Capstone Project on Investment and Trading: **Dipak Majhi**
Build a Stock Price Indicator
Machine Learning Engineer Nanodegree **17th June, 2017**

Definition

Project Overview:

Investment firms, hedge funds, and automated trading systems have used programming and advanced modeling to interact with and profit from the stock market since computerization of the exchanges in the 1970s. Whether by means of better analysis, signal identification, or automating the frequency of trades, the goal has been to leverage technology in order create investment systems that outperform alternatives.

I wanted to use my Machine Learning knowledge to solve a real problem. Since, I have always been interested in investment and trading, I chosen to work under the domain "Investment and Trading" for my Capstone project and Build a Stock Price Indicator to see if it could help in predicting stock prices in the short-term future.

I performed various financial analysis on the stocks data I downloaded from Yahoo Finance. I built few predictive models for predicting future prices one day, seven days, 14 days ahead in future etc. from the last date provided for training:

- LINEAR REGRESSION based model
- KNN (K nearest Neighbors based model)
- ARIMA (Auto regressive integrated moving Average) based model

Datasets:

I have downloaded the following Datasets from Yahoo Finance: for e.g. – Apple, Google, Amazon, Facebook, General Electric, Microsoft, Gold Shares (GOLD), S&P 500. S&P 500 is the American market index, I used it to see trending behavior of other stocks by the overall market changes tracked by S&P 500.

Capstone Project on Investment and Trading: **Dipak Majhi**
Build a Stock Price Indicator
Machine Learning Engineer Nanodegree **17th June, 2017**

Problem Statement:

Build machine learning models that learn from historical stock price attributes and predict the stock price on a future date (any day after the last training date), I am only predicting the Adjusted Closing pricing. Since the target variable is a continuous value so this is a regression task.

Strategy to solve the problem: Building few models such as Linear Regression, KNN Regression which learn from historical stock data, attributes such as: Open, High, Low, Close, Volume, Adjusted Closing and "Adjusted Closing price" n days ahead in future. After the model is built on the historical data, it would predict the Adjusted Closing price for a date (only one date) which is "n" days ahead in future from the last date of training. Please note splitting the datasets has to be done in the ascending order of dates manually, and not using the SKLEARN test train split module which randomly split the data. In the stock price prediction world, we must not train a model on future datasets and try to predict the stock prices for past dates

Non-seasonal ARIMA model: Non-seasonal ARIMA model is based on three terms:

- AR –Autoregressive – (p), A model that uses the dependent relationship between an observation and some number of lagged observations.

- I – Integrated – (d), The use of differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary.

- MA – Moving Average – (q). A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

Parameters p, d, and q are non-negative integers, p is the order (number of time lags) of the autoregressive model, d is the degree of differencing (the number of times the data have had past values subtracted), and q is the order of the moving-average model.

Three items should be considered to determine a first guess at an ARIMA model: a time series plot of the data, the ACF, and the PACF.

AR model: Identification of an AR model is often best done with the PACF. the theoretical PACF "shuts off" past the order of the model. The phrase "shuts off" means that in theory the partial autocorrelations are equal to 0 beyond that point

MA model: For an MA model, the theoretical PACF does not shut off, but instead tapers toward 0 in some manner. A clearer pattern for an MA model is in the ACF. The ACF will have non-zero autocorrelations only at lags involved in the model.

Udacity Free Course: Time Series Forecasting:

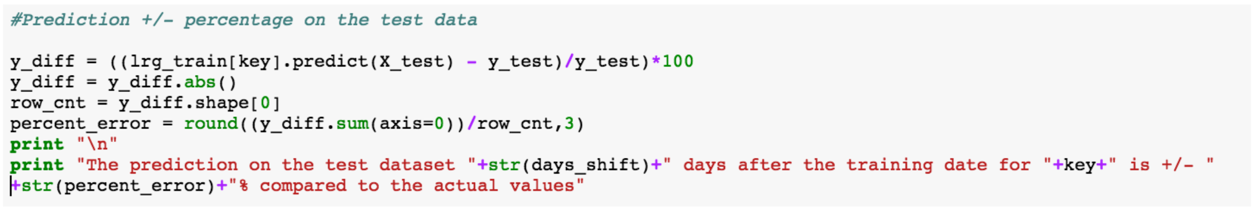
- https://www.udacity.com/course/time-series-forecasting--ud980
- https://onlinecourses.science.psu.edu/stat510/node/62
- https://onlinecourses.science.psu.edu/stat510/node/49
- https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average
- http://www.inertia7.com/projects/time-series-stock-market-python

Metrics:

R2 score on the train and test datasets are calculated to measure the performance of the model. Best possible score is 1. R2 Score is used to test the performance of all three models – Linear Regression, KNN Regression and ARIMA.

For e.g.: Score on training data for GOOGLE: 0.95
Score on test data for GOOGLE: 0.82

In addition to the R2 score, I am calculating the variation of the predicted stock price compared to the actual prices in the test dataset. On an average, I want to have an idea how off my predictions are compared to actual prices. Please note that the way I am calculating might not be the best way to do it, but it definitely gives me a mathematical inference to understand how off my predicted values are compared to the actual values. For e.g The prediction on the test dataset 6 days after the training date for GOOG is +/- 2.721% compared to the actual values. Please have a look at the code below to see how I have implemented it, please note that I am calculating absolute average difference in percentage between the predicted values and the actual values. Then I am just print out as +/- X% compared to the actual values:



```
#Prediction +/- percentage on the test data

y_diff = ((lrg_train[key].predict(X_test) - y_test)/y_test)*100
y_diff = y_diff.abs()
row_cnt = y_diff.shape[0]
percent_error = round((y_diff.sum(axis=0))/row_cnt,3)
print "\n"
print "The prediction on the test dataset "+str(days_shift)+" days after the training date for "+key+" is +/- "
+str(percent_error)+"% compared to the actual values"
```

Explaining R2 Score:

In statistics, the coefficient of determination, denoted R2 or r2 and pronounced "R squared", is a number that indicates the proportion of the variance in the dependent variable that is predictable from the independent variable. The coefficient R^2 is defined as (1 - u/v), where u is the regression sum of squares ((y_true - y_pred) ** 2).sum() and v is the residual sum of squares ((y_true - y_true.mean()) ** 2).sum(). Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0.

Analysis

Data Exploration:

I have downloaded daily historical stocks data from Yahoo Finance between 12-05-2011 and 12-02-2016 for the following stocks along with SNP_500(American Market index, I want to see how these stock trend with respect to market trends explained by SNP_500). ‘FACEBOOK’ went public around mid of 2012, so we don’t have data for ‘FACEBOOK’ prior to that.

The following stocks are considered: [link]

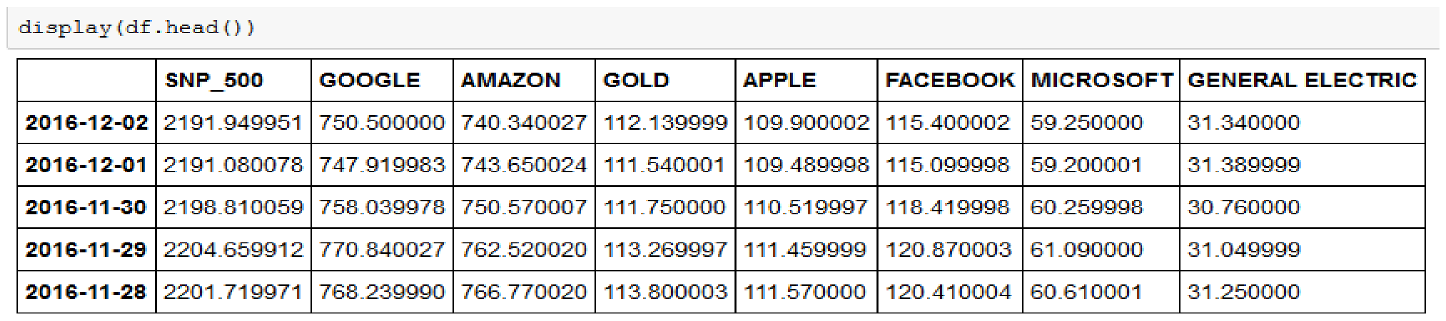
'GOOGLE', 'AMAZON', 'GOLD', 'APPLE', 'FACEBOOK', 'MICROSOFT', 'GENERAL ELECTRIC'

Each of this stock has the following attributes:

Date, Open, High, Low, Close, Volume and Adj Close

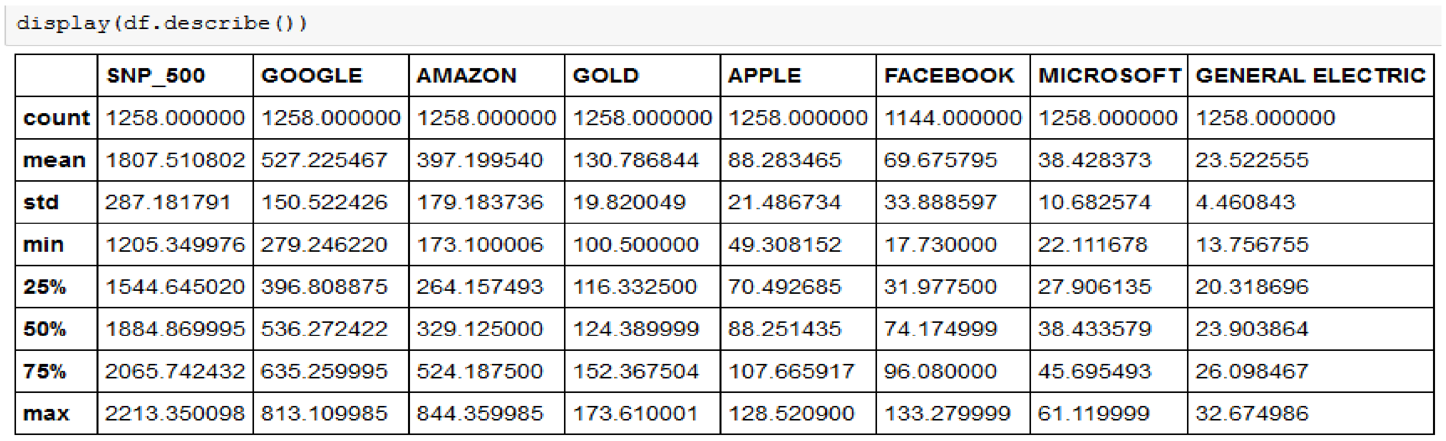
There are a total of 1258 data points for all stocks except for ‘FACEBOOK’ where we have a total of 1144 data points

Sample Data of all stocks as well as the statistical description of the dataset:



```
display(df.head())
```

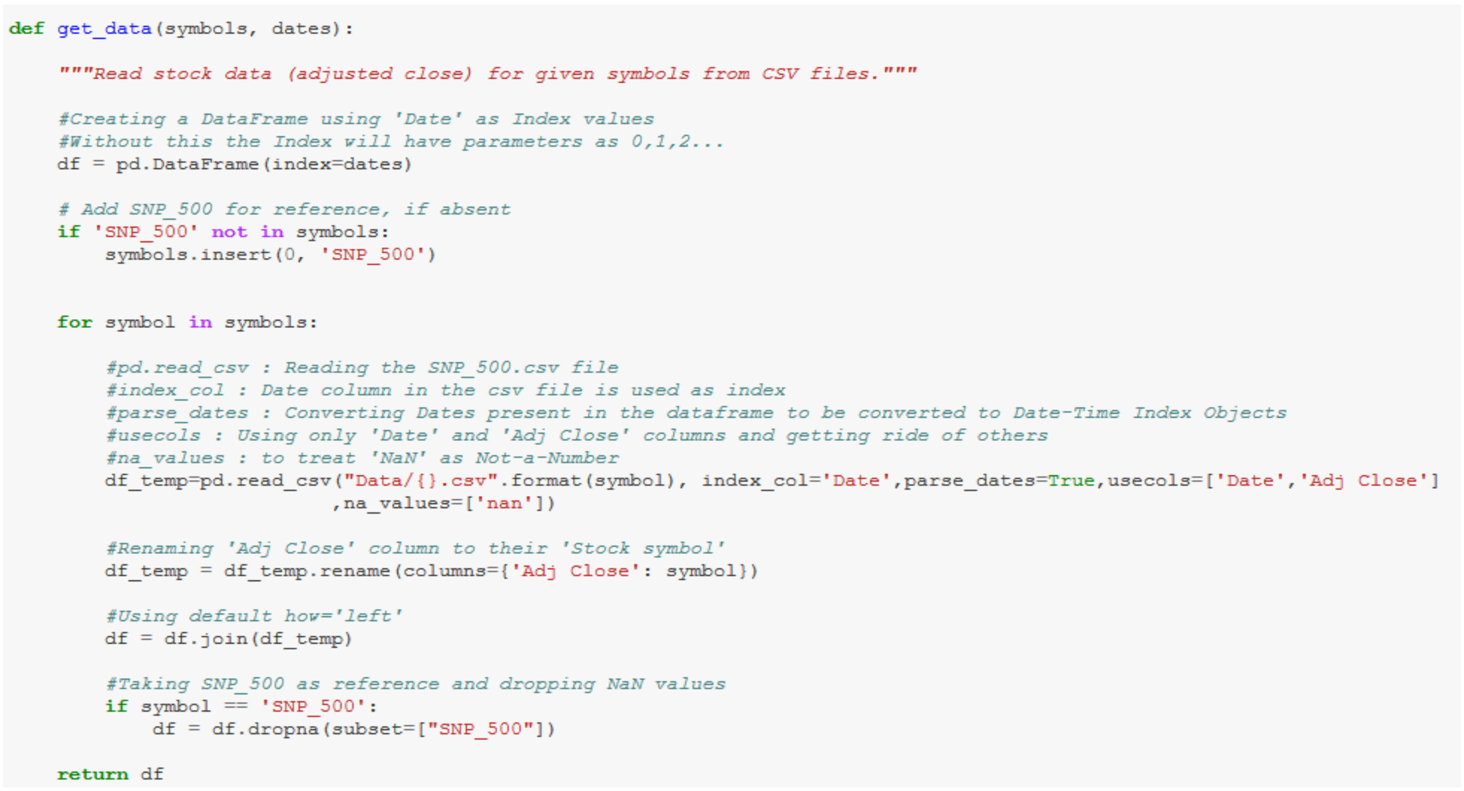
	SNP_500	GOOGLE	AMAZON	GOLD	APPLE	FACEBOOK	MICROSOFT	GENERAL ELECTRIC
2016-12-02	2191.949951	750.500000	740.340027	112.139999	109.900002	115.400002	59.250000	31.340000
2016-12-01	2191.080078	747.919983	743.650024	111.540001	109.489998	115.099998	59.200001	31.389999
2016-11-30	2198.810059	758.039978	750.570007	111.750000	110.519997	118.419998	60.259998	30.760000
2016-11-29	2204.659912	770.840027	762.520020	113.269997	111.459999	120.870003	61.090000	31.049999
2016-11-28	2201.719971	768.239990	766.770020	113.800003	111.570000	120.410004	60.610001	31.250000



```
display(df.describe())
```

	SNP_500	GOOGLE	AMAZON	GOLD	APPLE	FACEBOOK	MICROSOFT	GENERAL ELECTRIC
count	1258.000000	1258.000000	1258.000000	1258.000000	1258.000000	1144.000000	1258.000000	1258.000000
mean	1807.510802	527.225467	397.199540	130.786844	88.283465	69.675795	38.428373	23.522555
std	287.181791	150.522426	179.183736	19.820049	21.486734	33.888597	10.682574	4.460843
min	1205.349976	279.246220	173.100006	100.500000	49.308152	17.730000	22.111678	13.756755
25%	1544.645020	396.808875	264.157493	116.332500	70.492685	31.977500	27.906135	20.318696
50%	1884.869995	536.272422	329.125000	124.389999	88.251435	74.174999	38.433579	23.903864
75%	2065.742432	635.259995	524.187500	152.367504	107.665917	96.080000	45.695493	26.098467
max	2213.350098	813.109985	844.359985	173.610001	128.520900	133.279999	61.119999	32.674986

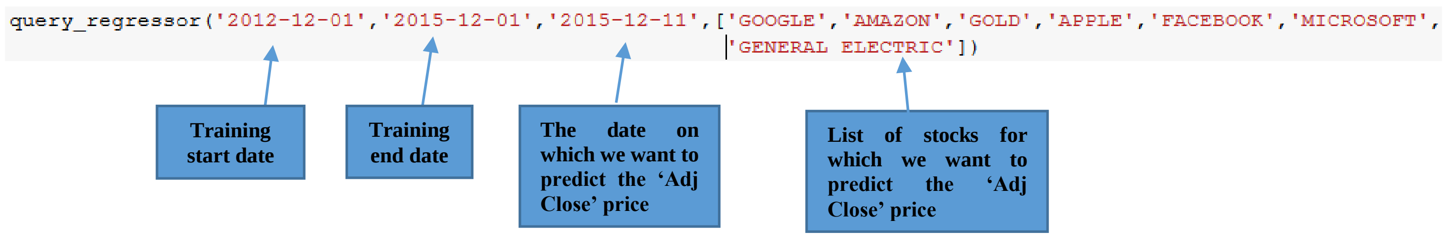
All these Data sets are loaded in panda Data Frame with Date as the index, sample below with all rows with no values for 'SNP 500' removed. Meaning, we want to keep SNP 500 as the standard and base of our analysis, if we missing data for 'SNP 500' for certain dates, we want to remove all data for all stocks. Please refer the code snippet:



```
def get_data(symbols, dates):

    """Read stock data (adjusted close) for given symbols from CSV files."""

    #Creating a DataFrame using 'Date' as Index values
    #Without this the Index will have parameters as 0,1,2...
    df = pd.DataFrame(index=dates)

    # Add SNP_500 for reference, if absent
    if 'SNP_500' not in symbols:
        symbols.insert(0, 'SNP_500')


    for symbol in symbols:

        #pd.read_csv : Reading the SNP_500.csv file
        #index_col : Date column in the csv file is used as index
        #parse_dates : Converting Dates present in the dataframe to be converted to Date-Time Index Objects
        #usecols : Using only 'Date' and 'Adj Close' columns and getting ride of others
        #na_values : to treat 'NaN' as Not-a-Number
        df_temp=pd.read_csv("Data/{}.csv".format(symbol), index_col='Date',parse_dates=True,usecols=['Date','Adj Close']
                           ,na_values=['nan'])

        #Renaming 'Adj Close' column to their 'Stock symbol'
        df_temp = df_temp.rename(columns={'Adj Close': symbol})

        #Using default how='left'
        df = df.join(df_temp)

        #Taking SNP_500 as reference and dropping NaN values
        if symbol == 'SNP_500':
            df = df.dropna(subset=["SNP_500"])

    return df
```

An addition to the above attributes, I will be appending one more attribute – 'Adj_Close_req_Days_Later'. The days to be shifted to get the future 'Adj Close' price will be based on the user input – the number of days between training_end date and the future date for which I need to predict 'Adj Close' price.

This is how I will call the “query_regressor” function:



```
query_regressor('2012-12-01','2015-12-01','2015-12-11',['GOOGLE','AMAZON','GOLD','APPLE','FACEBOOK','MICROSOFT',
                                                         'GENERAL ELECTRIC'])
```

Training start date
Training end date
The date on which we want to predict the ‘Adj Close’ price
List of stocks for which we want to predict the ‘Adj Close’ price

Code snippet below to get the number of days between the last training Date and the Prediction date:

```
def train_regressor(start_date, end_date, stock_list, days_shift):

    #prepare the date range for the DataFrame
    #I am keeping the the maximum date available on my collected data from Yahoo Finance

    dates = pd.date_range(start_date, end_date) #'2016-12-01')

    #Setting the variables to pick the indices for the training and test datasets

    print "days between the two training dates"
    date_format = "%Y-%m-%d"
    a = datetime.strptime(start_date, date_format)
    b = datetime.strptime(end_date, date_format)
    delta = b - a
    num_days = delta.days
```

If there are non-trading days, for e.g. Saturday, Sunday or holidays, we want to forward fill first and then backward filling, code snippet below:

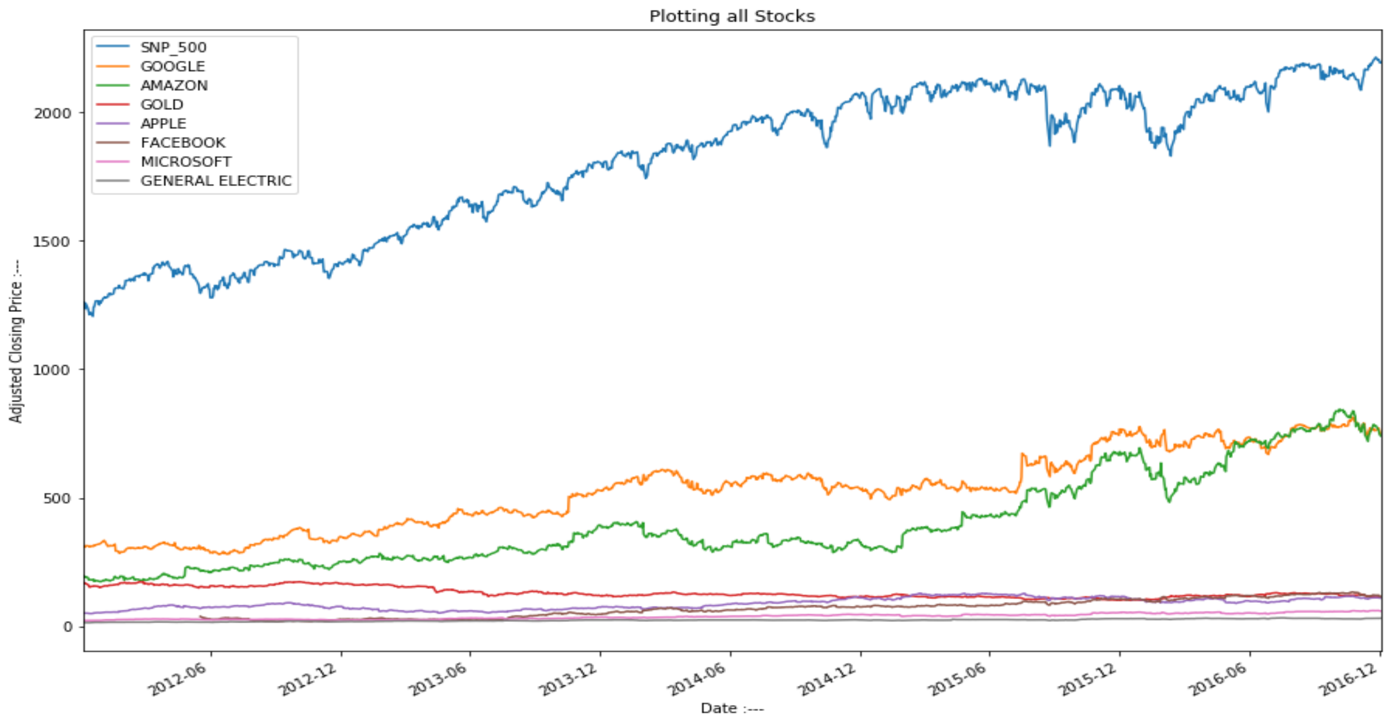
```
 # Fill empty trade dates with forward filling dates first and then backward filling
df.fillna(method="ffill", inplace="True")
df.fillna(method="bfill", inplace="True")
```

Exploratory Visualization:

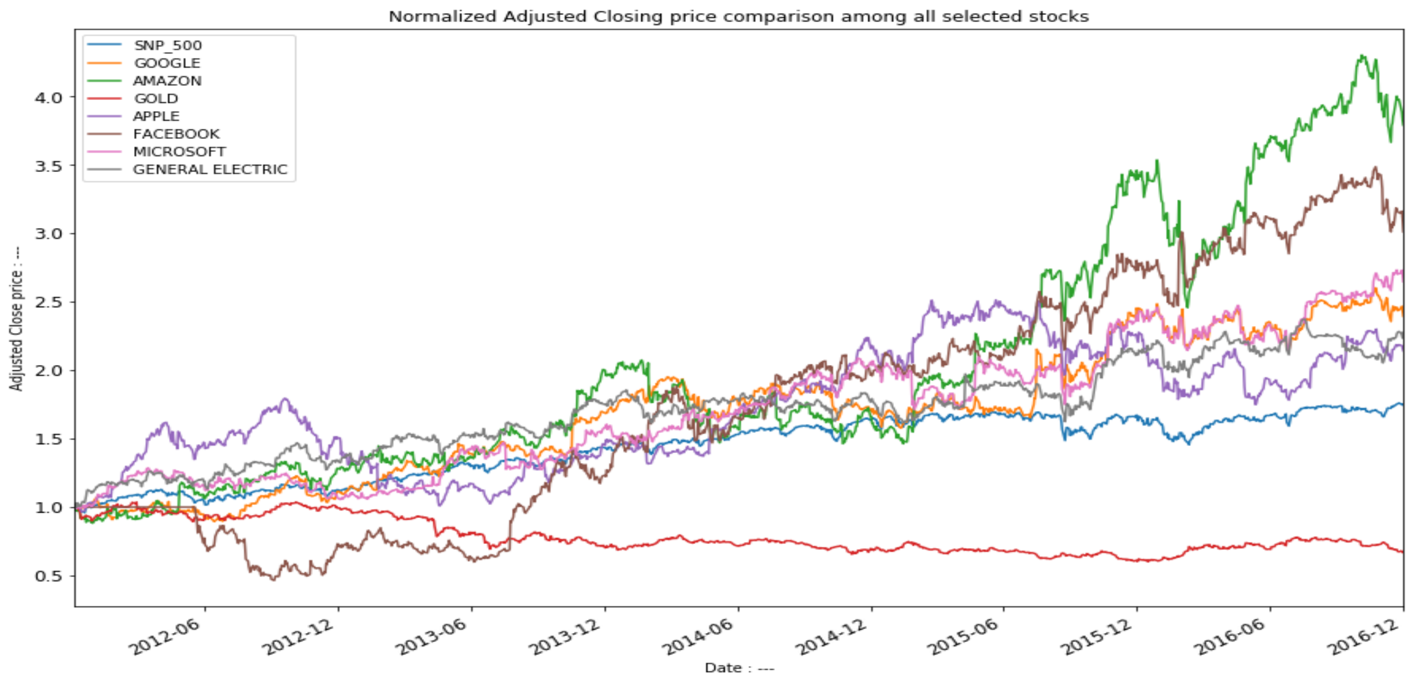
Please refer the IPython notebook “Analysis.ipynb” for all exploratory visualization code:

“Plotting Adjacency Close price for all stocks in comparison to the SNP 500 index”:

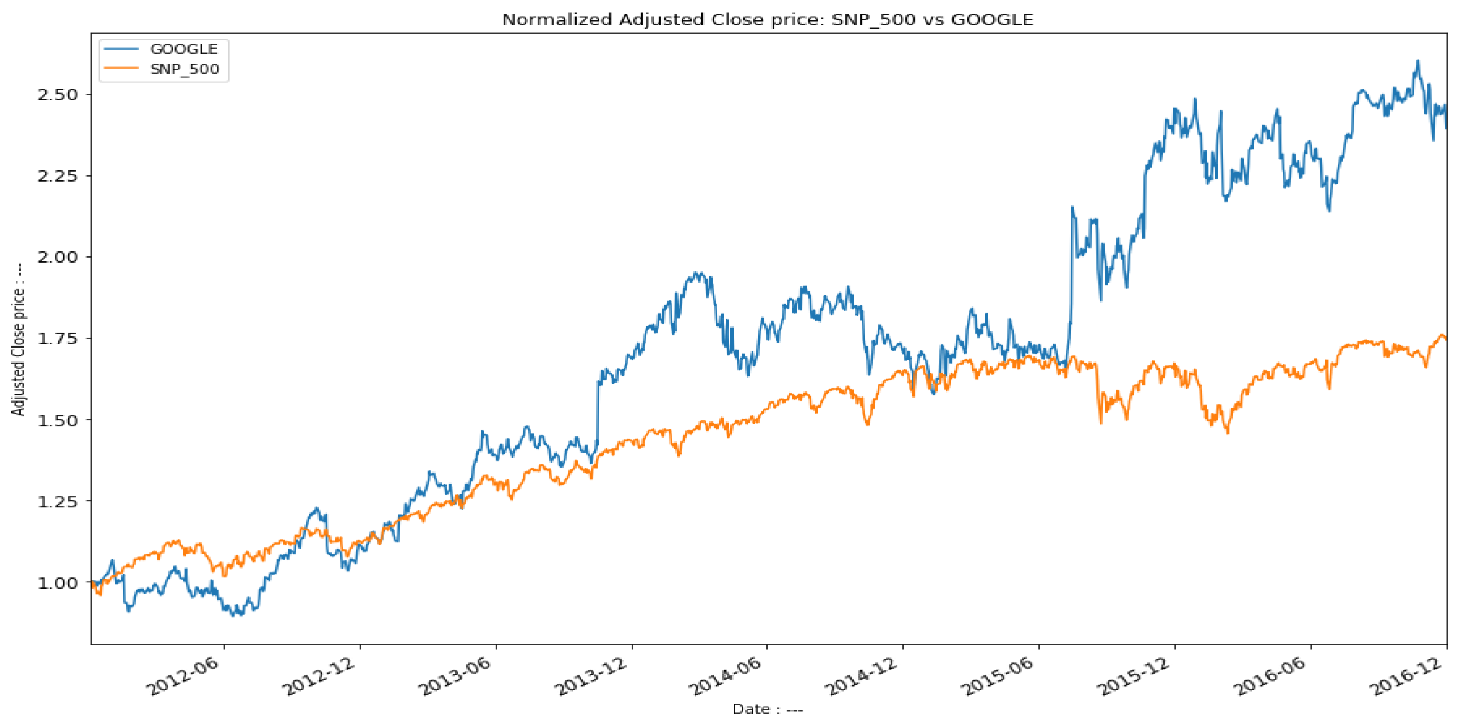
Please find a sample plot of “Adj Close” price for all stocks between the date '2011-01-05' and '2016-12-02':



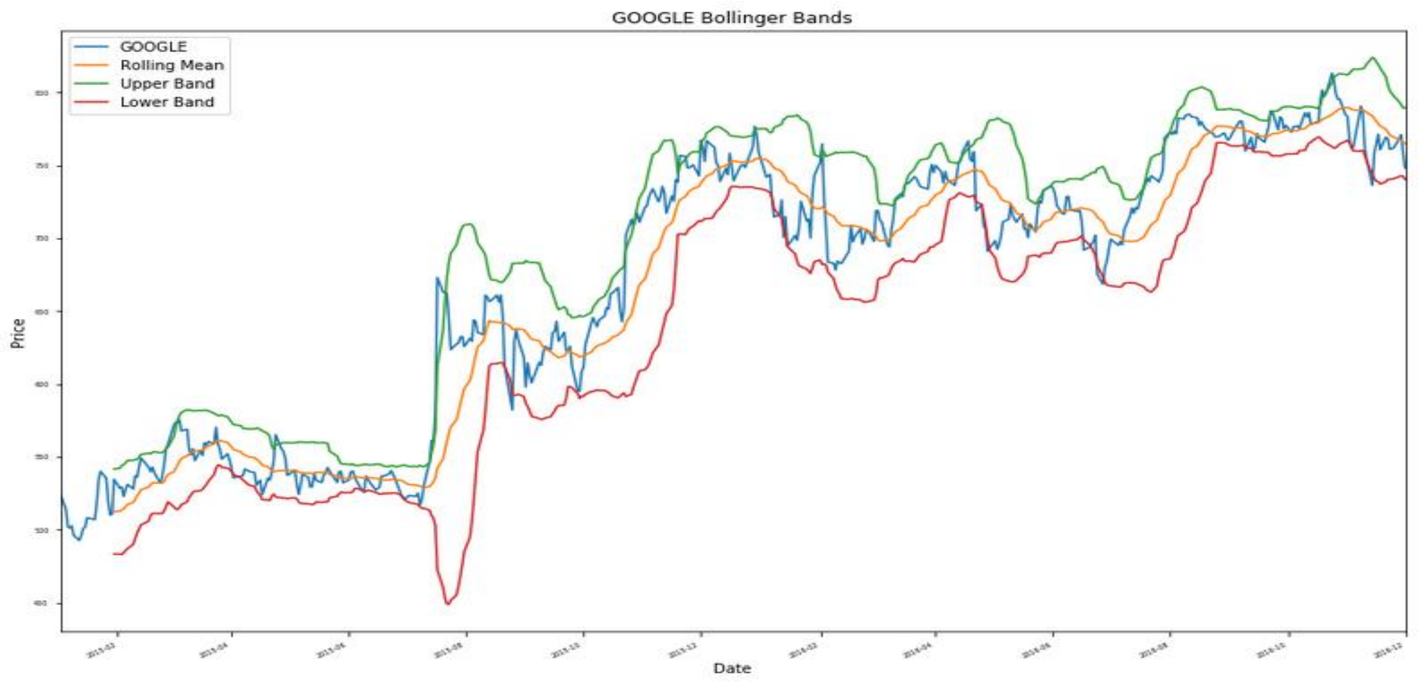
Please find below the plot- Comparison of Normalized stock price among all stocks between 2011 and 2017:



Please find below the plot: Normalized Adj Price comparison - SNP_500 vs GOOGLE:

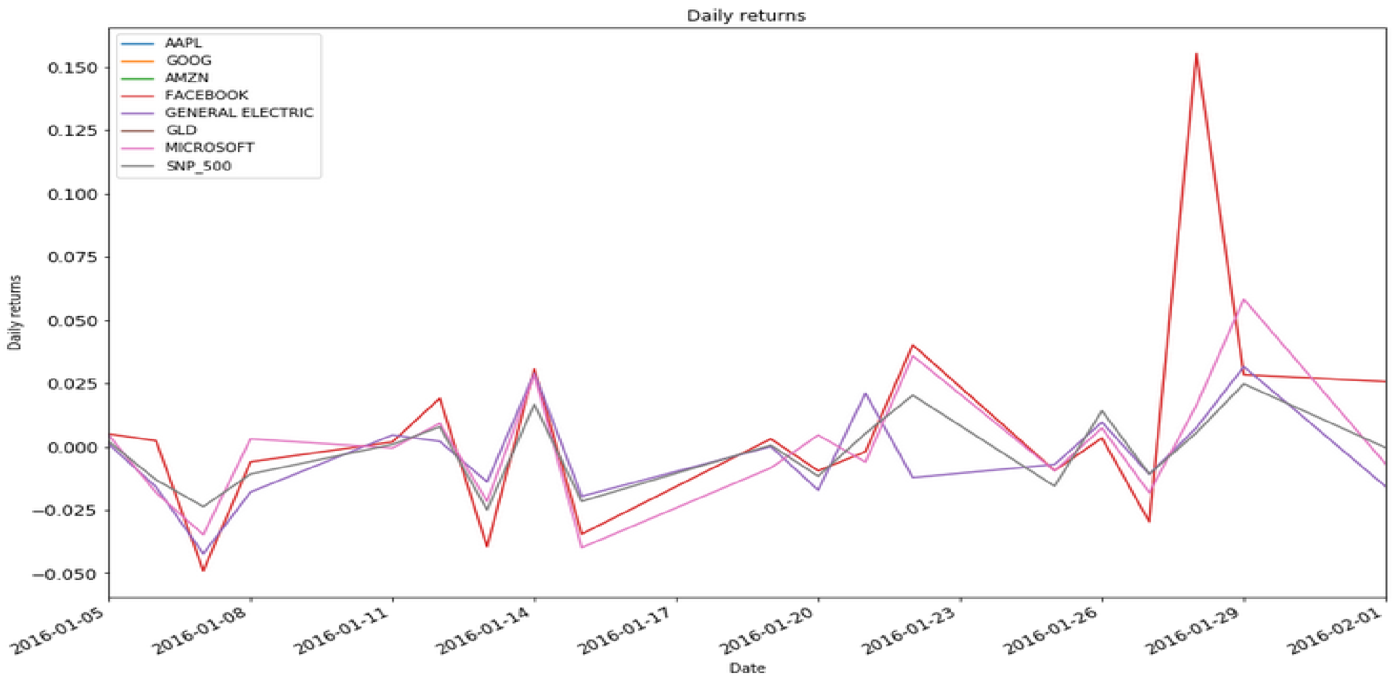


Bollinger Bands for GOOGLE: The popular notion is that when the stock price meets the lower Bollinger band (mean – 2*Standard deviation), then it's usually a buy signal & when the stock price meets the upper Bollinger band (mean + 2*Standard deviation) its usually a sell signal, Please find below the plot:

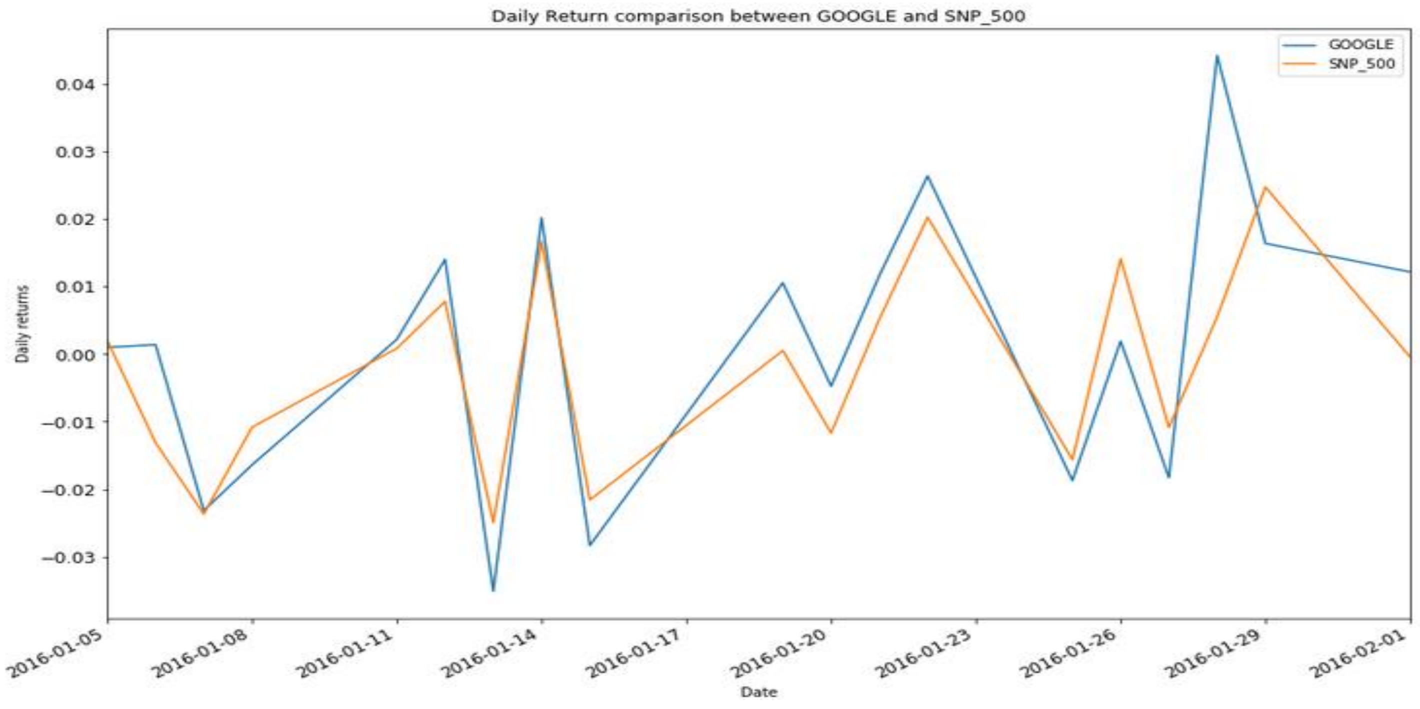


Daily returns:

- Following is the Daily returns plot of all stocks for the month of Jan 2016:

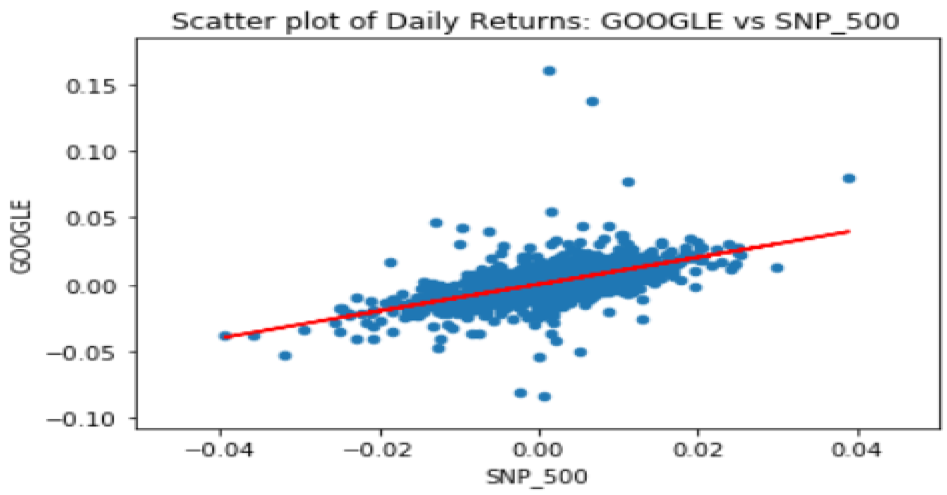


- Following is the daily returns plot between GOOGLE and SNP_500 for the month of Jan 2016, its seems GOOGLE frequently moves in the same direction as SNP_500 mostly, but in few instances, it also moves farther from SNP_500:

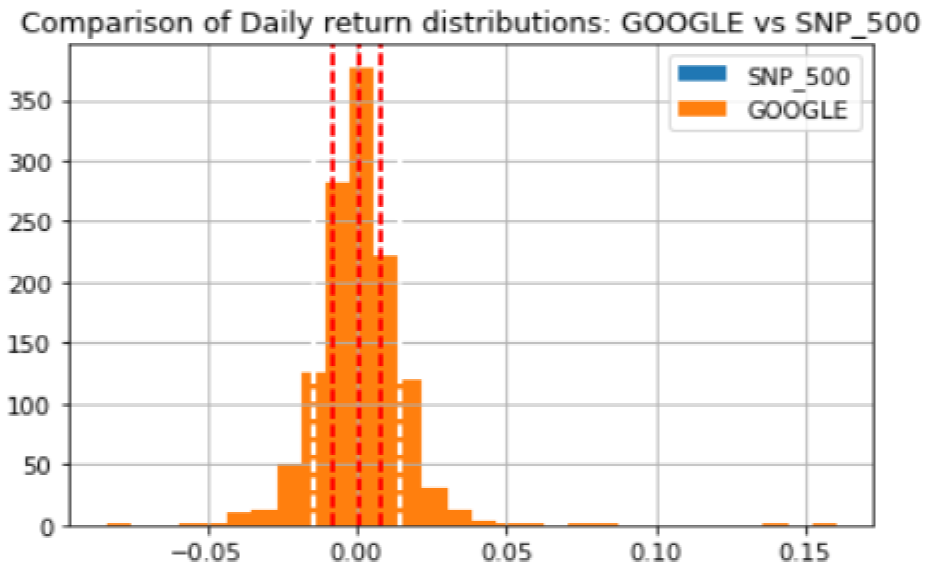


- Following is the Scatter plot between GOOGLE and SNP_500 (between dates '2011-01-01', '2016-12-01') and fitting a line & looking at the statistics below. As we can see the slope or Beta is almost 1, this infers GOOGLE is very reactive to the market meaning if the market goes up by 1%, GOOGLE also goes up by 1 percent. Alpha is positive, which means GOOGLE performs little bit better than SNP 500 everyday on an average:

```
Beta GOOG = 1.00683512076
Alpha GOOG = 0.00032240857603
```



- Following is the plot for Daily return distribution of GOOGLE vs SNP_500 represented by histogram as well the comparative analysis: The 'white' colored dotted line identifies GOOGLE's mean and standard deviation, while the 'red' colored identifies SNP_500's mean and std. as we can see the distribution for both of them is very similar, other than GOOGLE has a higher Standard Deviation and higher Kurtosis which infers GOOGLE's distribution is not exactly Gaussian with flat tails.



Mean, Standard Deviation and Kurtosis for SNP_500: -----
Mean = 0.000476
Standard Deviation= 0.008204
Kurtosis = 1.891211

Mean, Standard Deviation and Kurtosis for GOOGLE: -----
Mean = 0.000802
Standard Deviation = 0.014717
Kurtosis = 19.52001

- Following is the Pearson co-relation among Daily returns of the stocks, please refer below. We can see that GOOGLE is the only stock which has the best co-relation with SNP_500 as we would infer from explanation above:

	SNP_500	**GOOGLE**	**AMAZON**	**GOLD**	**APPLE**	**FACEBOOK**	**MICROSOFT**	**GENERAL ELECTRIC**
SNP_500	1.000000	0.561288	0.494226	-0.001239	0.499828	0.317453	0.613608	0.725937
GOOGLE	0.561288	1.000000	0.486442	-0.040297	0.317679	0.312197	0.428412	0.381830
AMAZON	0.494226	0.486442	1.000000	-0.023005	0.236503	0.296935	0.351674	0.321093
GOLD	-0.001239	-0.040297	-0.023005	1.000000	0.028924	-0.023161	-0.014787	0.006026
APPLE	0.499828	0.317679	0.236503	0.028924	1.000000	0.193689	0.324849	0.298751
FACEBOOK	0.317453	0.312197	0.296935	-0.023161	0.193689	1.000000	0.210004	0.208330
MICROSOFT	0.613608	0.428412	0.351674	-0.014787	0.324849	0.210004	1.000000	0.402501
GENERAL ELECTRIC	0.725937	0.381830	0.321093	0.006026	0.298751	0.208330	0.402501	1.000000

- Following are the Cumulative returns for all the stocks for the period between '2015-01-01' & '2016-12-01:

```
Commulative Return for SNP_500 is:
6.46%
Commulative Return for GOOGLE is:
42.51%
Commulative Return for AMAZON is:
141.04%
Commulative Return for GOLD is:
-2.23%
Commulative Return for APPLE is:
4.12%
Commulative Return for FACEBOOK is:
46.72%
Commulative Return for MICROSOFT is:
33.62%
Commulative Return for GENERAL ELECTRIC is:
32.64%
```

Portfolio Statistics between the dates '2014-12-01', '2016-12-01':

- Code snippet with equal allocation to a portfolio of Seven stocks:

```
# Define a date range
dates = pd.date_range('2014-12-01', '2016-12-01')

# Choose stock symbols to read
symbols = ['GOOGLE', 'AMAZON', 'GOLD', 'APPLE', 'FACEBOOK', 'MICROSOFT', 'GENERAL ELECTRIC']

# Get stock data
df = get_data(symbols, dates)

# Fill empty trade dates with forward filling dates first and then backward filling
df.fillna(method="ffill", inplace="True")
df.fillna(method="bfill", inplace="True")

df_port = df.drop('SNP_500', axis=1)

# Normalize stock prices
df_normalized = normalize_data(df_port)
print "\n"
print "Normalized: "
display(df_normalized.head())
print "\n"

# Portfolio allocation for each stock
allocation = [0.166, 0.166, 0.166, 0.166, 0.166, 0.166, 0.166]
df_allocation = df_normalized * allocation
print "Allocated: "
display(df_allocation.head())
print "\n"

# Starting value for each stock -- dividing 1 million equally among all stocks

starting_value = 1000000
df_values = df_allocation * starting_value
print "Portfolio Stock value : "
display(df_values.head())
print "\n"
```

- Sample: Value of each Stock in the portfolio:

```
Portfolio Stock value :
```

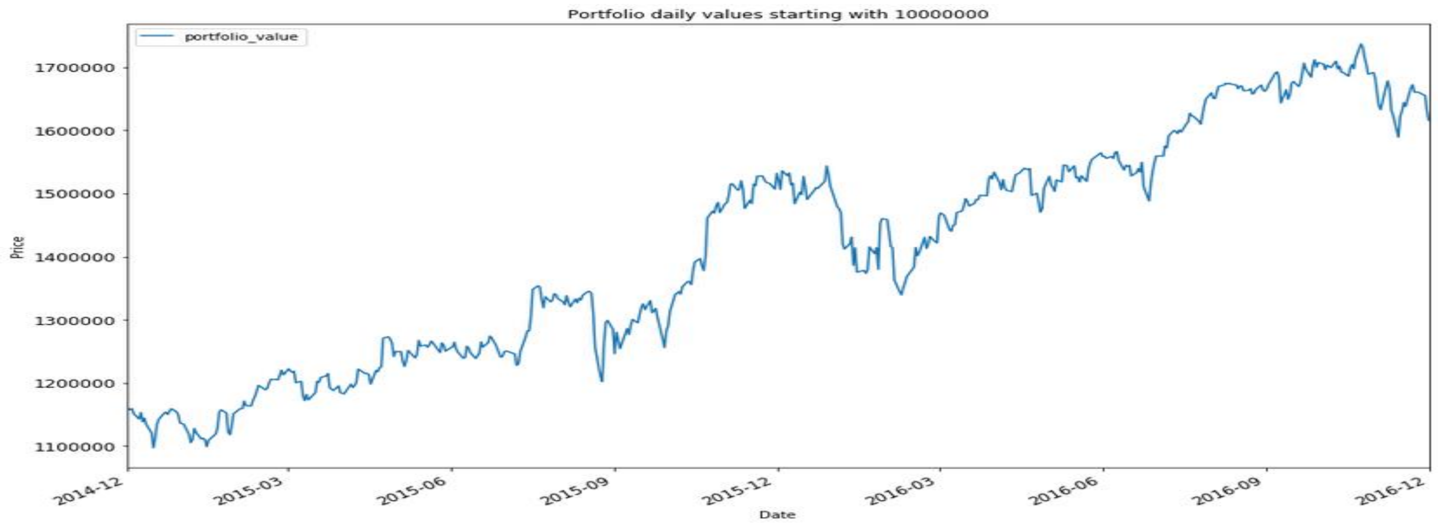
	GOOGLE	AMAZON	GOLD	APPLE	FACEBOOK	MICROSOFT	GENERAL ELECTRIC
2014-12-01	166000.000000	166000.000000	166000.000000	166000.000000	166000.000000	166000.000000	166000.000000
2014-12-02	165984.450553	166157.851742	163949.558296	165365.252335	166795.741246	165453.721704	166191.379370
2014-12-03	165228.776100	161162.576687	165644.021279	167240.637232	165513.712823	164156.321562	168296.682013
2014-12-04	167091.544313	161381.530178	165003.252462	166605.889567	166309.454069	166751.132657	166446.574596
2014-12-05	163344.253836	159191.965736	162938.580152	165899.018405	168785.093257	165317.148527	165936.197784

- Portfolio Value – summing up all the stock prices:

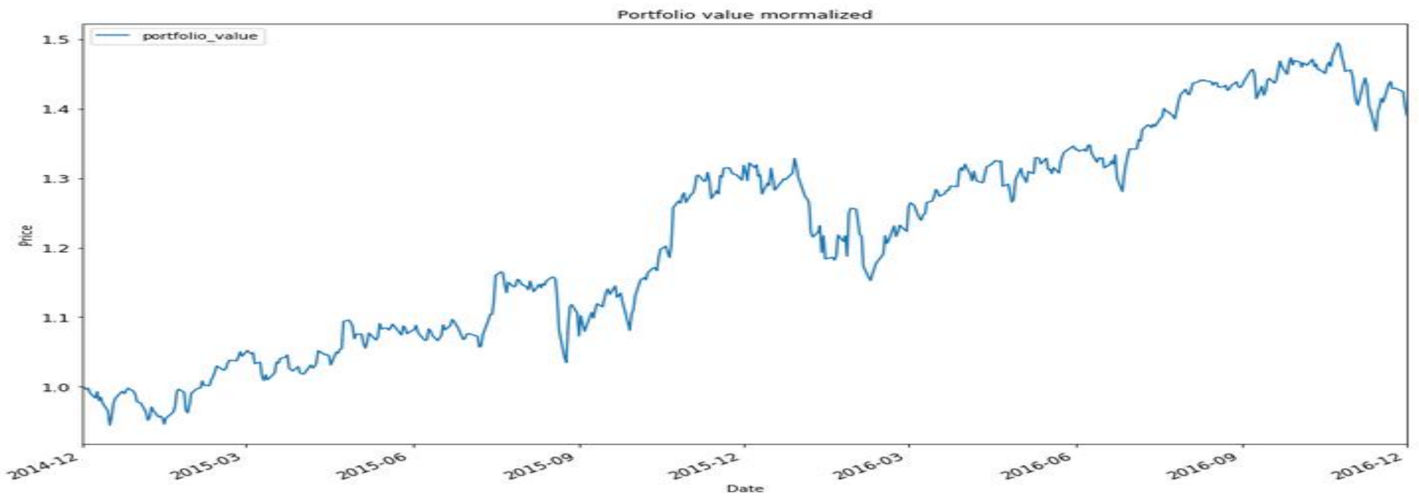
```
Portfolio values by day:
```

	portfolio_value
2014-12-01	1.162000e+06
2014-12-02	1.159898e+06
2014-12-03	1.157243e+06
2014-12-04	1.159589e+06
2014-12-05	1.151412e+06

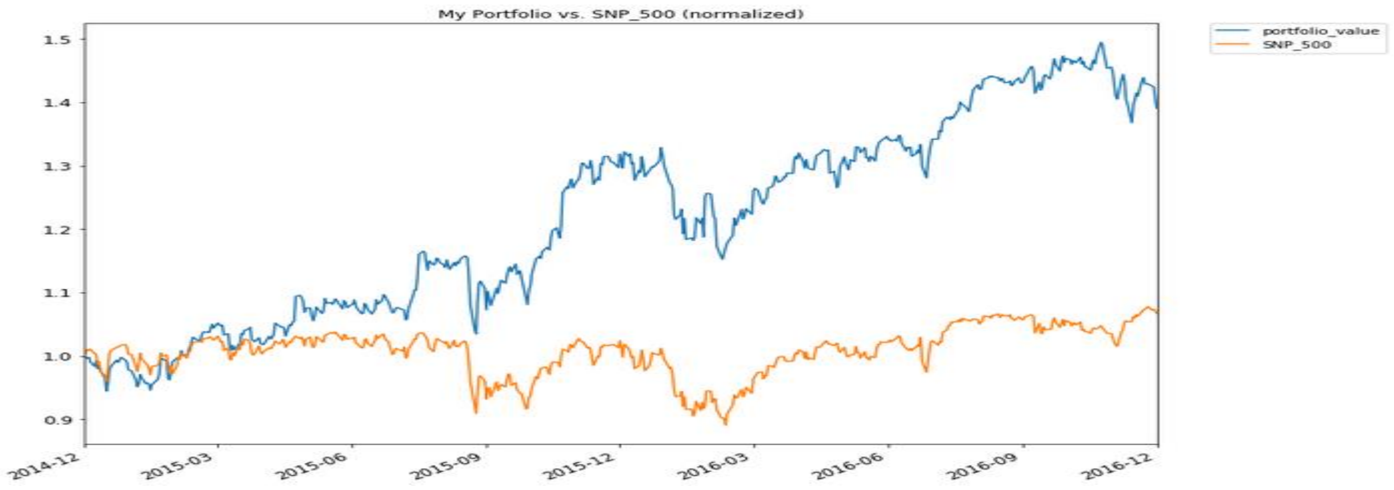
- Following is the plotted portfolio statistics:



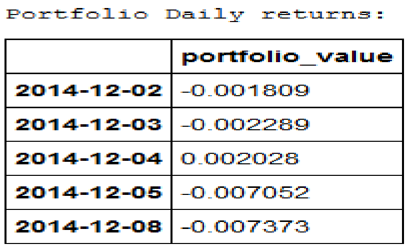
- Following is the plotted normalized portfolio statistics:



- Following is the portfolio value normalized plot vs normalized SNP_500



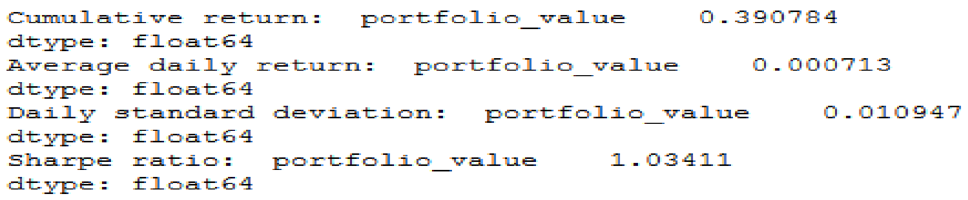
- Sample Data of Daily returns on the Portfolio



```
Portfolio Daily returns:
```

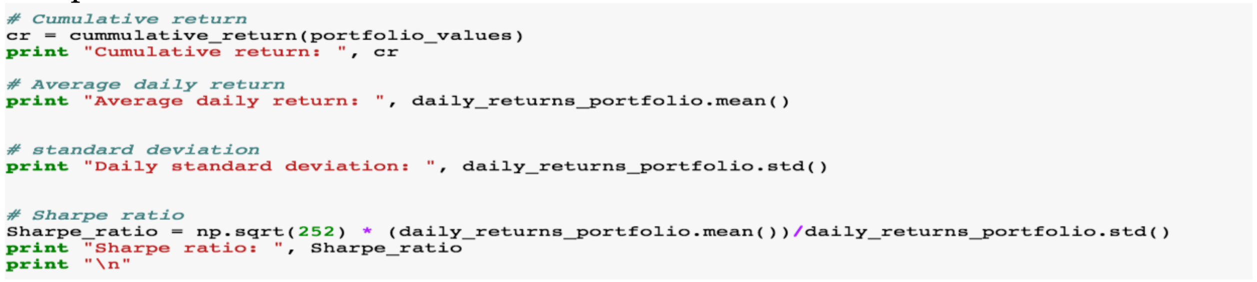
	portfolio_value
2014-12-02	-0.001809
2014-12-03	-0.002289
2014-12-04	0.002028
2014-12-05	-0.007052
2014-12-08	-0.007373

- Portfolio Cumulative Return, Average Daily Return, Standard Deviation, Sharpe Ratio



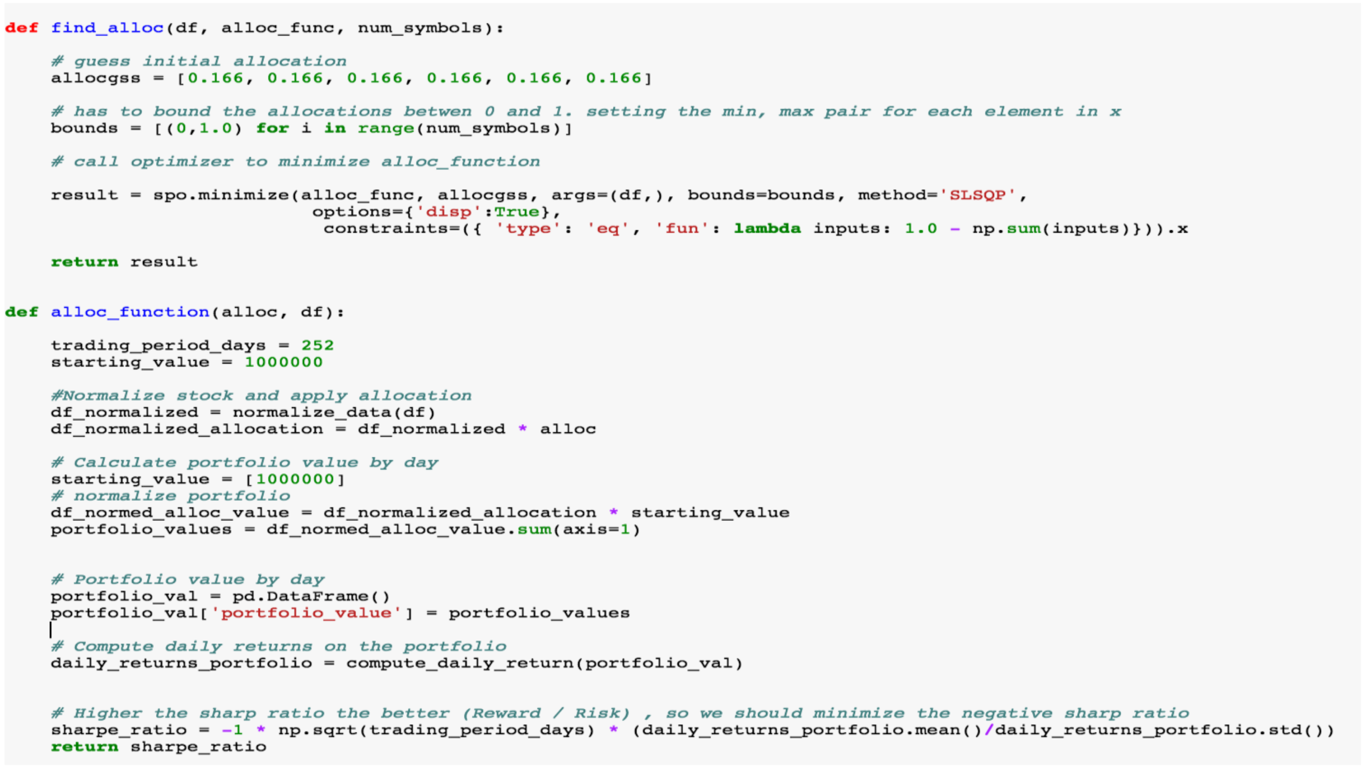
```
Cumulative return:  portfolio_value    0.390784
dtype: float64
Average daily return:  portfolio_value    0.000713
dtype: float64
Daily standard deviation:  portfolio_value    0.010947
dtype: float64
Sharpe ratio:  portfolio_value    1.03411
dtype: float64
```

- Code Snippet to calculate Cumulative Return, Average Daily Return, Standard Deviation, Sharpe Ratio for the Portfolio



```python
# Cumulative return
cr = cummulative_return(portfolio_values)
print "Cumulative return: ", cr

# Average daily return
print "Average daily return: ", daily_returns_portfolio.mean()


# standard deviation
print "Daily standard deviation: ", daily_returns_portfolio.std()


# Sharpe ratio
Sharpe_ratio = np.sqrt(252) * (daily_returns_portfolio.mean())/daily_returns_portfolio.std()
print "Sharpe ratio: ", Sharpe_ratio
print "\n"
```

Sharpe Ratio is a measure for calculating risk-adjusted return, and this ratio has become the industry standard for such calculations. Code Snippet below to optimize sharpe ratio:

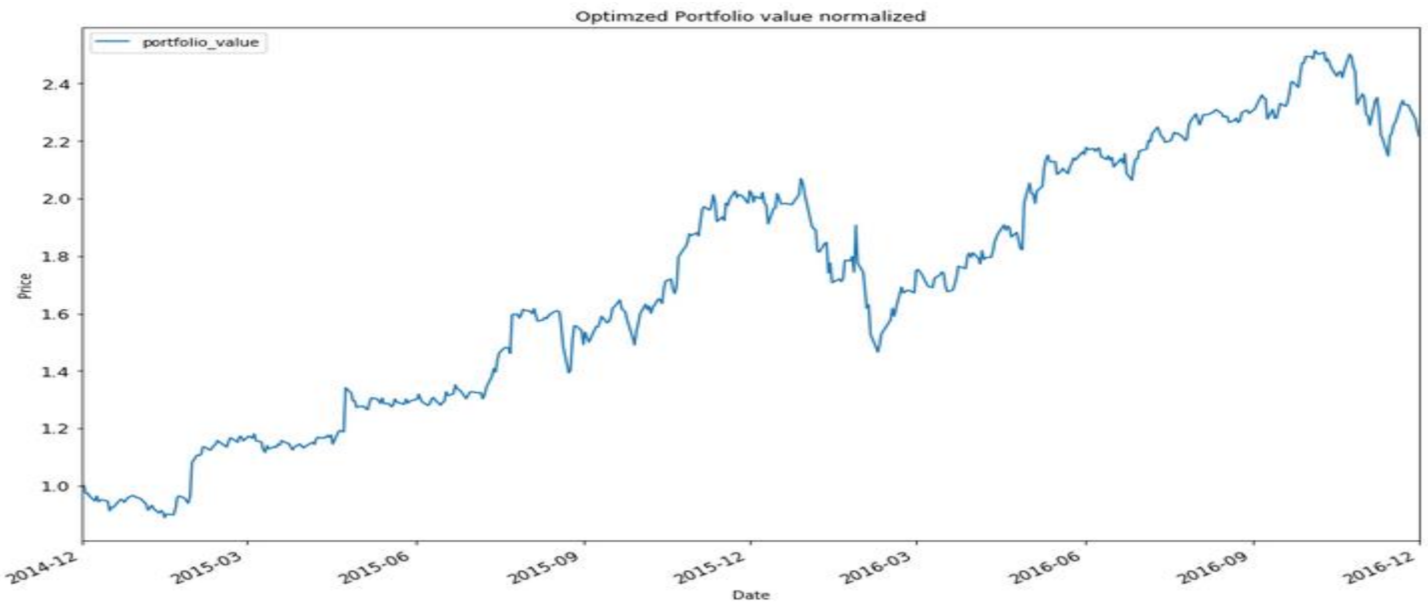


```python
def find_alloc(df, alloc_func, num_symbols):

    # guess initial allocation
    allocgss = [0.166, 0.166, 0.166, 0.166, 0.166, 0.166]

    # has to bound the allocations betwen 0 and 1. setting the min, max pair for each element in x
    bounds = [(0,1.0) for i in range(num_symbols)]

    # call optimizer to minimize alloc_function

    result = spo.minimize(alloc_func, allocgss, args=(df,), bounds=bounds, method='SLSQP',
                          options={'disp':True},
                           constraints=({ 'type': 'eq', 'fun': lambda inputs: 1.0 - np.sum(inputs)})).x

    return result


def alloc_function(alloc, df):

    trading_period_days = 252
    starting_value = 1000000

    #Normalize stock and apply allocation
    df_normalized = normalize_data(df)
    df_normalized_allocation = df_normalized * alloc

    # Calculate portfolio value by day
    starting_value = [1000000]
    # normalize portfolio
    df_normed_alloc_value = df_normalized_allocation * starting_value
    portfolio_values = df_normed_alloc_value.sum(axis=1)


    # Portfolio value by day
    portfolio_val = pd.DataFrame()
    portfolio_val['portfolio_value'] = portfolio_values
    |
    # Compute daily returns on the portfolio
    daily_returns_portfolio = compute_daily_return(portfolio_val)


    # Higher the sharp ratio the better (Reward / Risk) , so we should minimize the negative sharp ratio
    sharpe_ratio = -1 * np.sqrt(trading_period_days) * (daily_returns_portfolio.mean()/daily_returns_portfolio.std())
    return sharpe_ratio
```

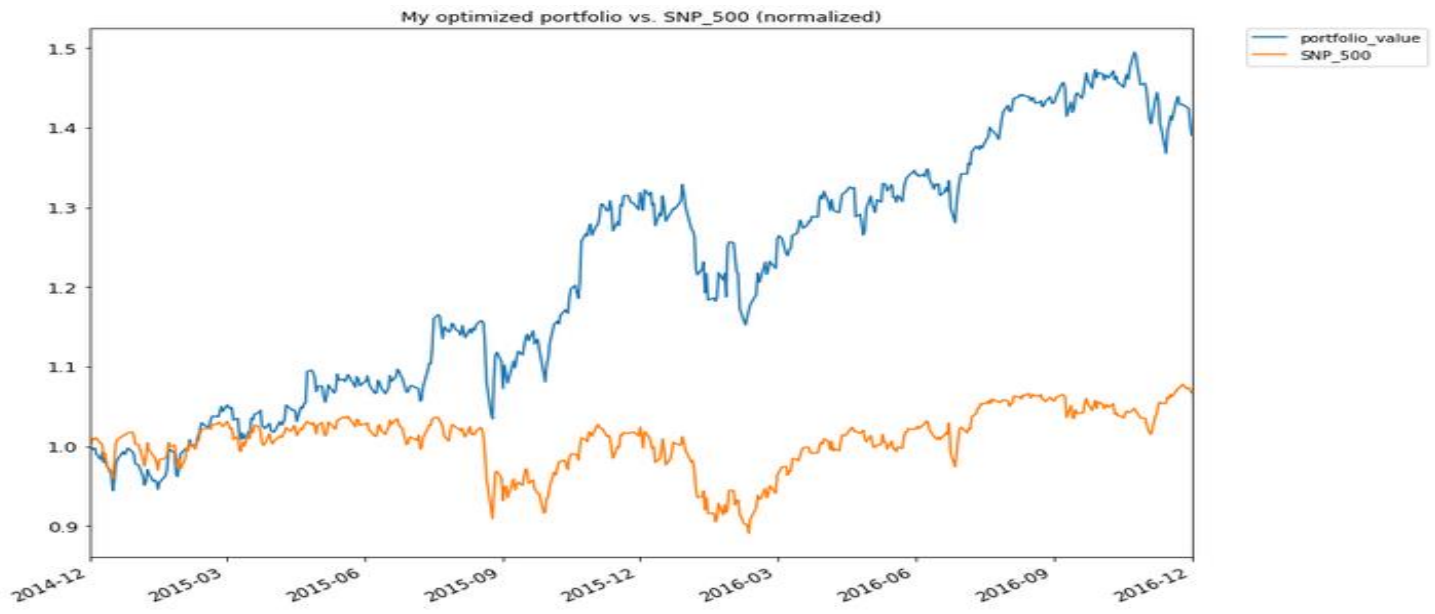
Optimal allocation below after Sharpe Ratio optimization:

```
Optimal Allocations:

GOOGLE: 0.000
AMAZON: 0.914
GOLD: 0.000
APPLE: 0.000
FACEBOOK: 0.086
MICROSOFT: 0.000
GENERAL ELECTRIC: 0.000
```

Following is the normalized Plotted Optimal Portfolio Statistics:



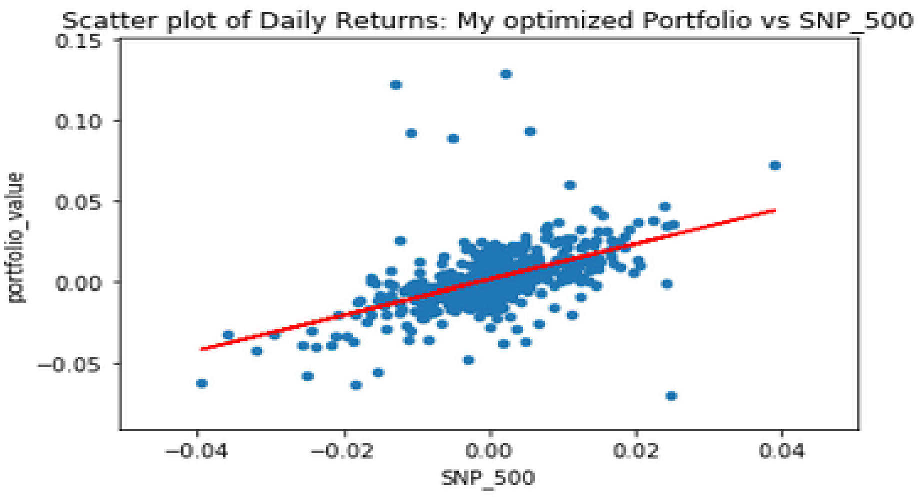
Following is the normalized Plotted "Optimal Portfolio Statistics" vs "SNP_500":



Portfolio Cumulative Return, Average Daily Return, Standard Deviation, Sharpe Ratio:

```
Cumulative Return:  portfolio_value    1.216763
dtype: float64
Average Daily Return:  portfolio_value    0.00176
dtype: float64
Daily Standard Deviation:  portfolio_value    0.019296
dtype: float64
Sharpe Ratio:  portfolio_value    1.448061
dtype: float64
```

Following is the Scatter plot between my optimized portfolio and SNP_500, please also refer the comments underneath the plot:



```
Beta APPLE = 1.08940744037
Alpha APPLE = 0.00157438228967
```

```
Aplha value is positive, so in an average my optimized portfolio is performing little bit
better than the SNP_500
```

Algorithms and Techniques

For this Capstone Project, I am using three supervised based algorithms – Linear Regression, KNN Regression, ARIMA (Auto Regressive integrated moving average for time series data)

Linear Regression:

In statistics, linear regression is an approach for modeling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variables) denoted X. The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regression.

Source: https://en.wikipedia.org/wiki/Linear_regression

Multiple Linear Regression represented in the form: $\mathbf{Y = m1X1 + m2X2 + m3X3...... + Slope}$

In this project for Stock Price Prediction for each stock the X attributes are: **'Open', 'High', 'Low', 'Close', 'Volume', 'Adj Close'**

After finding the Pearson co-relation among these attributes I found out that the attributes 'Open', 'High', 'Low', 'Close' are highly co-related. So, I am keeping only 'Open','Volume', 'Adj Close' as the X attributes

I am adding new column 'Adj_Close_req_Days_Later' as a target (Y) attributed to be predicted by the algorithm after training is done. This attribute is created as follows:

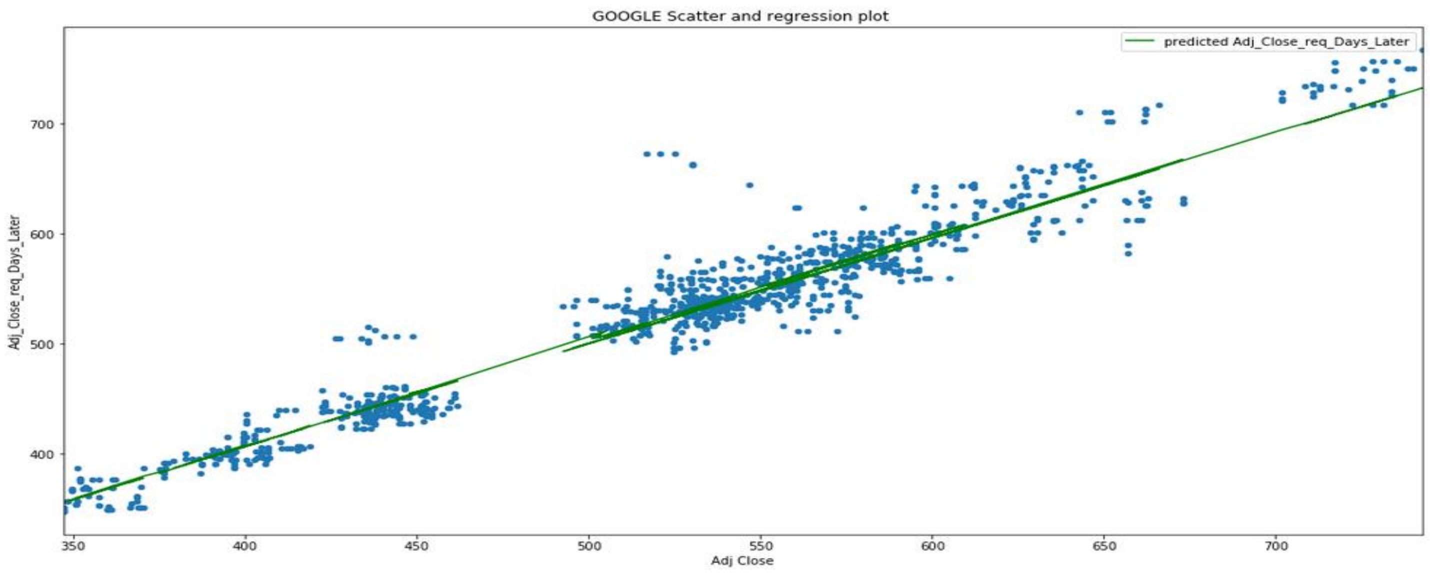
```
d[key]['Adj_Close_req_Days_Later'] = d[key]['Adj Close']
d[key]['Adj_Close_req_Days_Later'] = d[key]['Adj_Close_req_Days_Later'].shift(-days_shift)
```

Here, I want to build an algorithm that learns from prices of a stock in historical time and predicts the price of the stock for a date in future. In other words, the algorithm should model the relationship between the dependent variables such as 'Open','Volume', 'Adj Close' with the Stock price. So, the linear regression function or algorithm should learn from the relationship between the dependent variables and the stock prices producing the coefficients for each of the stock and the slope. For new values of dependent variables, we can them plug them in the linear regression equation and get the price or the predicted stock price. But the caveat in using linear regression for stock price prediction is it does not track the seasonality and the trend in the data well, it basically thinks the data points follow a linear path without paying attention to ups and dips in the stock prices. Another option is to use KNN regression, where it can relate to the behavior of the local data points rather than following a linear path.

I am printing out the training and test scores on the stocks, and the % +/- variation of the predicted stock value compared to the actual values. Additionally, I am also plotting the scatter plot between 'Adj Close' and 'Adj_Close_req_Days_laters' and the regression fitted line on the predicted 'Adj_Close_req_Days_laters'. For e.g:

```
Score on training data for GOOGLE :0.95
Score on test data for GOOGLE : 0.82


The prediction on the test dataset 10 days after the training date for GOOGLE is +/- 3.486% compared to the actual values
```

Scatter plot between 'Adj Close' and 'Adj_Close_req_Days_laters' and the regression fitted line on the predicted 'Adj_Close_req_Days_laters' for Google :



KNN Regression:

The k-Nearest Neighbors algorithm (or k-NN for short) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors. k-NN is a type of instance-based learning, where the function is only approximated locally and all computation is deferred until classification or Regression. The k-NN algorithm is among the simplest of all machine learning algorithms. Both for classification and regression, it can be useful to assign weight to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones.

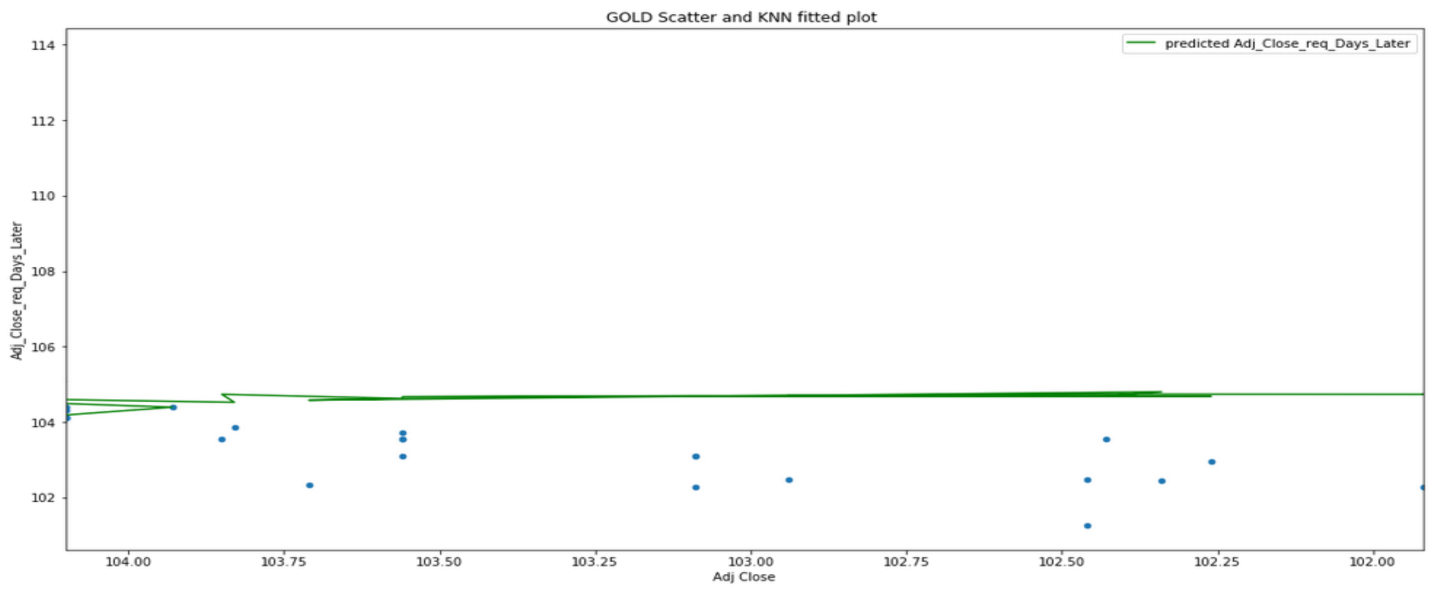
Source: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

In this case, for a given date in future the function would approximate the stock price based on the nearby values in the nearest past.

I am printing out the training and test scores on the stocks, and the % +/- variation of the predicted stock value compared to the actual values. Additionally, I am also plotting the scatter plot between ‘Adj Close’ and ‘Adj_Close_req_Days_laters’ and the regression fitted line on the predicted ‘Adj_Close_req_Days_laters’. For e.g:

```
Score on training data for GOLD :0.99
Score on test data for GOLD : 0.66
The prediction on the test dataset 1 days after the training date for GOLD is +/- 1.427% compared to the actual values
```

Scatter plot between ‘Adj Close’ and ‘Adj_Close_req_Days_laters’ and the regression fitted line on the predicted ‘Adj_Close_req_Days_laters’ for Gold :



ARIMA model:

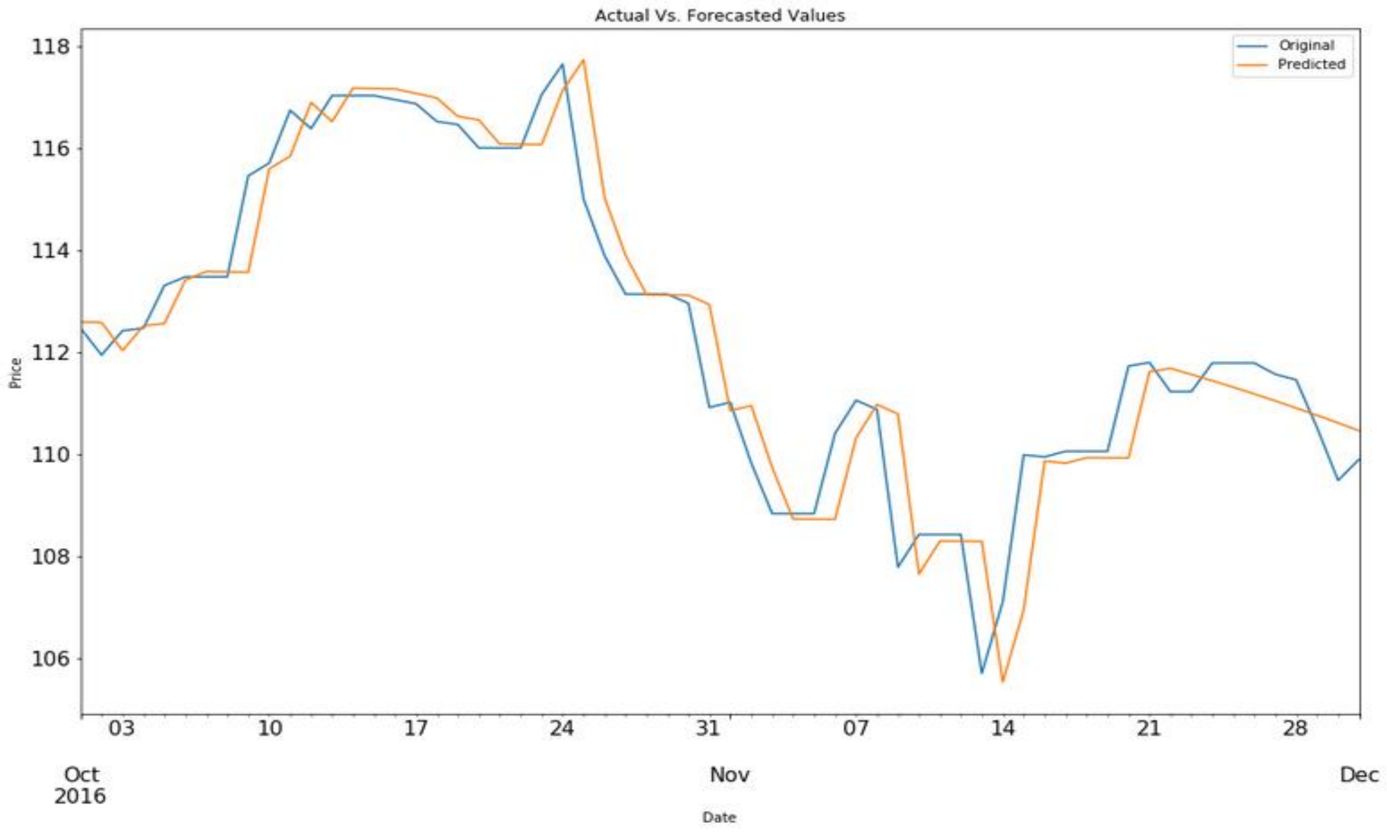
ARIMA stands for Auto Regressive Integrated Moving Average. There are two types ARIMA models – Seasonal and Non-Seasonal. I am using Non-seasonal ARIMA model in this Stock Price Prediction problem. Non-seasonal ARIMA models forecast future points based on the construction of three components, an autoregressive, AR, a differencing, I, and a moving average, MA, component. When we put it all together non-seasonal ARIMA models are displayed as ARIMA p ,d, q. The p, d and q values represent the number of periods to lag for in our ARIMA calculation. For e.g. if we say p =2, we will be using the two previous periods of the time series in the autoregressive portion of the calculation. This helps adjust the line fitted to forecast the series. Purely, AR models would look like a linear regression where the predictive variables are a certain number of previous periods which is represented by p. The differencing term refers to the process we use to transform a time series into a stationary one, which is a series without trend or seasonality. This process is called differencing and the 'd' refers to the number of transformation used in the process. The moving average term (MA) refers to the lag of the error component. The error component refers to the part of the time series not explained by trend or seasonality. MA models look like linear regression models where the predictive variables are the previous q periods of errors.

Three items should be considered to determine a first guess at an ARIMA model: a time series plot of the data, the ACF, and the PACF.

AR model: Identification of an AR model is often best done with the PACF. the theoretical PACF "shuts off" past the order of the model. The phrase "shuts off" means that in theory the partial autocorrelations are equal to 0 beyond that point

MA model: For an MA model, the theoretical PACF does not shut off, but instead tapers toward 0 in some manner. A clearer pattern for an MA model is in the ACF. The ACF will have non-zero autocorrelations only at lags involved in the model.

I am printing out the training and test scores of APPLE and a plot between actual and forecasted value of APPLE stocks. For e.g:



```
Train score  0.887920343472
Test score  0.472620104237
```

Sources:
Udacity Free Course: Time Series Forecasting

- https://onlinecourses.science.psu.edu/stat510/node/62
- https://onlinecourses.science.psu.edu/stat510/node/49
- https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average
- http://www.inertia7.com/projects/time-series-stock-market-python

Benchmark

For all the three models my benchmark is a R2 Score of 0.6 on test data (unseen data) and a R2 Score of 0.85 on train data. Please note if we pass a list of stocks in the "query_regressor" function for both Linear and KNN regression, my code builds individuals model for each stock. Each individual model for each must meet or exceed the benchmark.

Additionally, on an average the predicted prices from all the models must be more or less within +/- 5% of the actual stock prices.

Methodology

Data Preprocessing:

I have chosen these stocks for training and prediction: `'GOOGLE', 'AMAZON', 'GOLD', 'APPLE', 'FACEBOOK', 'MICROSOFT', 'GENERAL ELECTRIC'` as well as `'SNP_500'` index and `'GOLD'` for comparative analysis. I have manually downloaded the data from Yahoo finance, I went back as far as December 2011 until 1st week of December 2016. Please note that some stocks like `'Facebook'` did not even exist back in 2011.

I have built the following data preprocessing utility functions:

```
def symbol_to_path(symbol, base_dir="data"):

    """Return CSV file path given ticker symbol."""

    return os.path.join(base_dir, "{}.csv".format(str(symbol)))


def get_data(symbols, dates):

    """Read stock data (adjusted close) for given symbols from CSV files."""

    #Creating a DataFrame using 'Date' as Index values
    #Without this the Index will have parameters as 0,1,2...
    df = pd.DataFrame(index=dates)

    # Add SNP_500 for reference, if absent
    if 'SNP_500' not in symbols:
        symbols.insert(0, 'SNP_500')


    for symbol in symbols:

        #pd.read_csv : Reading the SNP_500.csv file
        #index_col : Date column in the csv file is used as index
        #parse_dates : Converting Dates present in the dataframe to be converted to Date-Time Index Objects
        #usecols : Using only 'Date' and 'Adj Close' columns and getting ride of others
        #na_values : to treat 'NaN' as Not-a-Number
        df_temp=pd.read_csv("Data/{}.csv".format(symbol), index_col='Date',parse_dates=True,usecols=['Date','Adj C
lose']
                            ,na_values=['nan'])

        #Renaming 'Adj Close' column to their 'Stock symbol'
        df_temp = df_temp.rename(columns={'Adj Close': symbol})

        #Using default how='left'
        df = df.join(df_temp)

        #Taking SNP_500 as reference and dropping NaN values
        if symbol == 'SNP_500':
            df = df.dropna(subset=["SNP_500"])

    return df
def plot_data(df, title="Stock prices", xlabel="Date", ylabel="Price"):

    """Plot stock prices with a custom title and meaningful axis labels."""

    axis = df.plot(title=title, fontsize=12,figsize = (15,10))

    #Setting X-Label
    axis.set_xlabel(xlabel)

    #Setting Y-Label
    axis.set_ylabel(ylabel)
    plt.show()
```

```
def plot_selected(df, columns, start_index, end_index, title="Stock prices"):
    """Plot the desired columns over index values in the given range."""
    df_new = df.ix[start_index : end_index,columns]
    ax = df_new.plot(title=title,fontsize=10,figsize = (15,10))
    ax.set_xlabel("Date")
    ax.set_ylabel("Price")
    plt.show()

def normalize_data(df):
    """Normalize data by uisng the first row of the dataframe"""
    return df/df.ix[0,:]

def get_rolling_mean(values, window):
    """Return rolling mean given values, using specified window size"""
    return pd.rolling_mean(values, window=window)

def get_rolling_std(values, window):
    """Return rolling standard deviation of given values, using specified window size"""
    return pd.rolling_std(values, window=window)

def get_bollinger_bands(rm, rstd):
    """Return upper lower bollinger bands."""
    upper_band = rm + rstd * 2
    lower_band = rm - rstd * 2
    return upper_band, lower_band

def compute_daily_return(df):
    daily_returns = (df / df.shift(1)) - 1
    daily_returns.ix[0, :] = 0
    return daily_returns[1:]

def cummulative_return(df):
    cummulative_return = (df.ix[-1] / df.ix[0]) - 1
    return cummulative_return
```

Explanation of some of the utility functions:

get_data: This function reads the csv files from the 'Data' directory and creates a DataFrame with 'Adj Close' price for each stock with Date as the index. It also replaces the columns with the Stock name with the Adj Close as the values for the respective stocks. Additionally, I am removing all the records from the DataFrame for which value for SNP_500 is 'nan' (we are already stock values when the market was open). I am also sorting the stocks with ascending order of dates. Within the main function I am so doing forward fill first and then backward for stocks which have 'nan' values (We cannot keep 'nan' values e.g. non-trading days like holidays or weekends):

```
    # Fill empty trade dates with forward filling dates first and then backward filling
df.fillna(method="ffill", inplace="True")
df.fillna(method="bfill", inplace="True")
```

normalize_data: This function returns the normalized DataFrame by normalizes the data by dividing each value of each stock by the starting value (starting date of the stock) or in a word diving each row by the first row for the corresponding columns.

get_rolling_mean: This function returns rolling mean given values, using specified window size.

get_rolling_std: This function returns rolling standard deviation of given values, using specified window size.

get_bollinger_bands: This function returns the Upper and lower Bollinger bands using the rolling mean and rolling standard deviation.

compute_daily_return: This functions returns the daily return of the stock compared to the previous date's price. Please note this will return 'nan' for the starting date as we don't 've the value for that stock in the previous date, so we need to set the first row of the Daily return DataFrame to '0'.

Implementation:

LINEAR REGRESSION:

I have used SKLEARN's Linear Regression to implement this algorithm.

class sklearn.linear_model.**LinearRegression**(*fit_intercept=True*, *normalize=False*, *copy_X=True* , *n_jobs=1*)

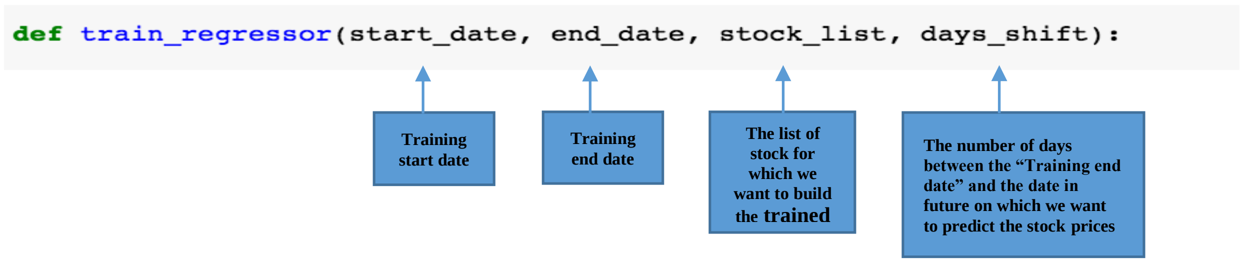
I have used the get_data function to preprocess the data fames for each stock. For training the algorithms, I have implemented the 'get_data' function in a little different way than the one I implemented for the initial data analysis as discussed earlier in this document:

```
def get_data(symbol, dates):
    df = pd.DataFrame(index=dates)
    df_temp = pd.read_csv("Data/{}.csv".format(symbol), index_col='Date',parse_dates=True,usecols=['Date','Open',
                                                                                                    'High', 'Low',
                                                                                                    'Close', 'Volume',
                                                                                                    'Adj Close']
                              ,na_values=['nan'])
    df = df.join(df_temp)
    df.fillna(method="ffill", inplace="True") # Forward fill empty trade dates
    df.fillna(method="bfill", inplace="True") # backfill empty trade dates
    return df
```

Please note that I am passing a stock and not a list of stocks to this function, pulling all attributes for that stock in the DataFrame. I want to see how the attributes are co-related to one another and use the attributes which are least co-related to build the model. My goal is to build a model for each stock for the list of stocks passed in the "query_regresssor" function. I 've also implemented few extra functions: "train_regressor", "query_regressor" & "train_classifier_gridsearch"

Explanations of the new functions implemented:

"train_regressor":

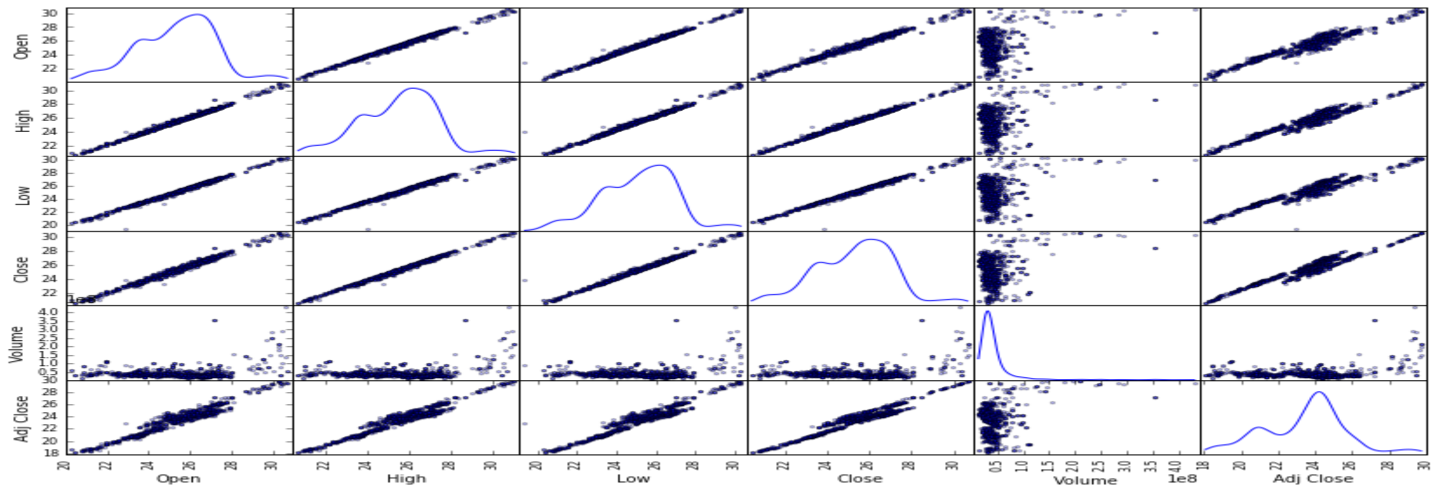


```
def train_regressor(start_date, end_date, stock_list, days_shift):
```

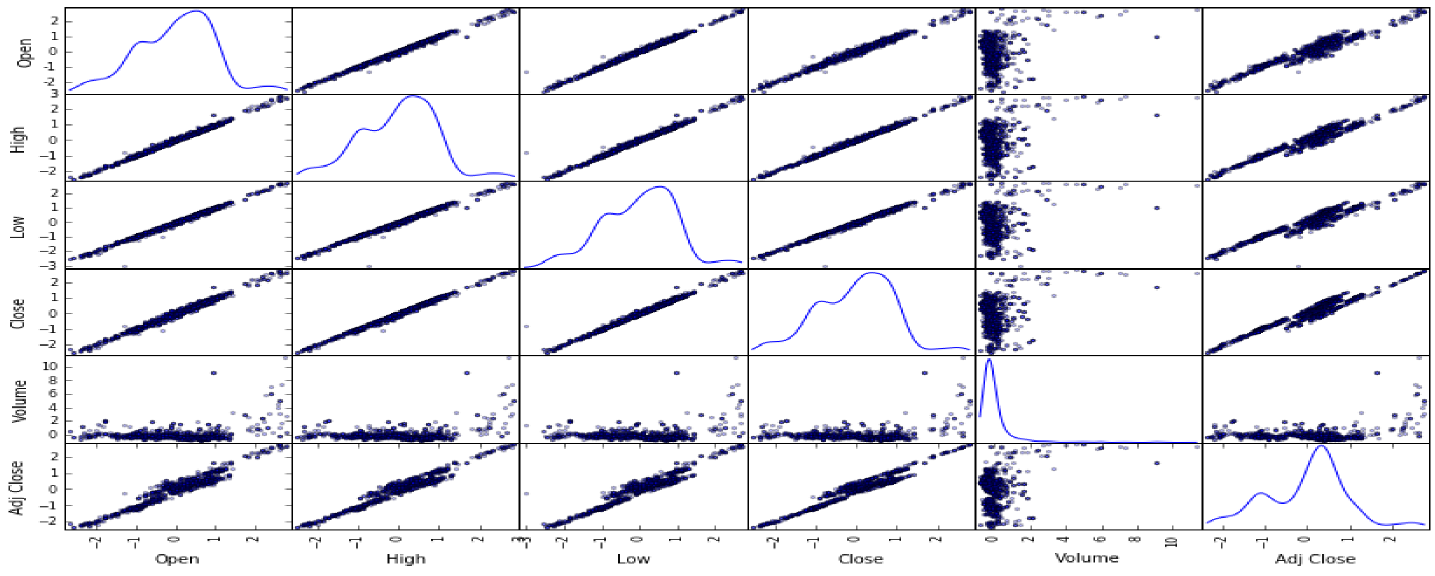
Training start date
Training end date
The list of stock for which we want to build the trained
The number of days between the "Training end date" and the date in future on which we want to predict the stock prices

This function primarily returns a dictionary of trained models(with key as the stock name and value as the trained regressor) for all stocks in the list and also performs analysis on the attributes such as how are they co-related to one another and as well as their distributions. I could find out that the attributes ['Open', 'High', 'Low', 'Close'] are highly correlated so I decided to keep 'Open', 'Volume', 'Adj Close' for each stock. The distributions for most of the attributes for most of the stocks are non-Gaussian, although few of them are close to normal. Nevertheless, I used the SKLEARN's "StandarScaler' for all of them to convert them to zero mean normal distributions.

Below is the distribution of GOOGLE's attributes before performing standardization:



Below is the distribution of GOOGLE's attributes after performing Standardization:



I am doing a manual test train split on the data as we need to the split in the ascending order of dates in the "stock price prediction world". The Sklearn's test train split module performs randomized test train split, we must not get into a situation where we are testing on datasets which is prior to trained datasets. Please refer below the code snippet to perform the manual test train split:

```
X_train = d[key].iloc[0:train_num,:-1]
scaler.fit(d[key].iloc[0:,:-1])
X_train = scaler.transform(X_train)
y_train = d[key].iloc[0:train_num, -1]
X_test = d[key].iloc[train_num:num_days,:-1]
X_test = scaler.transform(X_test)
y_test = d[key].iloc[train_num:num_days, -1]
```

Just re-iterating, please also note that I am engineering the feature "Adj_Close_Req_Days_Later". Here "days_shift" is the number of days between the last training date and the date on which I want to predict the stock price in future:

```
d[key]['Adj_Close_req_Days_Later'] = d[key]['Adj Close']
d[key]['Adj_Close_req_Days_Later'] = d[key]['Adj_Close_req_Days_Later'].shift(-days_shift)
```

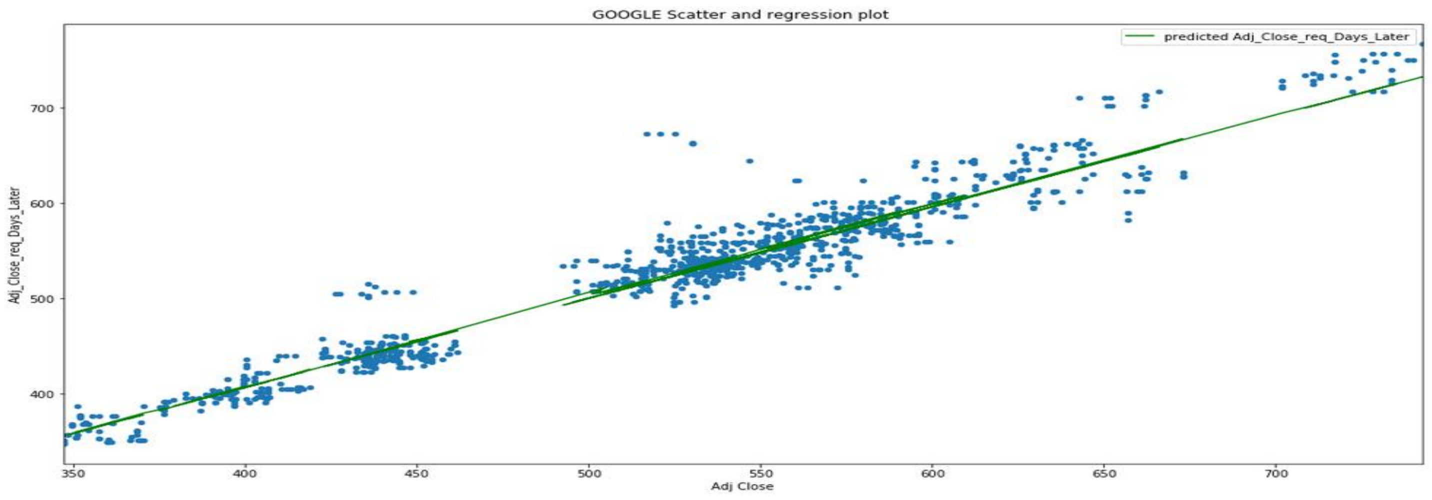
I also tried to peform hyperparmeter tuning for few parameters such as 'fit_intercept', 'normalize', 'copy_X' on the Sklearn's Linear Regression module but didn't see any difference, please refer below the parameters:

```
params = {'fit_intercept':[True, False],
          'normalize' :[True, False],
          'copy_X': [True, False],
          'n_jobs':[-1]}
```

I am printing out the training and test scores on the stocks, and the % +/- variation of the predicted stock value compared to the actual values. Additionally, I am also plotting the scatter plot between 'Adj Close' and 'Adj_Close_req_Days_laters' and the regression fitted line on the predicted 'Adj_Close_req_Days_laters'. For e.g:

```
Score on training data for GOOGLE :0.95
Score on test data for GOOGLE : 0.82


The prediction on the test dataset 10 days after the training date for GOOGLE is +/- 3.486% compared to the actual values
```

Scatter plot between 'Adj Close' and 'Adj_Close_req_Days_laters' and the regression fitted line on the predicted 'Adj_Close_req_Days_laters' for Google



KNN REGRESSION:

The steps for KNN Regression is ditto to Linear Regression except that I am using SKLEARN's KNN Regression with default parameters:

```
class sklearn.neighbors. KNeighborsRegressor (n_neighbors=5, weights='uniform', algorithm='auto',
leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=1, **kwargs)
```

Additionally, since the R2 scores on test data for most of the stocks using the default parameters are terrible, I thought of using gridsearch to find the best tune parameters for KNN Regressor, please refer below the code snippet for hyper parameter tuning:

```
params = {'n_neighbors': range(1, 50, 5),
          'leaf_size': range(1, 50, 5),
          'algorithm' : ['auto'] ,
          'weights': ['uniform', 'distance']}

scorer = make_scorer(r2_score)

regr = train_classfier_gridsearch(regr, params, scorer, X_train, y_train)
```

R2 Score on Google to predict the date which is one day ahead in future compared to the last training date without gridsearch:

```
Score on training data for GOOGLE :0.98
Score on test data for GOOGLE : -23.66
The prediction on the test dataset 1 days after the training date for GOOGLE is +/- 10.513% compared to the actual values
```

After using grid search as described above, astonishingly my R2score on test data is even poor but the score on trained data has improved a lot, please refer below. This typical sign of overfitting. So, overall I would think KNN is a bad algorithm for Stock price prediction:

```
Score on training data for GOOG :0.98
Score on test data for GOOG : -21.83
The prediction on the test dataset 1 days after the training date for GOOG is +/- 10.111% compared to the actual v
alues
```

ARIMA:

I have tried ARIMA with APPLE stock to predict the stock price one day ahead in future, please refer the function "**exec_arima**" in the header below in the "Analysis.ipynb" notebook:

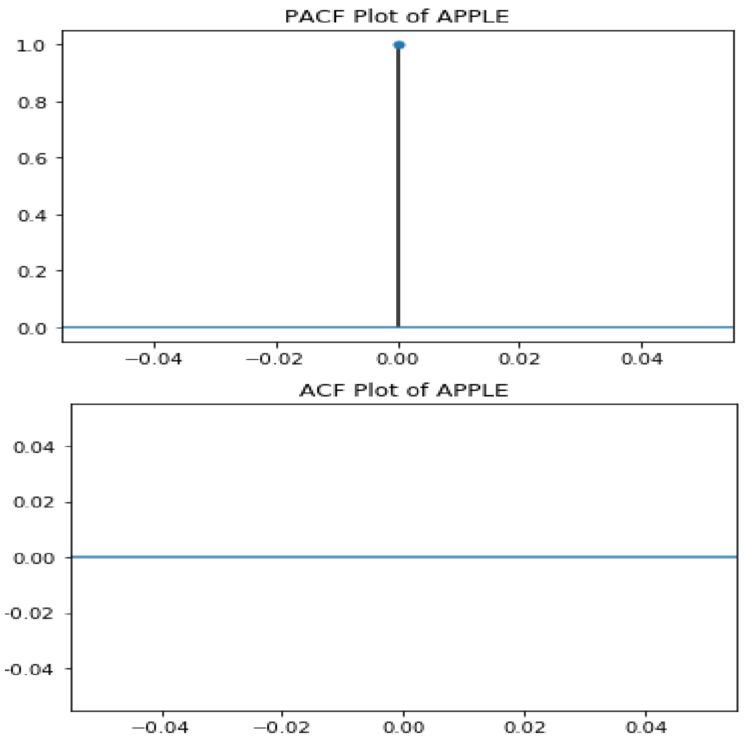
```
import pandas as pd
import numpy as np
from statsmodels.tsa.arima_model import ARIMA, ARIMAResults
#from statsmodels.tsa.stattools import acf, pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import matplotlib.pylab as plt
#import matplotlib.pyplot as plt
import matplotlib.dates as dates
from sklearn.metrics import r2_score
```

Please note for ARIMA I am just using 'ADJ CLOSE' as the attribute and since I am trying to predict the stock price one ahead I am adjusting the 'ADJ CLOSE' as below:

```
df['Adj Close'] = df['Adj Close'].shift(-1)
```

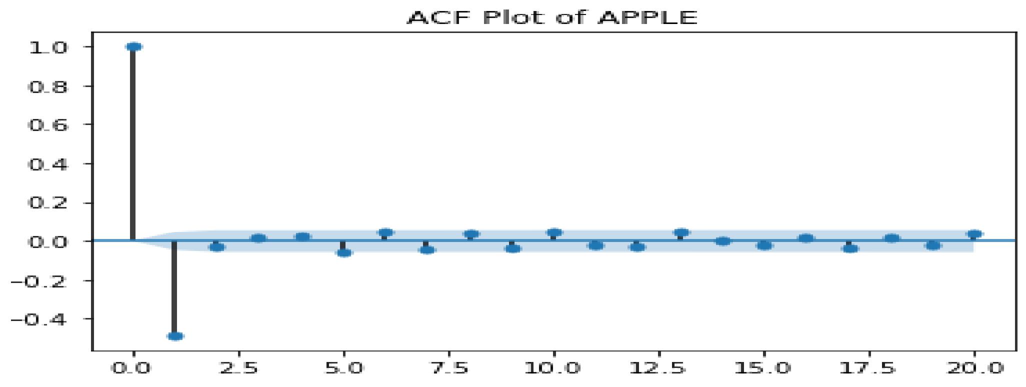
I started with diagnosing the AR, I and MA terms for the ARIMA model, please refer below the code snippet as well the ACF and PACF plots:

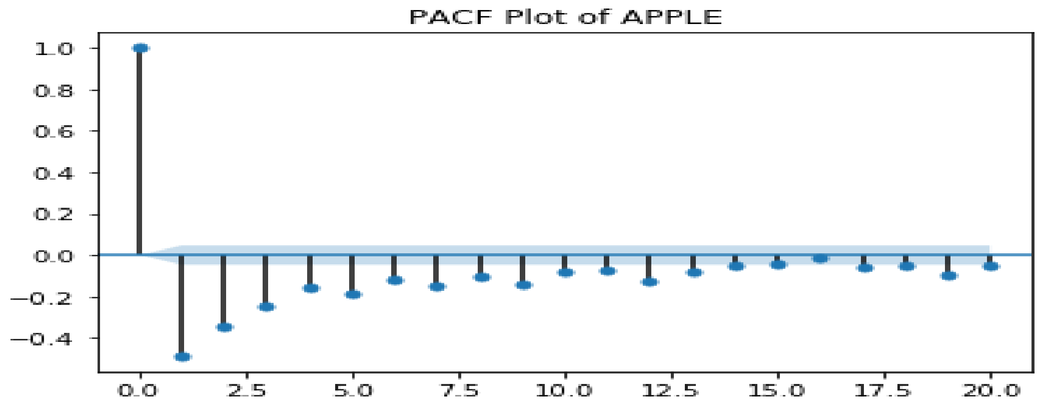
```
##DIAGNOSING THE ACF AND PACF PLOTS
acf = plot_acf(df, lags = 20)
plt.title("ACF Plot of APPLE")
acf.show()

pacf = plot_pacf(df, lags = 20)
plt.title("PACF Plot of APPLE")
pacf.show()
```



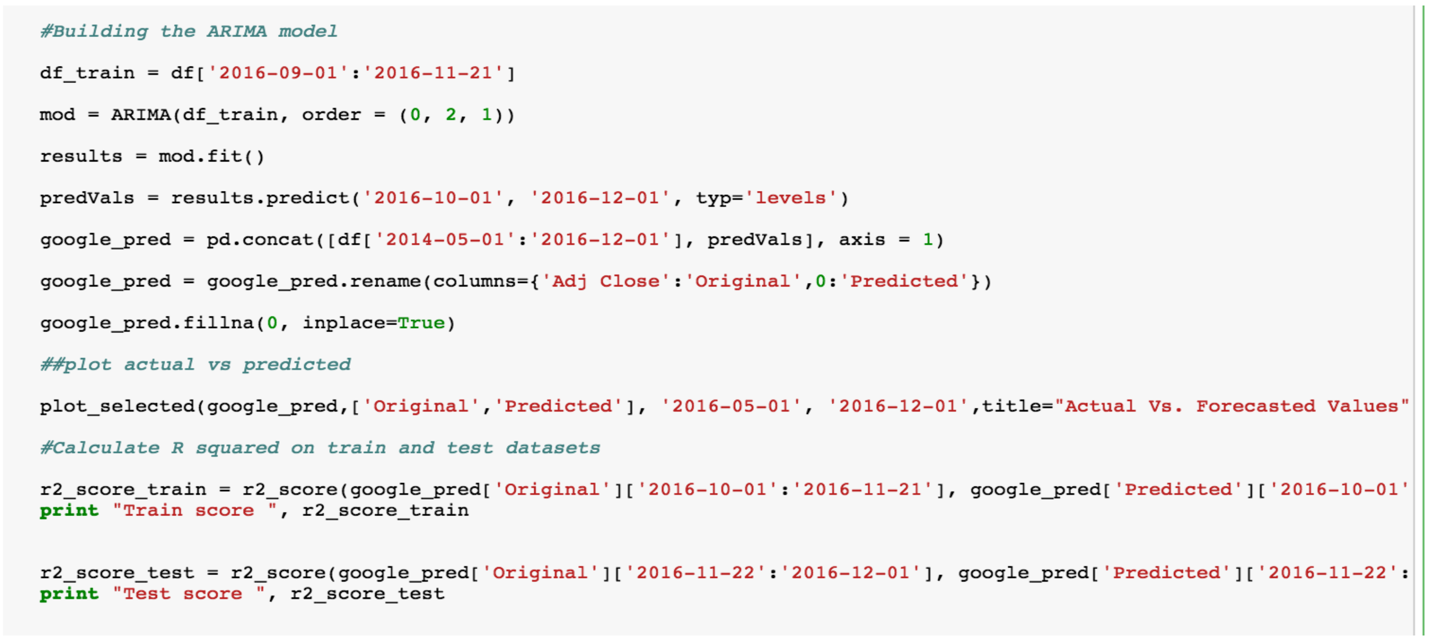
The ACF plot above suggests there is no auto-correlation, so I decided to do 1 lag differencing which is not helping much either, so I went ahead with 2 lag differencing. Please refer the code snippet as well the ACF and PACF plots below:

```
    #The ACF graph suggest there is constant auto co-relation. Lets do 1 lag differencing and dig further to confirm th
    #1 lag differencing is not helping much and so decided to do 2 lag differencing

    df_diff = df- df.shift()
    df_diff = df_diff - df_diff.shift()
    df_diff.fillna(method="ffill", inplace="True") # Forward fill empty trade dates
    df_diff.fillna(method="bfill", inplace="True") # backfill empty trade dates


##DIAGNOSING THE ACF AND PACF PLOTS for DIFF
    acf = plot_acf(df_diff, lags = 20)
    plt.title("ACF Plot of APPLE")
    acf.show()

    pacf = plot_pacf(df_diff, lags = 20)
    plt.title("PACF Plot of APPLE")
    pacf.show()
```

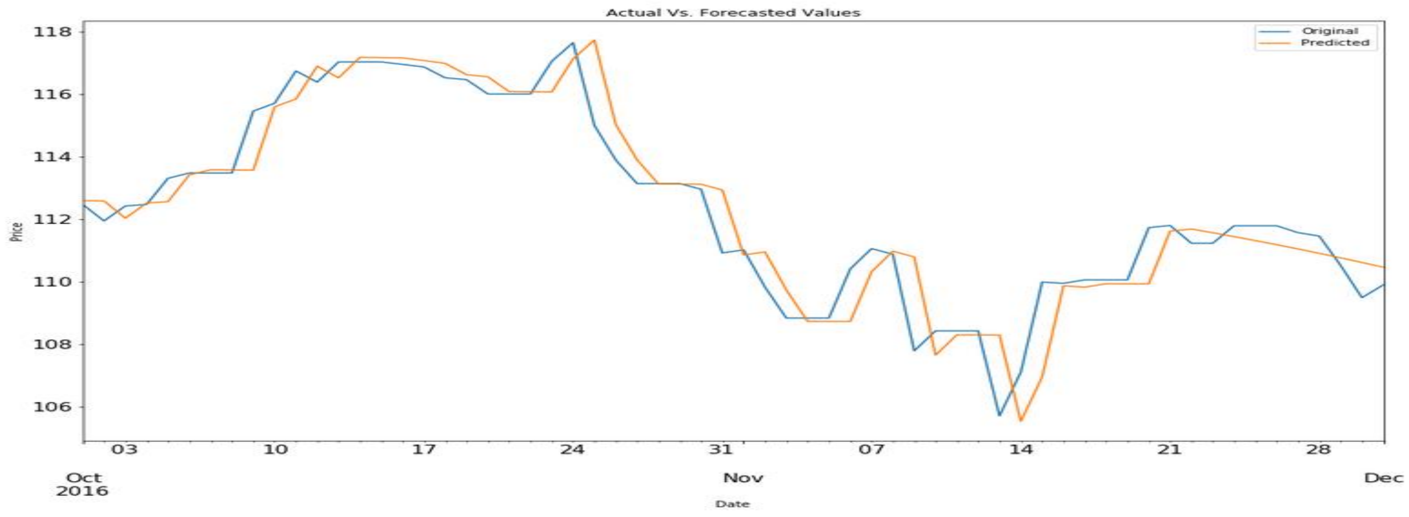




As we can see, the PACF plot is slowly tapering towards ‘0’. So it’s basically a MA model based on the ACF plot with q =2 as the non-zero auto correlation disappears after lag 2. So, this ARIMA has p=0, d=2, q=1.
Please refer the code snippet below to observe how I kept a slice of the dataset for training, predicted on a slice of the dataset part of which is also include in the training set. This would help me observe how the ARIMA model performs in both train and test dataset:



```
#Building the ARIMA model

df_train = df['2016-09-01':'2016-11-21']

mod = ARIMA(df_train, order = (0, 2, 1))

results = mod.fit()

predVals = results.predict('2016-10-01', '2016-12-01', typ='levels')

google_pred = pd.concat([df['2014-05-01':'2016-12-01'], predVals], axis = 1)

google_pred = google_pred.rename(columns={'Adj Close':'Original',0:'Predicted'})

google_pred.fillna(0, inplace=True)

##plot actual vs predicted

plot_selected(google_pred,['Original','Predicted'], '2016-05-01', '2016-12-01',title="Actual Vs. Forecasted Values"

#Calculate R squared on train and test datasets

r2_score_train = r2_score(google_pred['Original']['2016-10-01':'2016-11-21'], google_pred['Predicted']['2016-10-01'
print "Train score ", r2_score_train


r2_score_test = r2_score(google_pred['Original']['2016-11-22':'2016-12-01'], google_pred['Predicted']['2016-11-22':
print "Test score ", r2_score_test
```



Please observe the plot above - the 'green' plot almost perfectly follows the original 'blue' plot for the trained dataset. Towards the end of the plot, please observe how the green plot follows the 'blue' plot for the test dataset.

The train score: 0.887920343472
The test score: 0.472620109556

Refinement:

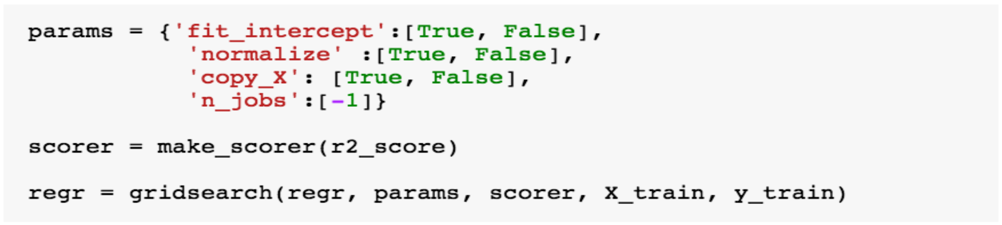
The process of improving upon the algorithms and techniques used is clearly documented. Both the initial and final solutions are reported, along with intermediate solutions, if necessary.
Among the three algorithms, the "Linear Regression" model has the best score. I tried to improve the R2 score by performing hyper parameter tuning on the parameters, but I am getting same scores.

Scores without grid search:

Score on training data for GOOG :0.95
Score on test data for GOOG: 0.82

Score with grid search:
Score on training data for GOOG :0.95
Score on test data for GOOG: 0.82

This observation is expected for Linear Regression as there not parameters to tune. Please refer the code snippet below:



```
params = {'fit_intercept':[True, False],
          'normalize' :[True, False],
          'copy_X': [True, False],
          'n_jobs':[-1]}

scorer = make_scorer(r2_score)

regr = gridsearch(regr, params, scorer, X_train, y_train)
```

As discussed above in the implementation section "KNN regression" model is terrible (score on test data is negative), gridsearch for hyper parameter tuning didn't help at all.

For ARIMA (please refer the ARIMA code), I got a score of 0.47 for 'APPL' to predict the stock price one day ahead in future. This score is below the benchmark score of 0.6

Results:

Model Evaluation and Justification:

By far, based on R2 score and +/- % average prediction accuracy compared to actual values, I found the **Linear Regression** model is much better off compared to KNN Regression and ARIMA for all the stocks I am chosen for this project. Although the Linear Regression model's accuracy decreases when the number of days from the last training date to the date to be predicted increases, up to **10 days in future**, the R2 score on trained datasets for all the stocks taken into consideration is more or less 0.9 and R2 score on the test datasets is more than 0.6 . On an average the predicted prices are also **more or less within +/- 5%** of the actual stock prices up to **10 days** stock price prediction in future. Hence, the model is a robust model and the results from it can be trusted.

Non-Seasonal ARIMA showed some promising results, for Apple Stock its R2 score on the train dataset **1 day** out in future is **0.88** but the score on the unseen test dataset is **0.47** and missed the benchmark of 0.6 on test dataset. I could have tried out the Season ARIMA to bring in seasonality into consideration but I would keep it for a later time.

Hence, the final model chosen for this problem is Linear Regression model for which the R2 score on trained datasets for all the stocks taken into consideration is more or less 0.9 and R2 score on the test datasets is more than 0.6. On an average the predicted prices are also within +/- 5% of the actual stock prices up to 10 days stock price prediction in future which is reasonable and aligns itself with solution or benchmark expectations.

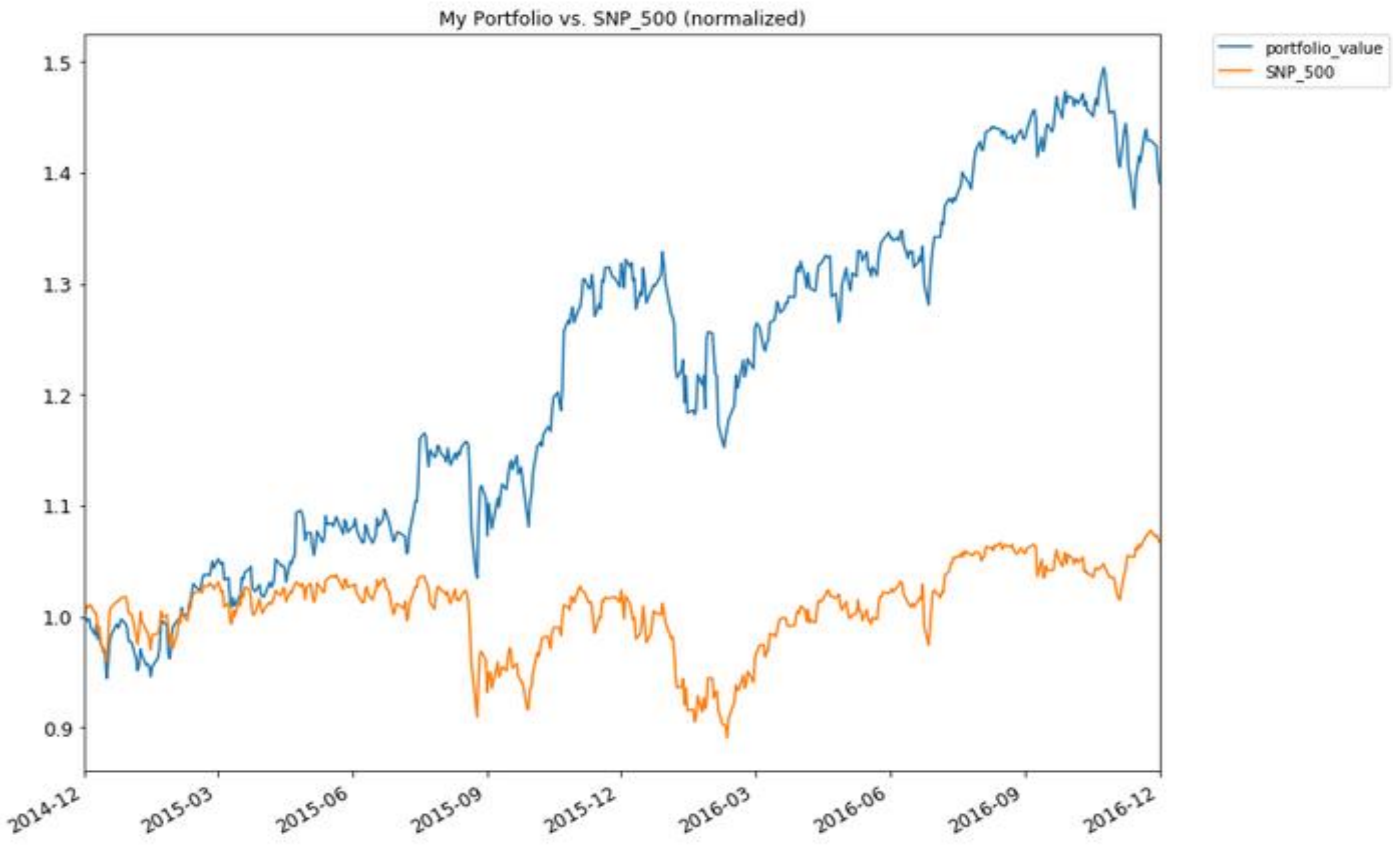
Conclusion

Free form visualization, Reflection and Improvement:

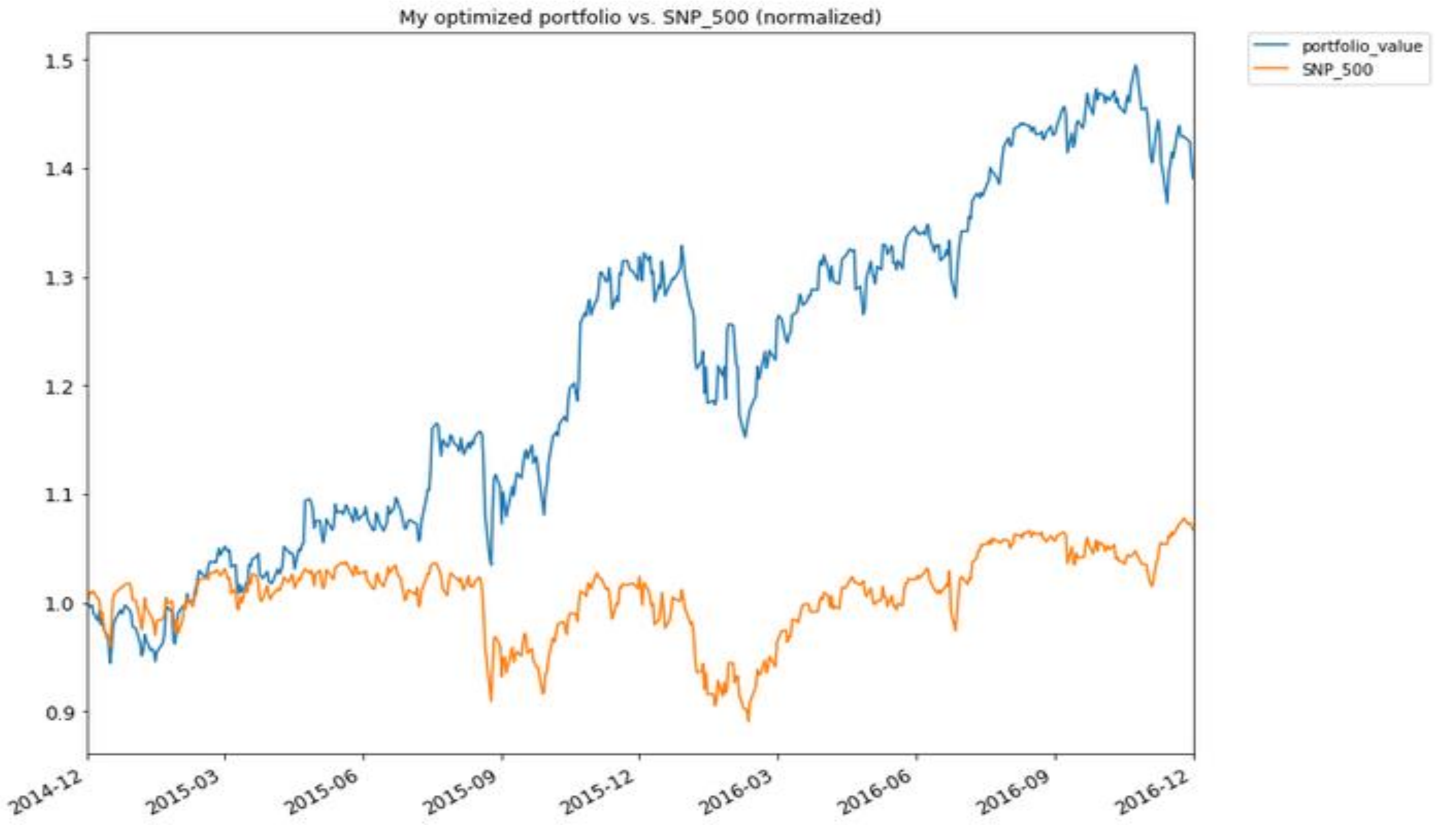
As discussed in the Data Exploratory and Exploratory visualization section, the following two visualizations on: normalized historical Portfolio Data **vs** normalized SNP_500 index seems to be very interesting for me. The first one is without portfolio optimization but equal allocation to all stocks- Fund allocations by symbol: symbols [`'GOOGLE', 'AMAZON', 'GOLD', 'APPLE', 'FACEBOOK', 'MICROSOFT', 'GENERAL ELECTRIC'`], allocation = [0.166, 0.166, 0.166, 0.166, 0.166, 0.166, 0.166]

The second is based on allocation using Sharpe ratio optimization:
Fund allocations by symbol: symbols = [`'GOOGLE', 'AMAZON', 'GOLD', 'APPLE', 'FACEBOOK', 'MICROSOFT', 'GENERAL ELECTRIC'`],
allocation = [`0.0, 0.914, 0.0, 0.0, 0.086, 0.0, 0.0`]

It's evident that performance of the optimized portfolio is better off compared to the equally allocated portfolio:



Capstone Project on Investment and Trading: **Dipak Majhi**
Build a Stock Price Indicator
Machine Learning Engineer Nanodegree **17th June, 2017**



Overall Summary:

Built a machine learning models that learn from historical stock price attributes and predict the stock price on a future date (any day after the last training date), I am only predicting the future Adjusted Closing pricing. Since the target variable is a continuous value so this is a regression task.
Built a few models such as [Linear Regression](), [KNN Regression](), [ARIMA]() which learn from historical stock data, attributes such as: Open, High, Low, Close, Volume, Adjusted Closing and "Adjusted Closing price" n days ahead in future.

In the solution, I build a Python based machine learning regression model for each stock provided, where each model learns from the historical data of the stock between user provided start date and end date, and predicts the 'Adj Close' price provided future date which must be greater than the end date. The data between the 'Start date' and 'End date' is divided into two parts in the ascending order of the dates, first 70% would be kept for training and the rest 30% for testing. The machine learning regression model must have a R2 score of 0.85 or above on the trained dataset and R2 score of 0.6 and above on the test dataset. Additionally, on an average the predicted prices from all the models must be more or less within +/- 5% of the actual stock prices on the test dataset.

The model is built on the historical data, it predicts the Adjusted Closing price for a date (only one date) which is n days ahead in future from the last date of training. Splitting the datasets has to be done in the ascending order of dates manually, and not using the SKLEARN's test train split module which randomly split the data. In the stock price prediction world, we must not train a model on future datasets and try to predict the stock prices for past dates

Overall, I enjoyed throughout this project, there are two interesting aspects I enjoyed the most –

i) The financial data analysis and its implementation in Python which I learnt throughout the course "Machine Learning for Trading"
ii) I got the chance to get my hands dirty on Time Series Forecasting. I learnt techniques like ARIMA and exponential smoothing. Courtesy: Udacity course: "Time Series Forecasting"

I believe stock price prediction is a mature and old problem, I am certain many companies have built mature machine learning models and operationalized them on real-time software engineering infrastructure.

For me, to build a better model in future so that I can predict stock prices accurately for months ahead and not just days, I am thinking of using **Exponential Smoothing** as discussed in the "**Time Series Forecasting**" by apply **error**, **trend** and **seasonality** in the smoothing method calculation. I feel stock prediction is more of a time series prediction problem than just a regression problem, each stock would have its own trend & seasonality. I also have plans to implement an **ensembled** learning technique which mostly gives better results than a single learner because it cancels the biases of the individual learners. I also have great enthusiasm to use the techniques of **Reinforcement** learning, as there many other external parameters which a smart self-learned model would want to learn from – such as current news on economic, political, geographical, climatic situations around the country, world etc. which affects the stock market.