fama and French model has three factors: size of firms, book-to-market values and excess return on the market. In other words, the three factors used are SMB (small minus big), HML (high minus low) and the portfolio's return less the risk free rate of return.

```
RMSE------- 0.011371359685062876
R-Squared-- 0.6428036133911343
```

```
 REGRESSION STATISTICS

------------------------


 Joint significance of all coefficients
 [array([[1.38929805e-09]]), array([[1.]])]
Beta Values
 [[ 0.00021121]
 [ 0.01361902]
 [-0.00031779]
 [-0.00881567]]
P values
 [[0.4926146  0.13148143 0.5055787  0.67020041]]
R-Square is
 [[0.64280361]]
Adjusted R Square
 [[0.6406475]]
Standard Error
 [[0.01137136]]
Observations
 501
-------------------------
```



RIDGE REGRESSION

Ridge Regression

Ridge Regression is a technique for analyzing multiple regression data that suffer from multicollinearity.
When multicollinearity occurs, least squares estimates are unbiased, but their variances are large so they may be far from the true value. By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors.
In ridge regression, you can tune the lambda parameter so that model coefficients change.

OLS treats all the variables equally (unbiased).
OLS model becomes more complex as new variables are added.

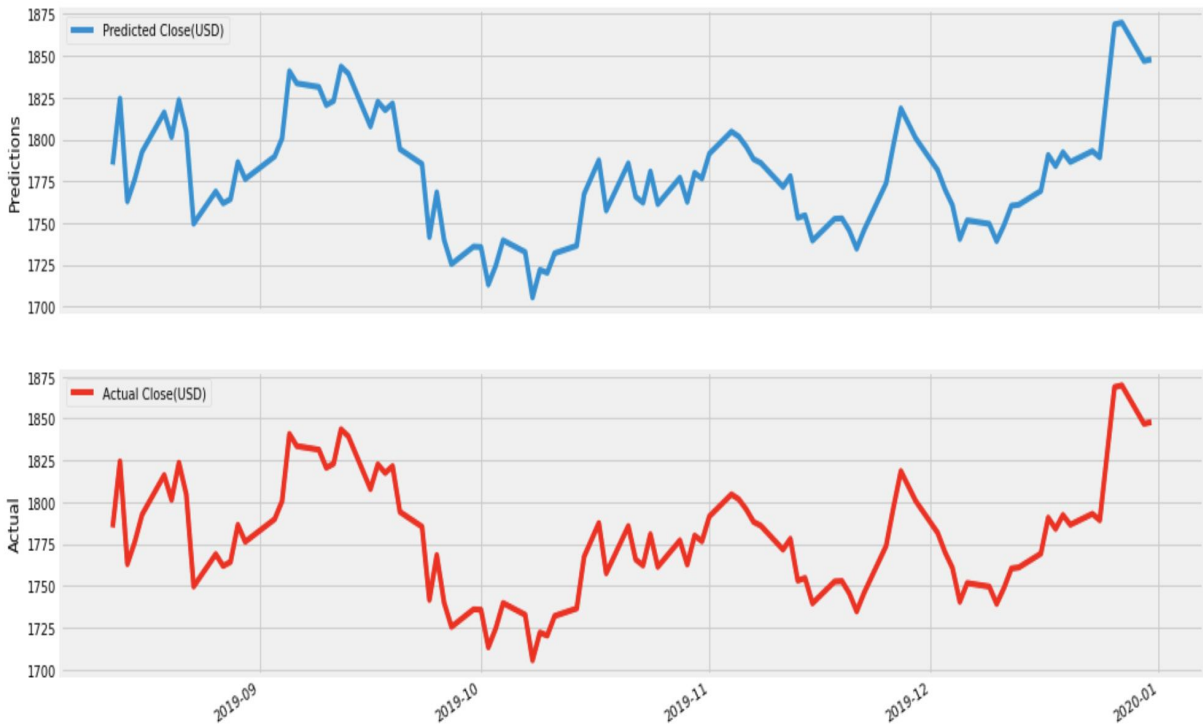
Why Ridge over OLS?

- Loss function ~ error made by model in comparison to real life values.
- Penalty methods are needed to make sure the coefficients to the predictors don't go out of bounds

Ridge Regression

In [9]:
```python
class RidgeRegression(object):
    def __init__(self, lmbda=0.1):
        self.lmbda = lmbda

    def fit(self, X, y):
        C = X.T.dot(X) + self.lmbda*np.eye(X.shape[1])
        self.w = np.linalg.inv(C).dot(X.T.dot(y))

    def predict(self, X):
        return X.dot(self.w)

    def get_params(self, deep=True):
        return {"lmbda": self.lmbda}

    def set_params(self, lmbda=0.1):
        self.lmbda = lmbda
        return self
```

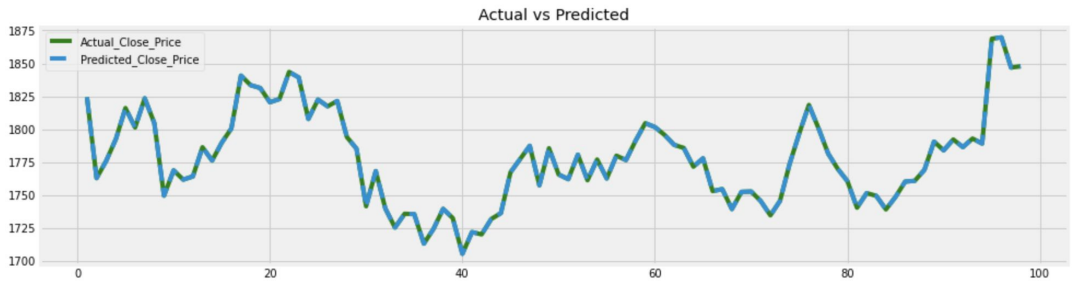
In [10]:
```python
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
ridge = RidgeRegression()
param_grid = [{"lmbda": 2.0**np.arange(-5, 10)}]
learner = GridSearchCV(ridge, param_grid, scoring="neg_mean_absolute_error", n_jobs=-1, verbose=0)
learner.fit(X_train, y_train)

y_pred = learner.predict(X_test)
```

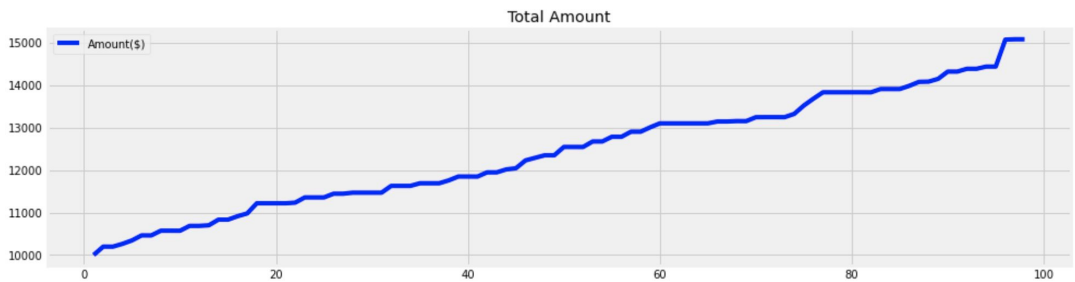


Ridge Regression- Trading Strategy

1 Buy & Hold Trading Strategy

- Position trading
- Buy if we have next days predicted_value greater than todays close value and hold if already bought
- Sell if we have next days predicted_value lesser than todays close value and dont buy until rule 1



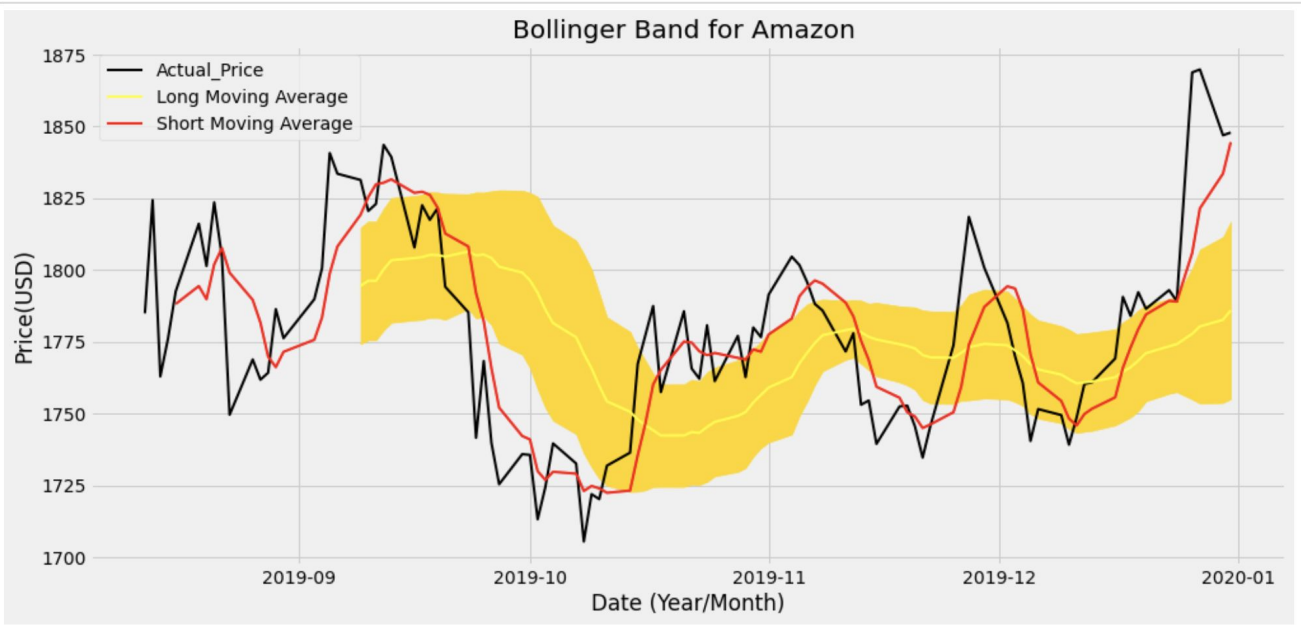



```
signal = 0
amount = 10000
Amount = []
balance = 0
action = []
portfolio = 0
Portfolio = []
stocks = 0
Stocks = []


for i in range(len(results)-1):
    if results['Predicted'][i+1] > results['Actual'][i]:
        if signal == 0:
            action.append('Buy')
            stocks = int(amount / results['Actual'][i])
            balance = int(amount % results['Actual'][i])
            portfolio = stocks * results['Actual'][i]
            signal = 1
            amount = portfolio + balance
            print('Stock:',results['Actual'][i] ,'Action:',action[i],'Portfolio:',round(portfolio,2),'Stocks:', stocks,
            Portfolio.append(round(portfolio,5))
            Amount.append(round(amount,0))
            Stocks.append(stocks)
        else:
            action.append('Bought--Holding')
            portfolio = stocks * results['Actual'][i]
            amount = portfolio + balance
            print('Stock:',results['Actual'][i],'Action:',action[i],'Portfolio:',round(portfolio,2),'Stocks:', stocks,'
            Portfolio.append(round(portfolio,5))
            Amount.append(round(amount,0))
            Stocks.append(stocks)

    elif results['Predicted'][i+1] < results['Actual'][i]:
        if signal == 1:
            action.append('Sell')
            portfolio = stocks * results['Actual'][i]

            signal = 0
            stocks = 0
            amount = balance + portfolio
            portfolio = 0
            balance = 0
            print('Stock:',results['Actual'][i],'Action:',action[i],'Portfolio:',round(portfolio,2),'Stocks:', stocks,'
            Portfolio.append(round(portfolio,5))
            Amount.append(round(amount,0))
            Stocks.append(stocks)
        else:
            action.append('Price-Prediction-Already-Lower')
            print('Stock:',results['Actual'][i],'Action:',action[i],'Portfolio:',round(portfolio,2),'Stocks:', stocks,'
            Portfolio.append(round(portfolio,5))
            Amount.append(round(amount,0))
            Stocks.append(stocks)

    print('\n')
```

Ridge Regression- Trading Strategy Results

	Actual_Close_Price	Predicted_Close_Price	Date	Action	Stocks	Portfolio($)	Amount($)
94	1789.21	1789.210103	2019-12-24	Sell	0	0.00	14443.0
95	1868.77	1868.769986	2019-12-26	Buy	8	14313.68	14443.0
96	1869.80	1869.800402	2019-12-27	Bought--Holding	8	14950.16	15079.0
97	1846.89	1846.890280	2019-12-30	Sell	0	0.00	15087.0
98	1847.84	1847.840004	2019-12-31	Buy	8	14775.12	15087.0

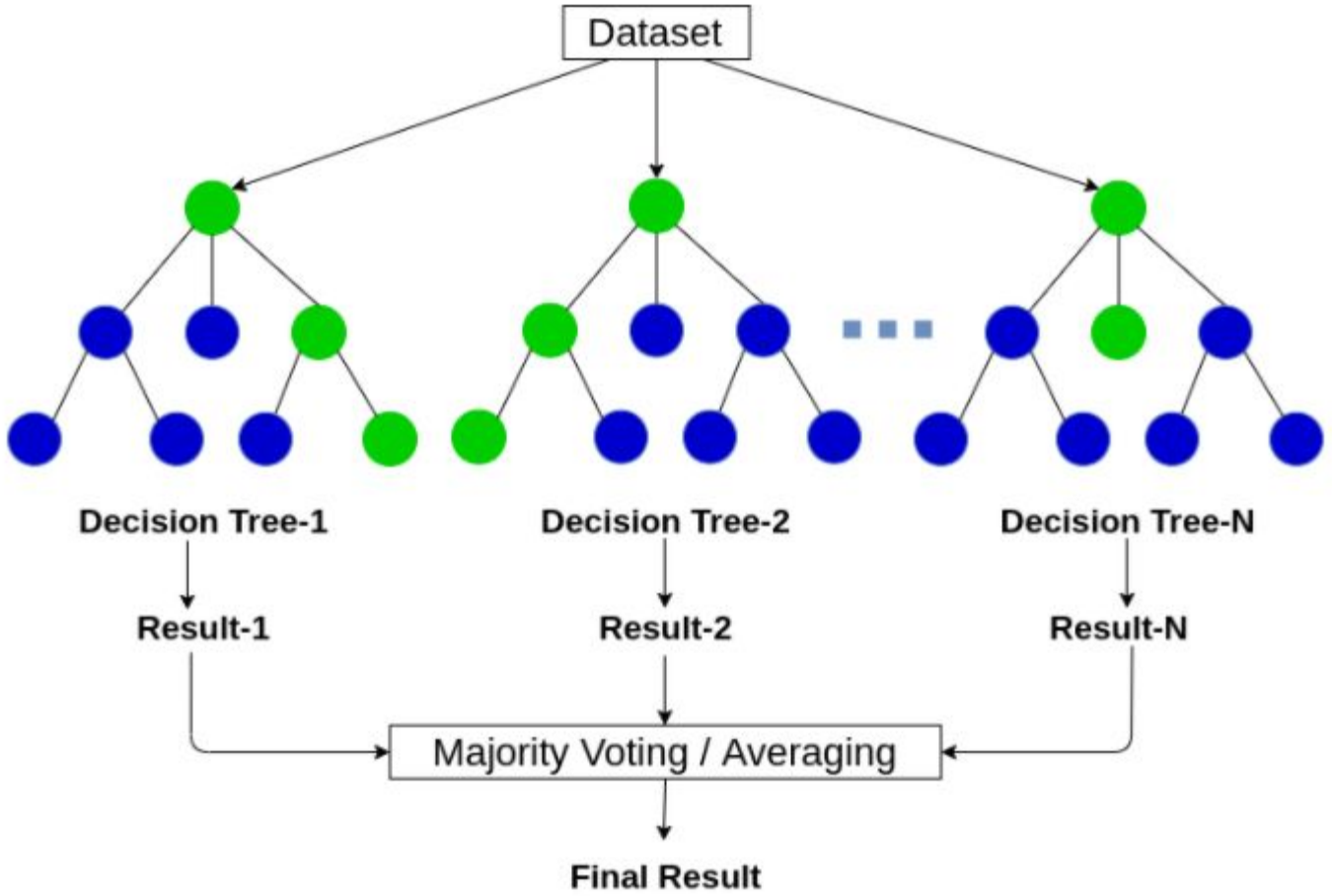
```
15087.0
Initial_Investment : 10000$
Final Amount: 15087.0 $
Profit_Percent: 50.870000000000005 %
```



RANDOM FOREST

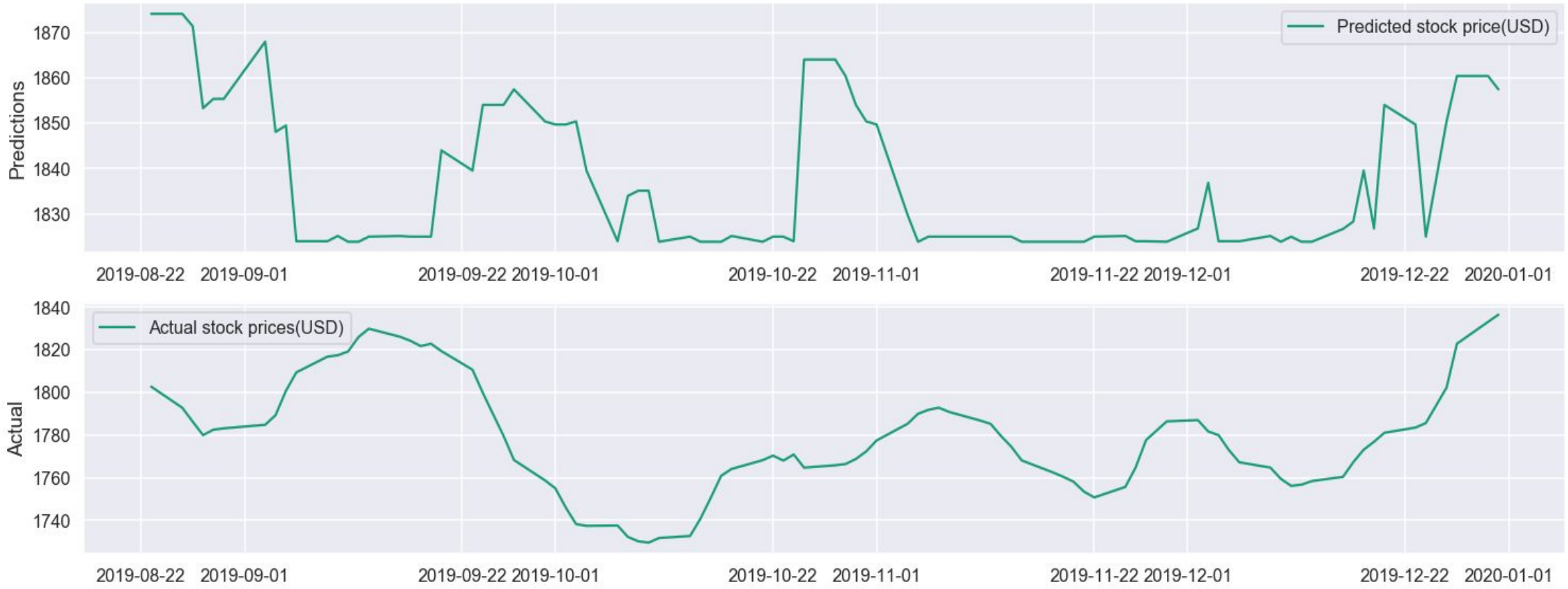
Random Forest is a tree-based machine learning algorithm that leverages the power of multiple decision trees for making decisions. As the name suggests, it is a “forest” of trees!

It is a forest of randomly created decision trees. Each node in the decision tree works on a random subset of features to calculate the output. The random forest then combines the output of individual decision trees to generate the final output.



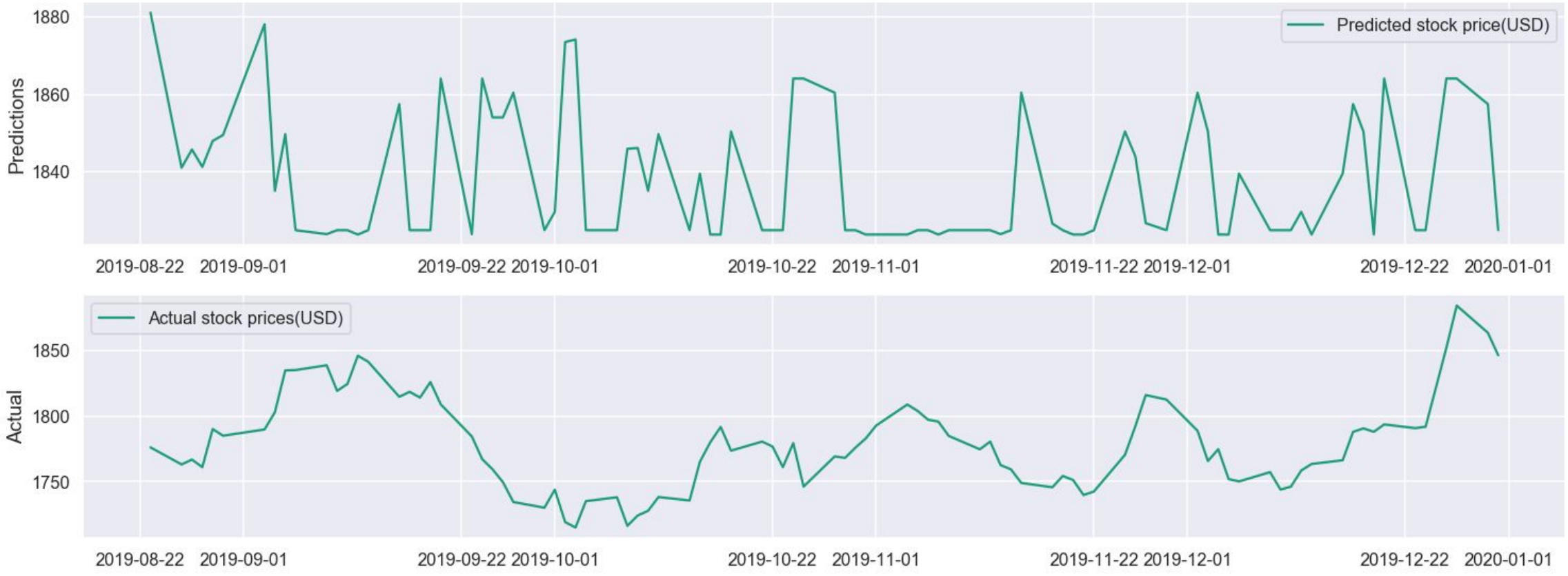
Random Forest-Decision Tree Model

Comparing predicted and actual values-for 90 days



Random Forest-Moving Averages model

Comparing predicted and actual values-for 90 day

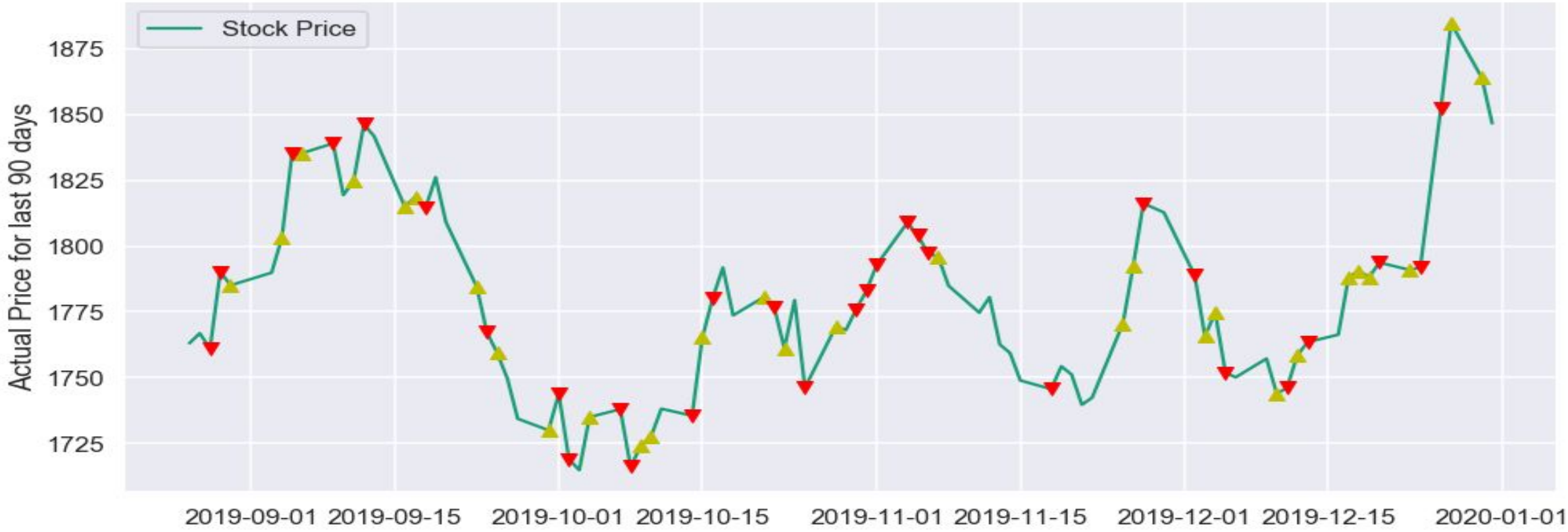


Trading Strategy

```
fig, (ax2) = plt.subplots(1, 1, figsize=(16,7))
duration = 89
y_temp = pd.DataFrame(y_short.tail(duration),index =X.tail(duration).index)
ax2.plot(X.tail(duration).index,y.tail(duration),label="Stock Price")

ax2.set_ylabel("Actual Price for last 90 days")

ax2.legend(loc='best')

# # Plot the buy signals
ax2.plot(trading_signal_week.tail(duration).loc[trading_signal_week.signal == 1.0].index,
         y.tail(duration)[trading_signal_week.signal == 1.0],'^', markersize=10, color='y')

# Plot the sell signals
ax2.plot(trading_signal_week.tail(duration).loc[trading_signal_week.signal == -1.0].tail(duration).index,
         y.tail(duration)[trading_signal_week.signal == -1.0],'v', markersize=10, color='r')
```

Trading Signal for last 90 days:

Red ^: Sell Signals
Yellow^: Buy Signals



Portfolio

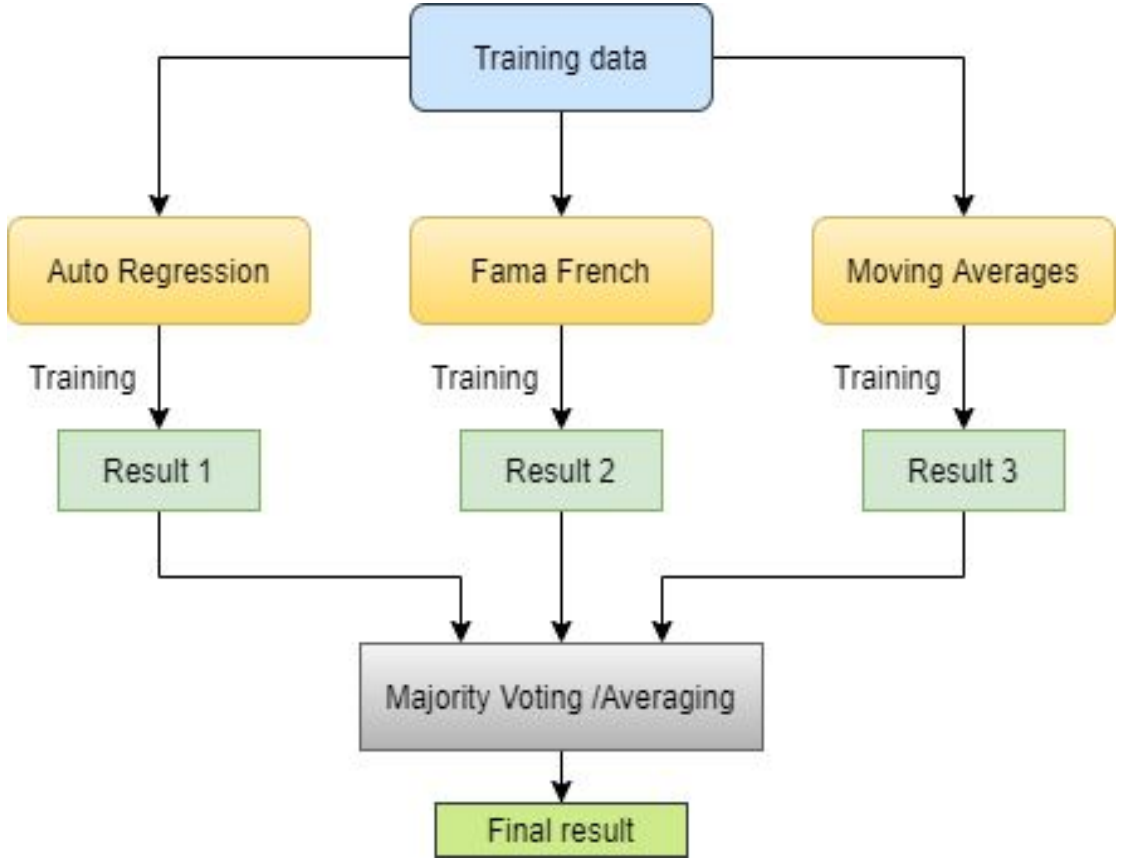
```
# Set the initial capital
initial_capital= float(10000.0)

# Create a DataFrame 'positions'
positions = pd.DataFrame(index=trading_signal_week.index).fillna(0.0)

# Buy
positions['stock_price'] = trading_signal_week['signal']

# Initialize the portfolio with value owned
portfolio = positions.multiply(y_short.tail(90), axis=0)

# Store the difference in shares owned
pos_diff = positions.diff()

# Add 'holdings' to portfolio
portfolio['holdings'] = (positions.multiply(y_short.tail(90), axis=0)).sum(axis=1)

# Add 'cash' to portfolio
portfolio['cash'] = initial_capital - (pos_diff.multiply(y_short.tail(90), axis=0)).sum(axis=1).cumsum()

# Add 'total' & 'returns' to portfolio
portfolio['total'] = portfolio['cash'] + portfolio['holdings']

portfolio['returns'] = portfolio['total'].pct_change()
```

Total of $11887.312 for the initial value of $1000

Model Combination

1. Auto Regression
2. Fama French
3. Moving Averages

Computed for 1Y and 5Y

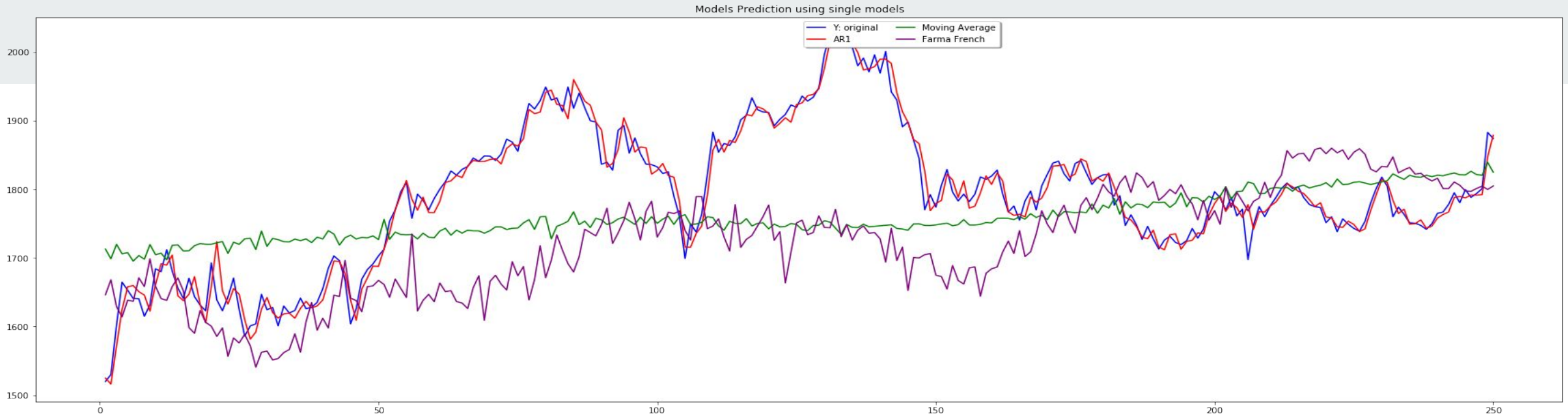


Predictions for 1 year's data-2019-2020

```
final_results
```

	Y	Y_rf	Y_close
0	1520.010010	1524.949833	1500.280029
1	1530.000000	1516.412440	1575.390015
2	1602.310059	1571.936993	1629.510010
3	1664.689941	1624.723042	1656.579956
4	1652.979980	1658.195009	1659.420044
...	...	...	...
245	1788.260010	1791.488294	1793.000000
246	1793.810059	1791.696327	1789.209961
247	1801.010010	1803.845828	1868.770020
248	1882.920044	1847.111292	1869.800049
249	1874.000000	1878.560846	1846.890015

```
17.098443470541543
124.7508085461156
104.95610454435186
ar1_RMSE:  17.098443470541543
ma_RMSE:  104.95610454435186
FF_RMSE:  124.7508085461156
random forest rmse:  15.334507369265808
```


Models Prediction using single models
Y: original
AR1
Moving Average
Farma French
2000
1900
1800
1700
1600
1500
0
50
100
150
200
250


Random Forest Prediction
Y: original
Y_rf
2000
1900
1800
1700
1600
1500
0
50
100
150
200
250

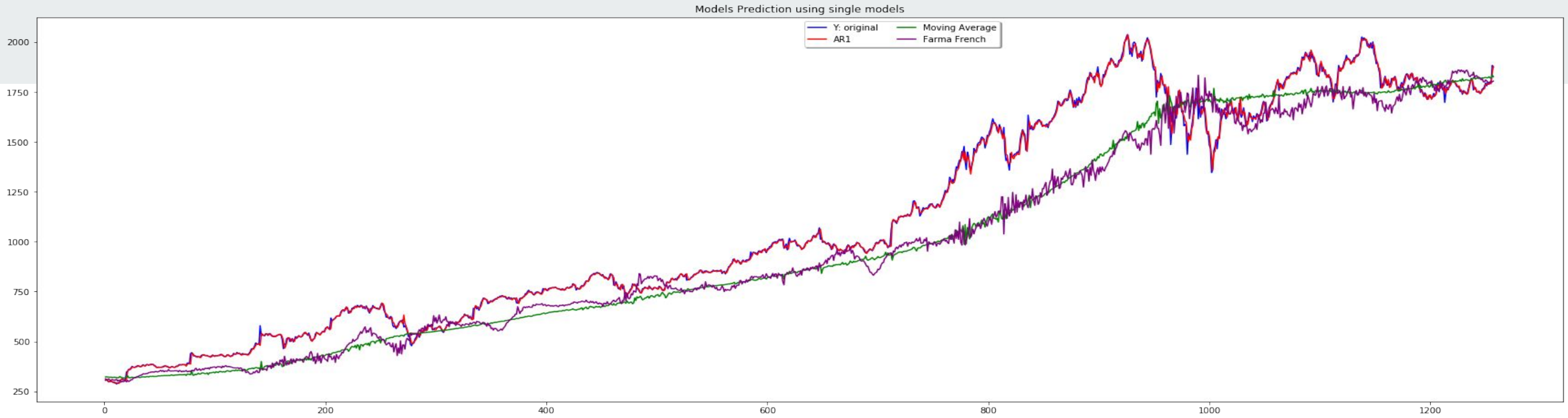
Predictions for 5 years' data 2014-2020

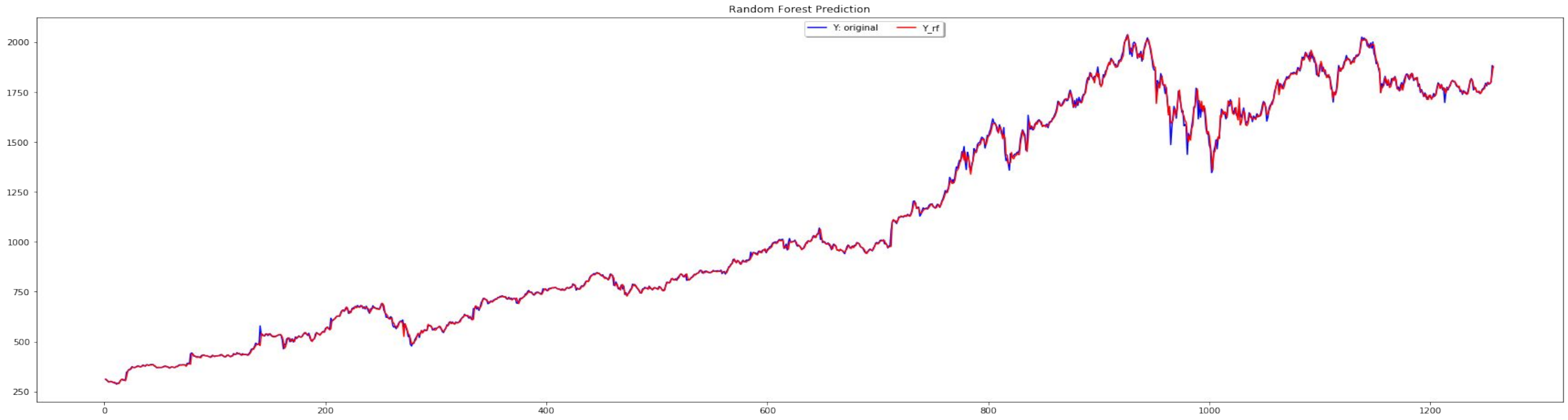
```
In [60]: final_results
```

Out[60]:

	Y	Y_rf	Y_close
0	312.579987	310.901093	308.519989
1	307.010010	309.110986	302.190002
2	302.239990	303.679216	295.290009
3	297.500000	297.002240	298.420013
4	300.320007	298.517109	300.459991
...	...	...	...
1252	1788.260010	1791.623123	1793.000000
1253	1793.810059	1791.556574	1789.209961
1254	1801.010010	1803.682564	1868.770020
1255	1882.920044	1847.082728	1869.800049
1256	1874.000000	1878.681590	1846.890015

```
15.724308727952726
187.19139489375146
180.61188019016862
ar1_RMSE:  15.724308727952726
ma_RMSE:  180.61188019016862
FF_RMSE:  187.19139489375146
random forest rmse:  14.473664297392956
```

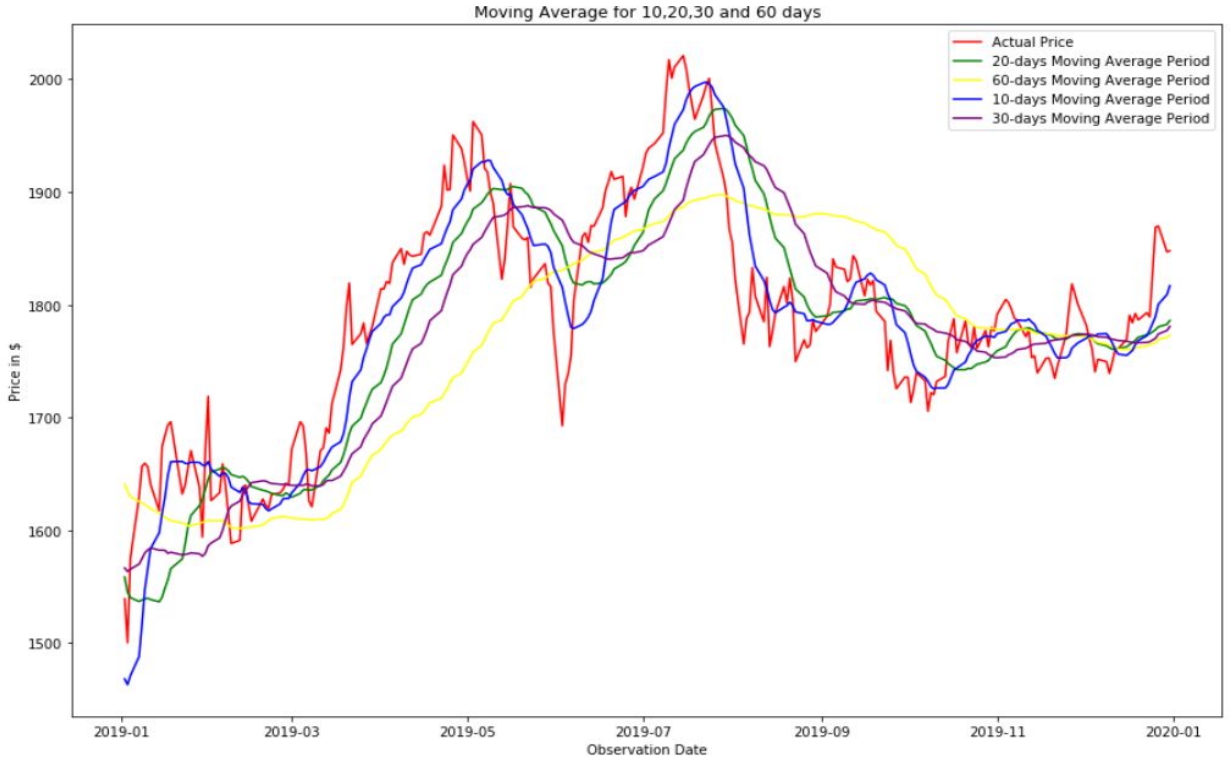

Models Prediction using single models
Y: original
AR1
Moving Average
Farma French
2000
1750
1500
1250
1000
750
500
250
0
200
400
600
800
1000
1200


Random Forest Prediction
Y: original
Y_rf
2000
1750
1500
1250
1000
750
500
250
0
200
400
600
800
1000
1200

MOVING AVERAGE

Simple Moving Average

MA= Sum of stocks for specific days/ Total no. of days




```
start_date = '2019-01-01'
end_date = '2019-12-31'

fig, ax = plt.subplots(figsize=(15,10))

ax.plot(amzn.loc[start_date:end_date, :].index, amzn.loc[start_date:end_date, 'Close'], label='Actual Price', color = 'red')
ax.plot(amzn.loc[start_date:end_date, :].index, amzn.loc[start_date:end_date, '20d_sma'], label = '20-days Moving Average Period'
ax.plot(amzn.loc[start_date:end_date, :].index, amzn.loc[start_date:end_date, '60d_sma'], label = '60-days Moving Average Period'
ax.plot(amzn.loc[start_date:end_date, :].index, amzn.loc[start_date:end_date, '10d_sma'], label = '10-days Moving Average Period'
ax.plot(amzn.loc[start_date:end_date, :].index, amzn.loc[start_date:end_date, '30d_sma'], label = '30-days Moving Average Period'
ax.legend(loc='best')
ax.set_title('Moving Average for 10,20,30 and 60 days')
ax.set_xlabel('Observation Date')
ax.set_ylabel('Price in $')
```

Exponential Moving Average

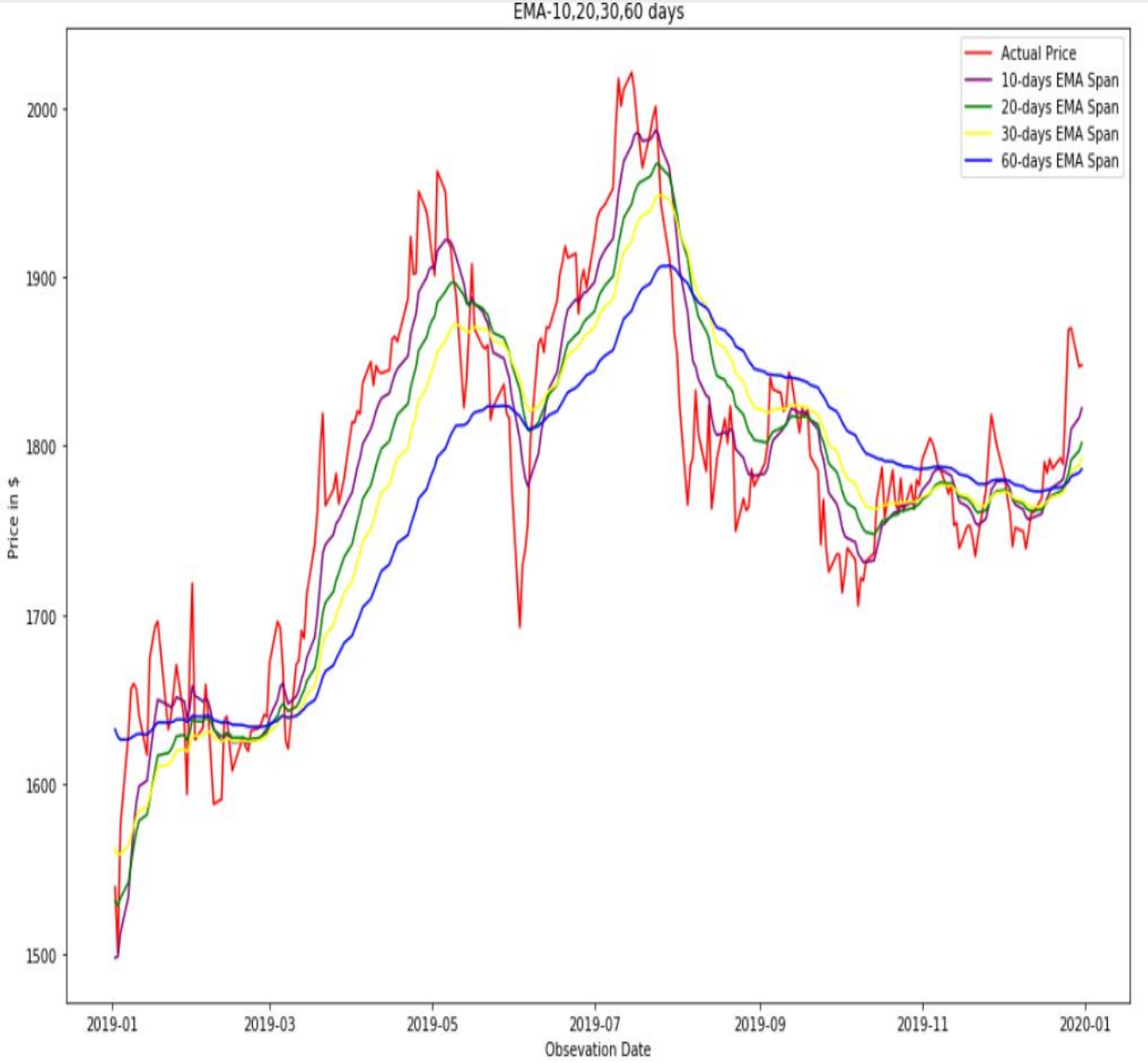
The Formula for EMA Is

$$EMA_{\text{Today}} = \left(\text{Value}_{\text{Today}} * \left(\frac{\text{Smoothing}}{1+\text{Days}}\right)\right) + EMA_{\text{Yesterday}} * \left(1 - \left(\frac{\text{Smoothing}}{1+\text{Days}}\right)\right)$$

where:

$EMA = \text{Exponential moving average}$

```
# ewm(exponential weighted) function in pandas which helps to find exponent
ema_10= amzn.ewm(span=10, adjust=False).mean()
ema_20= amzn.ewm(span=20, adjust=False).mean()
ema_30= amzn.ewm(span=30, adjust=False).mean()
ema_60 = amzn.ewm(span=60, adjust=False).mean()

fig, ax = plt.subplots(figsize=(15,10))

ax.plot(amzn.loc[start_date:end_date, :].index, amzn.loc[start_date:end_date, 'Close'], label='Actual Price', color = 'red')
ax.plot(ema_10.loc[start_date:end_date, :].index, ema_10.loc[start_date:end_date, 'Close'], label = '10-days EMA Span', color =
ax.plot(ema_20.loc[start_date:end_date, :].index, ema_20.loc[start_date:end_date, 'Close'], label = '20-days EMA Span', color =
ax.plot(ema_30.loc[start_date:end_date, :].index, ema_30.loc[start_date:end_date, 'Close'], label = '30-days EMA Span', color =
ax.plot(ema_60.loc[start_date:end_date, :].index, ema_60.loc[start_date:end_date, 'Close'], label = '60-days EMA Span', color =

ax.legend(loc='best')
ax.set_title('EMA-10,20,30,60 days')
ax.set_xlabel('Obsevation Date')
ax.set_ylabel('Price in $')
```



SMA VS EMA

OBSERVATION

- EMA is closer to Actual than SMA as, SMA applies same weight to all, while EMA applies greater weights to recent data points

Trading Strategy

- **Buy**

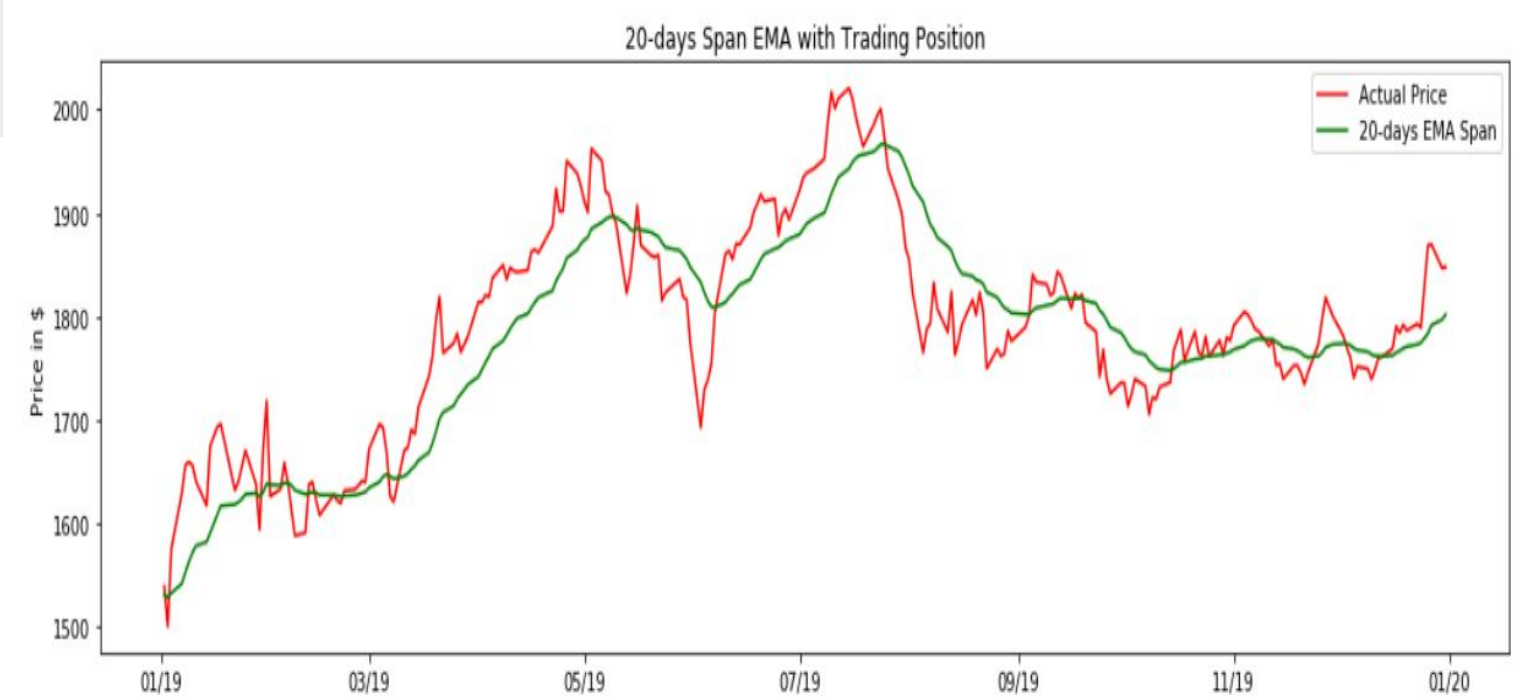
if Actual[‘Close’] >Predicted[‘Close’]

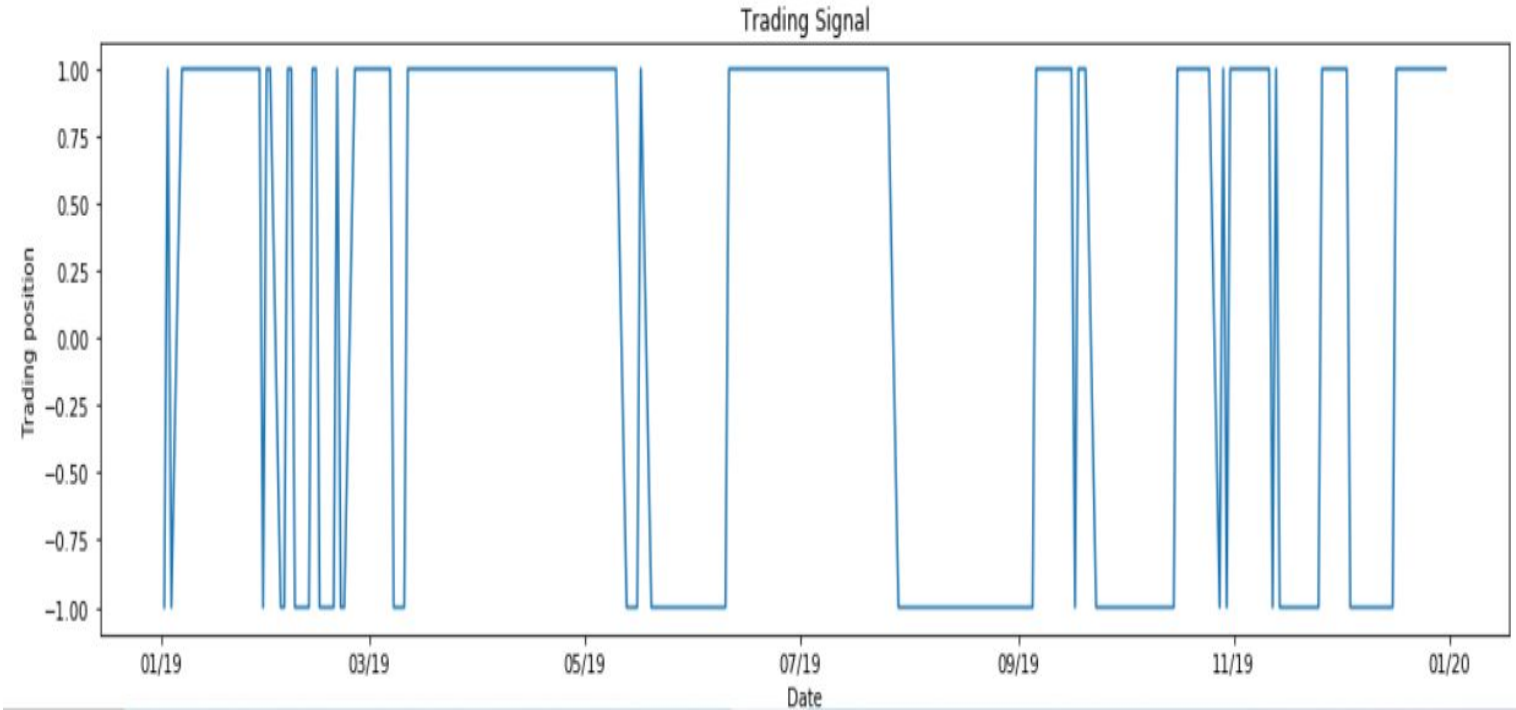
then **Signal=1**

- **Sell**

if Actual[‘Close’] < Predicted[‘Close’]

then **Signal=-1**





Monthly Analysis

OBSERVATIONS

- Balance =100000
- Shares owned=30

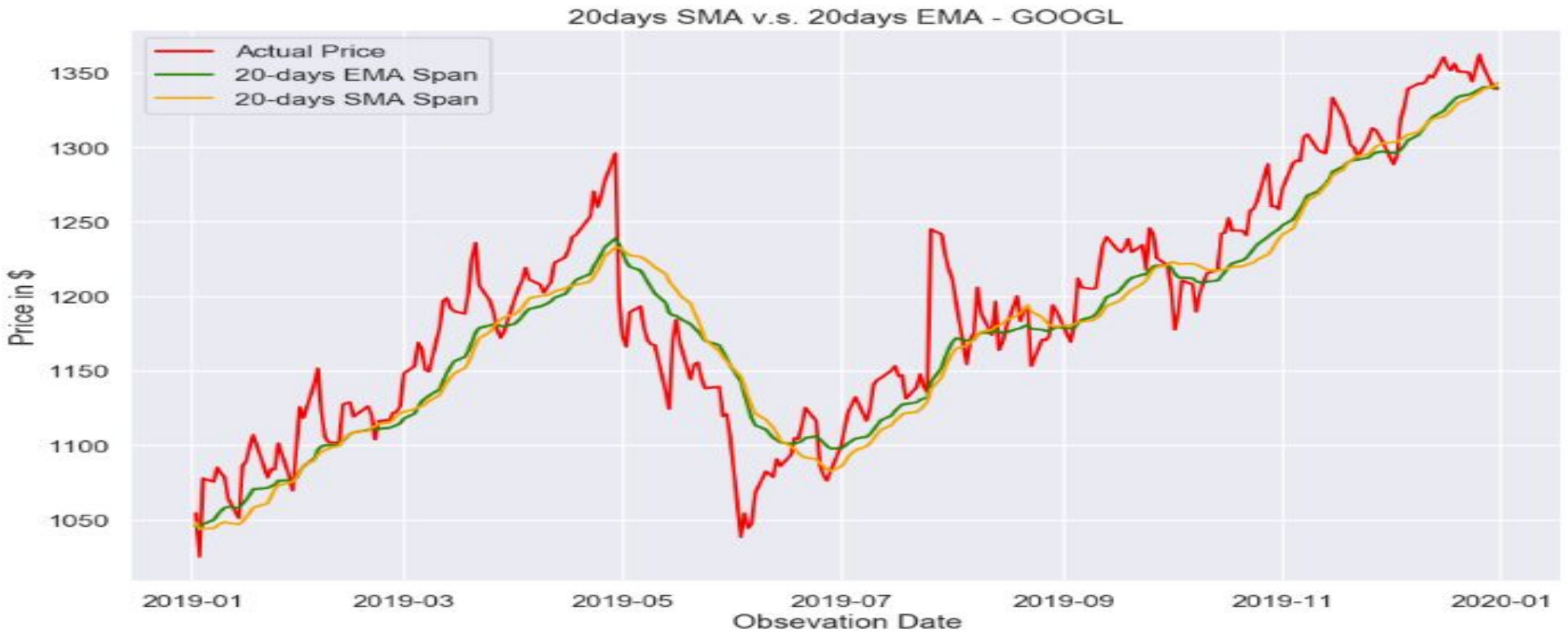
By following Buy and Sell Strategy :

- Shares owned=171
- Balance left = 40513
- Total value=388383.5

```
Account balance  100000
Initial number of shares owned  30
----------------Signals Given During Day To Day Trade---------
            Actual Close  20days EMA Close  Signal Action
Date
2019-12-03   1769.959961       1773.585217     1.0    Buy
2019-12-04   1760.689941       1772.357096    -1.0   Sell
2019-12-05   1740.479980       1769.321180    -1.0   Sell
2019-12-06   1751.599976       1767.633446    -1.0   Sell
2019-12-09   1749.510010       1765.907405    -1.0   Sell
2019-12-10   1739.209961       1763.364791    -1.0   Sell
2019-12-11   1748.719971       1761.970046    -1.0   Sell
2019-12-12   1760.329956       1761.813847    -1.0   Sell
2019-12-13   1760.939941       1761.730618    -1.0   Sell
2019-12-16   1769.209961       1762.442936    -1.0   Sell
2019-12-17   1790.660034       1765.130279     1.0    Buy
2019-12-18   1784.030029       1766.930255     1.0    Buy
2019-12-19   1792.280029       1769.344519     1.0    Buy
2019-12-20   1786.500000       1770.978375     1.0    Buy
2019-12-23   1793.000000       1773.075672     1.0    Buy
2019-12-24   1789.209961       1774.612271     1.0    Buy
2019-12-26   1868.770020       1783.579676     1.0    Buy
2019-12-27   1869.800049       1791.791140     1.0    Buy
2019-12-30   1846.890015       1797.038652     1.0    Buy
2019-12-31   1847.839966       1801.876872     1.0    Buy
----------------Results-------------------------
Balance left in Account 40513.06887817383
Total Number of shares  171
Total value of the shares  388383.4988781738
```

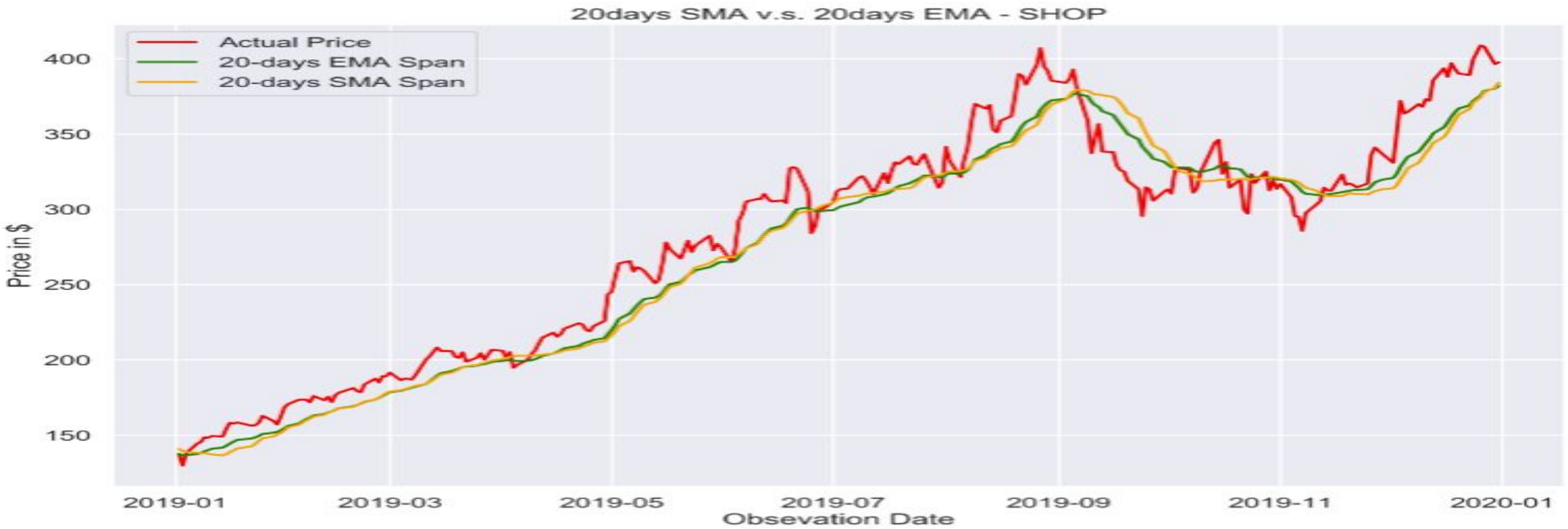
Competitor's Moving Averages

As competitors we have worked on GOOGLE and SHOPIFY Stock data




```
from sklearn.metrics import mean_squared_error
from math import sqrt
rmse_googl = sqrt(mean_squared_error(googl_new['Actual Close'], googl_new['20days EMA Close']))
print('rmse_googl','=', rmse_googl)
```

```
rmse_googl = 24.312460418069303
```




```
from sklearn.metrics import mean_squared_error
from math import sqrt
rmse_shop = sqrt(mean_squared_error(shop_new['Actual Close'], shop_new['20days EMA Close']))
print('rmse_shop','=', rmse_shop)
```

```
rmse_shop = 9.873831862414923
```


```
# Conclusions
from prettytable import PrettyTable
import random
x = PrettyTable()
x.field_names = ["Stock","RMSE","RSquared"]
x.add_row(["AMZN", 39, 0.995])
x.add_row(["GOOGL",24.31, 0.9906 ])
x.add_row(["SHOP",9.87 ,0.9917 ])

print(x)
```

```
+-------+-------+----------+
| Stock |  RMSE | RSquared |
+-------+-------+----------+
|  AMZN |   39  |  0.995   |
| GOOGL | 24.31 |  0.9906  |
|  SHOP |  9.87 |  0.9917  |
+-------+-------+----------+
```

CONCLUSION & RESULTS

CODE : https://github.com/akashmdubey/AmazonStockAnalysis

Models	RMSE	R-Square	Sharpe Ratio	Profit%	Strategy
Kalman Filter	28.36	0.512	0.49	38.99%	Long and Short
GARCH	3.09	0.90	1.26	208.5 %	Buy and Hold, Sell
MIDAS	21.07	0.70	0.46	11.46%	Buy and Hold, Sell
Ridge Regression	6.49	0.96	0.36791703	50.87 %	Buy and Hold
Random Forest	64.47	0.72	0.6	18%	Buy and Hold
Moving Average	39.55	0.995	2.576	146.9%	Buy And Sell
LSTM	46.76	0.83	0.085	-	-

THANK YOU PROFESSOR and TA

THANK YOU GUYS FOR BEING SUCH WONDERFUL AUDIENCE

IT WAS A GREAT SEMESTER OF LEARNING UNDER YOUR GUIDANCE :)

WE LOVED YOUR JOKES AND STORIES WHICH KEPT US ENTERTAINED :)