**1. Definition:**

**Project Overview:**

I have chosen the Investment and Trading Capstone project provided as one of the options. I wanted to use my Machine Learning knowledge to solve a real problem and where I can learn the domain too. I have always been interested in investment and trading as an amateur and now I want to use my machine learning knowledge and see if it could help in predicting stock prices in the short-term future as strategy for good investment. I extensively took help of the free courses – “Machine learning for Trading” and “Time Series Forecasting” for this Capstone Project. Based on the instructions provided in “Machine Learning free course”, 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 day, 14 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 (The free course “Time Series Forecasting” gave me a pretty good background on Time series data analysis and prediction)

Datasets:

I have downloaded the following Datasets from yahoo finance: for e.g. – Apple, Google, Amazon, Facebook, Ge, Microsoft, SPDR Gold Shares (GLD), 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.

**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.

**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 (Source - “Machine learning for Trading” Udacity course)

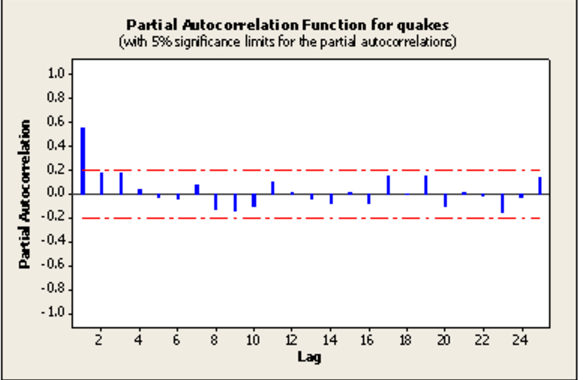
Non-seasonal ARIMA model: Non-seasonal ARIMA model is based on three terms: AR – Autoregressive - P, I – differencing - d, MA – Moving Average - q.  [Parameters](https://en.wikipedia.org/wiki/Parameter) p, d, and q are non-negative integers, p is the order (number of time lags) of the [autoregressive model](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/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.

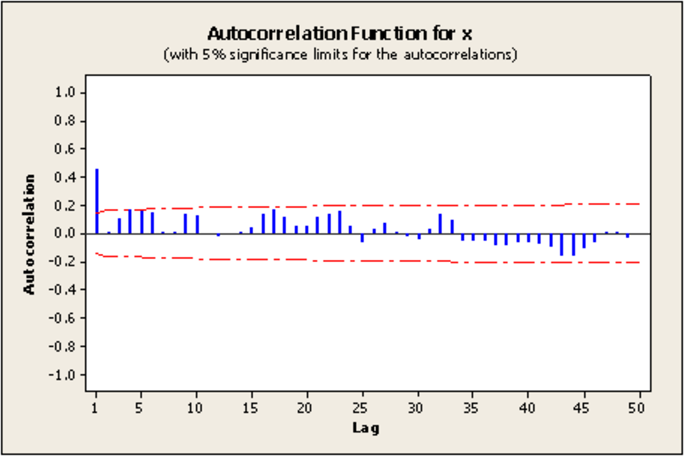
AR model: **Identification of an AR model is often best done with the PACF.** the theoretical PACF “shuts off” past the order of the model.  The phrase “shuts off” means that in theory the partial autocorrelations are equal to 0 beyond that point

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.

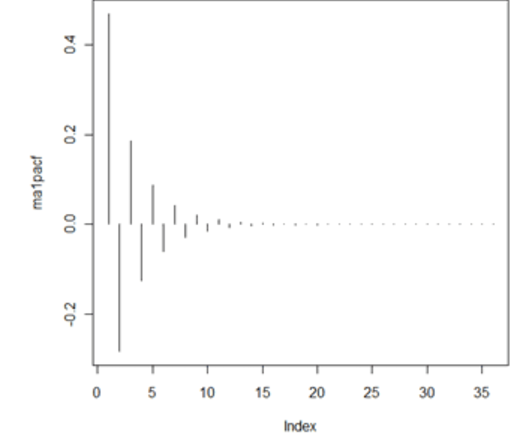
Sample PACF for an AR model below where the PACF shuts off past the 1st order so, AR =1 in this case:



Sample ACF for a MA model below where MA =1



Theoretical PACF where MA =1



Sources:

<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. Besides R2 score, I am also calculating the variation of the predicted stock price compared to the actual prices in the test dataset. 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 GOOG:0.97

Score on test data for GOOG: 0.88

The prediction on the test dataset 6 days after the training date for GOOG is +/- 2.721% compared to the actual values

Explaining R2 Score:

In [statistics](https://en.wikipedia.org/wiki/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.

Source:

<http://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html>

<http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression.score>

<https://en.wikipedia.org/wiki/Coefficient_of_determination>

**2. Analysis:**

**Data Exploration:**

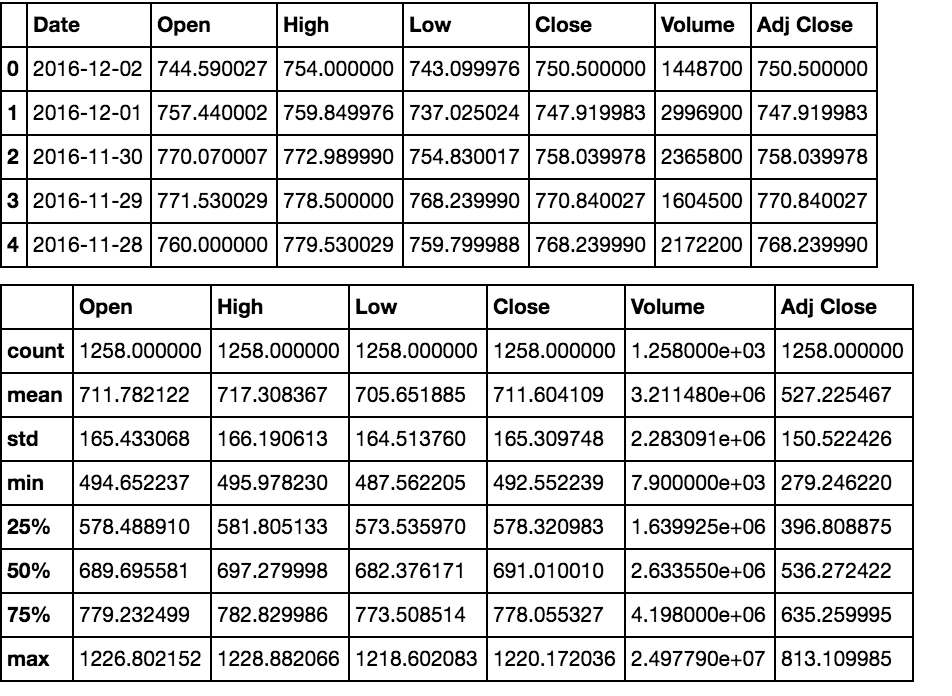
I have downloaded daily historical stocks data from Yahoo Finance between 12-05-2011 and 12-02-106 for the following stocks along with SNP 500(American Market index, I want to see how these stock trend with respect to market trends explained by SNP 500):

* 'AAPL', 'GOOG', 'AMZN', 'FACEBOOK', 'GE', 'GLD', 'MICROSOFT'

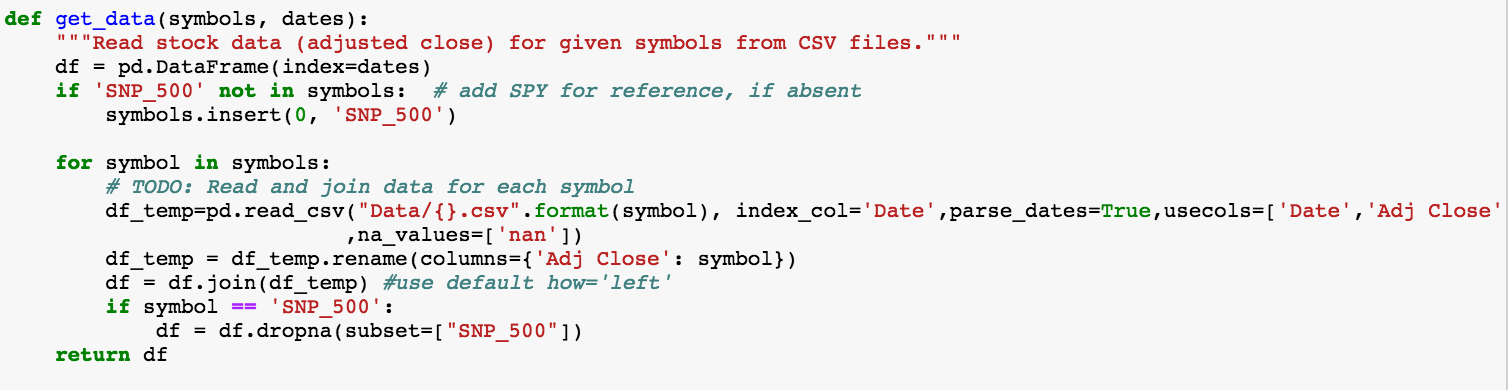
Each of this stock has the following attributes:

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

Sample Google Data and as well as the statistical description of the dataset:



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:



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:



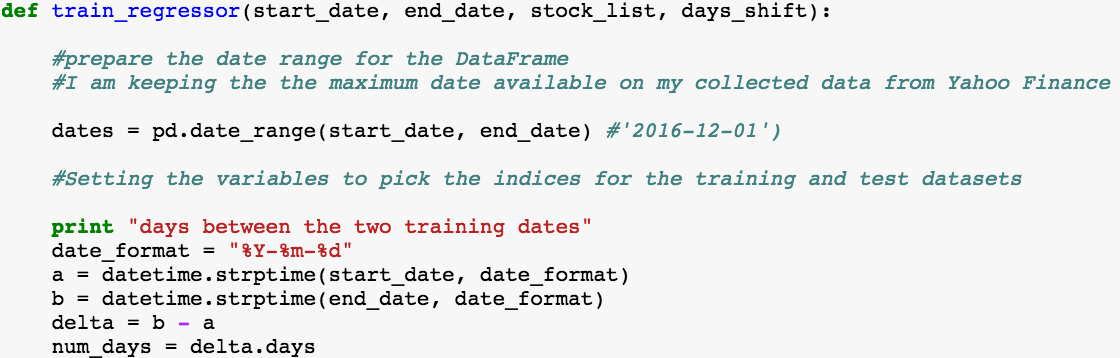
**The date on which we want to predict the ‘Adj Close’ price**

**Training end date**

**Training start date**

**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:



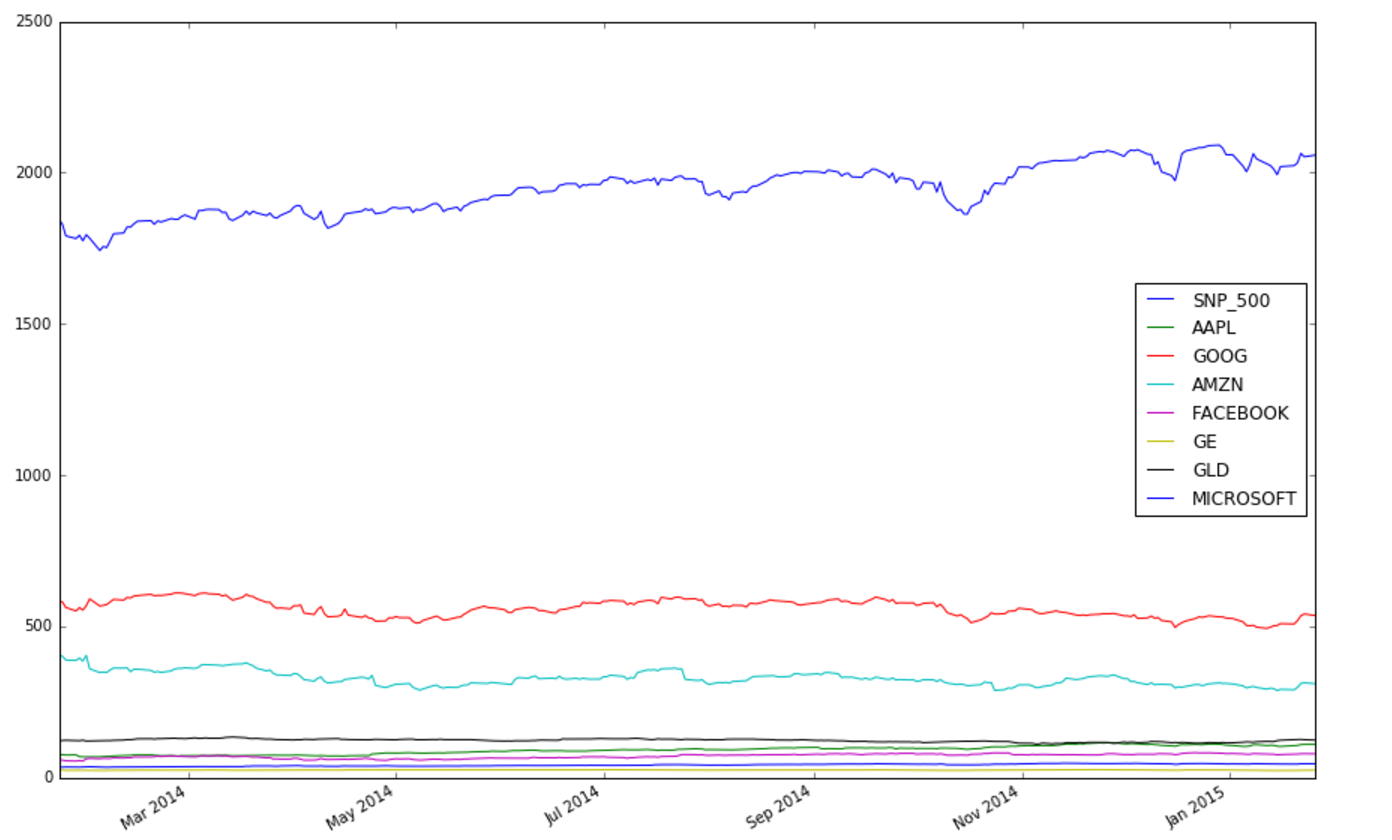
If there are non-trading days, for e.g. Saturday, Sunday or holidays, we want to forward fill first and then backward filling, code snippet below:



**Exploratory Visualization:**

Please refer the ipython notebook “All\_Analysis.ipynb” for all exploratory visualization code:

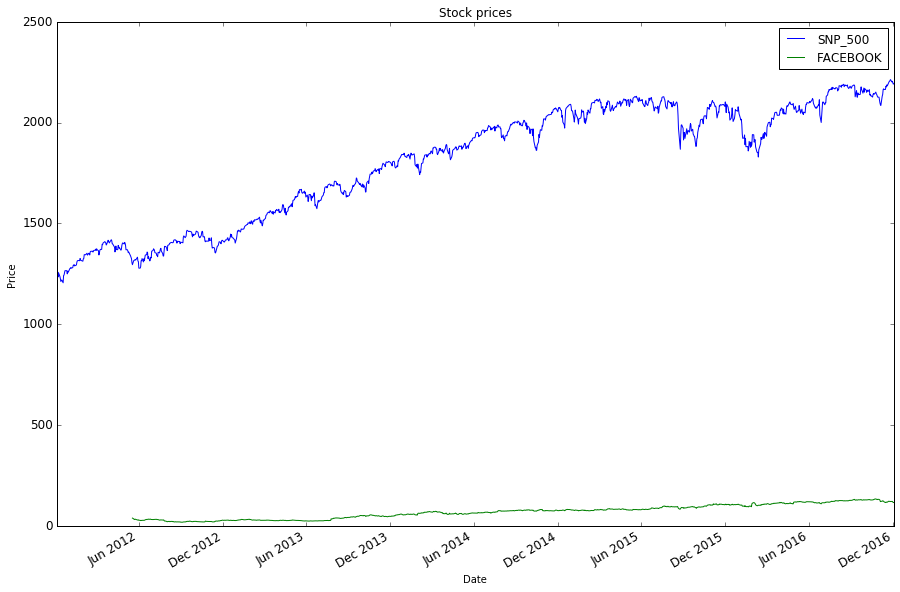
**HEADER 1: “Plotting Adj Close price for all stocks in comparison to the SNP 500 index”:**



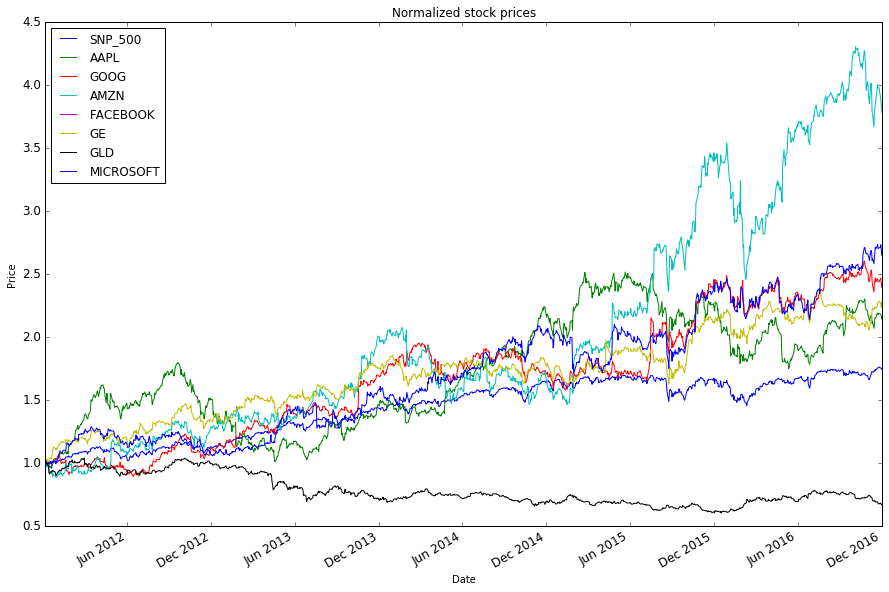
**HEADER 2:**

**“Further Plotting and Analysis”:**

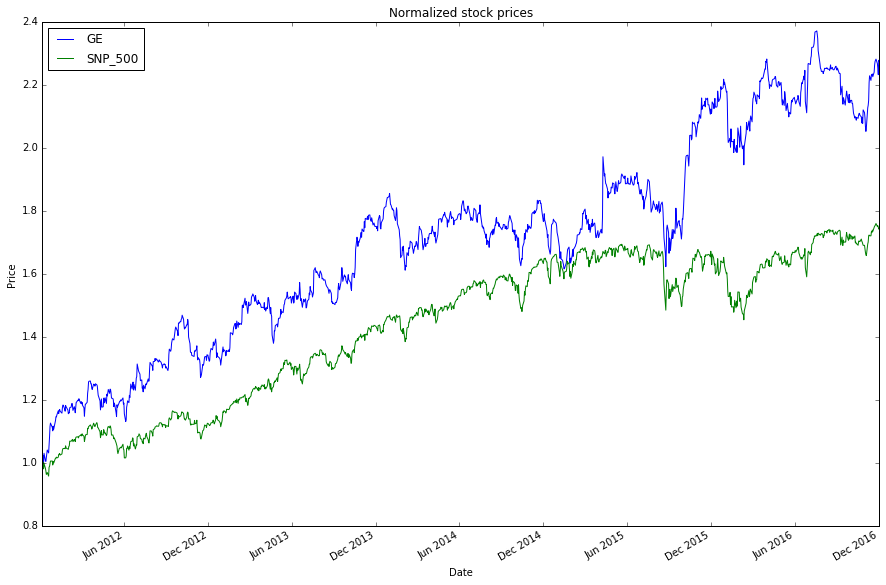
Please find below the plot- Comparison SNP\_500 vs FACEBOOK:



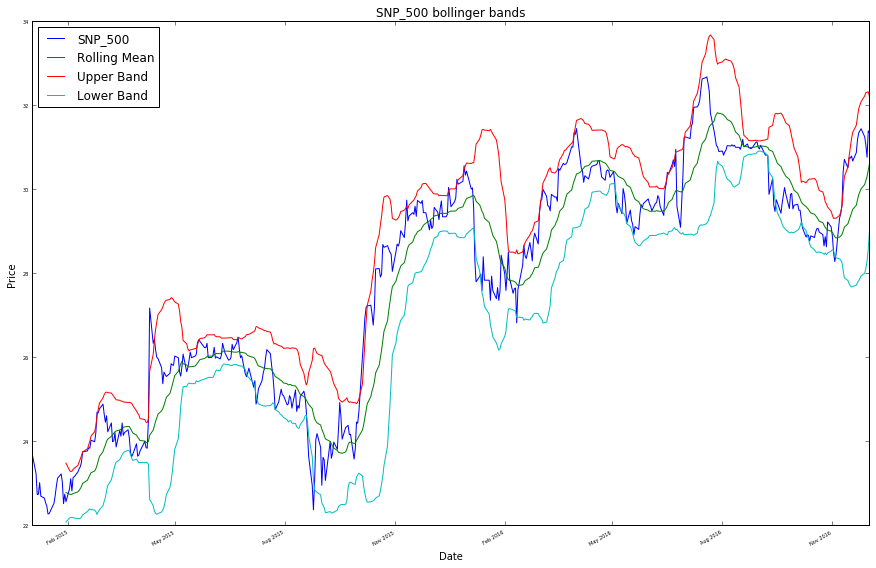
Please find below the Normalized plot for all stocks for “Adj Close” along with SNP\_500”:



Please find below the Normalized plot GE vs SNP\_500:

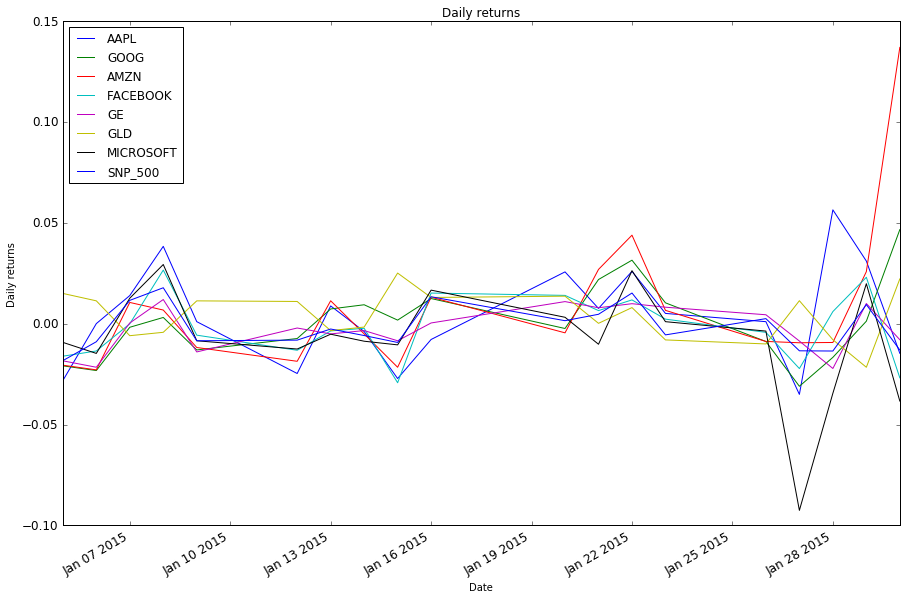


Bollinger Bands for GE: The popular notion is that when the stock price meets the lower Bollinger band (mean – 2\*Standard deviation), then it’s usually a buy signal & when the stock price meets the upper Bollinger band (mean + 2\*Standard deviation) its usually a sell signal:



Daily returns:

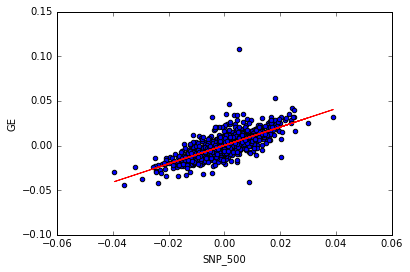
* Comparing the Daily returns of all stocks for the month of Jan 2015:



* Comparing daily returns between GE and SNP\_500 for the month of Jan 2015, its seems GE frequently moves in the same direction as SNP\_500 mostly, but in few instances, it also moves farther from SNP\_500:



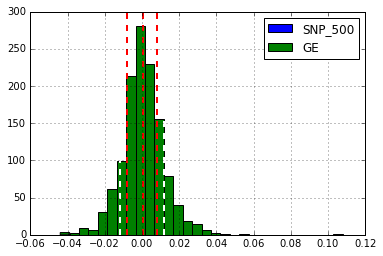
* Scatter plot between GE and SNP\_500 (between dates '2011-01-01', '2016-12-01') and fitting a line & looking at the statistics below. As we can see the slope or Beta is almost 1, this infers GE is very reactive to the market meaning if the market goes up by 1%, GE also goes up by 1 percent. Alpha is positive, which means GE performs little bit better than SNP 500 everyday on an average:



beta\_GE = 1.0291719923

alpha\_GE = 0.000233231271241

* Daily return distribution of GE compared to SNP\_500 represented by histogram: The ‘white’ colored dotted line identifies GE’s mean and std, 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 GE has a higher std and higher Kurtosis which infers GE’ distribution is not exactly Gaussian with flat tails.



Mean, Standard Deviation and Kurtosis for SNP\_500

Mean = SNP\_500 0.000476

std = SNP\_500 0.008204

kurtosis = SNP\_500 1.891211

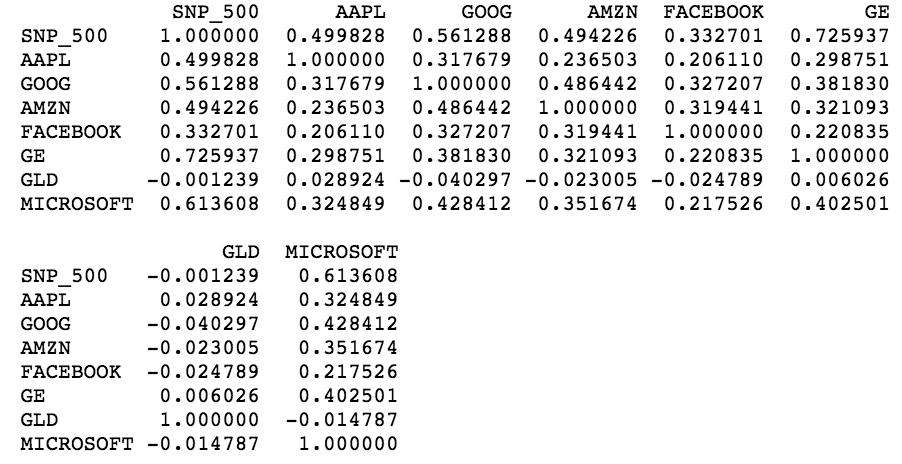
Mean, Standard Deviation and Kurtosis for GE

Mean = GE 0.000723

std = GE 0.011632

kurtosis = GE 6.805789

* Pearson co-relation among Daily returns of the stocks, please refer below. We can see that GE is the only stock which has the best co-relation with SNP\_500 as we would infer from explanation above:



* Cumulative returns for all the stocks for the period between '2015-01-01' and '2016-12-01:

commulative return for SNP\_500 is:

6.46%

commulative return for AAPL is:

4.12%

commulative return for GOOG is:

42.51%

commulative return for AMZN is:

141.04%

commulative return for FACEBOOK is:

46.72%

commulative return for GE is:

32.64%

commulative return for GLD is:

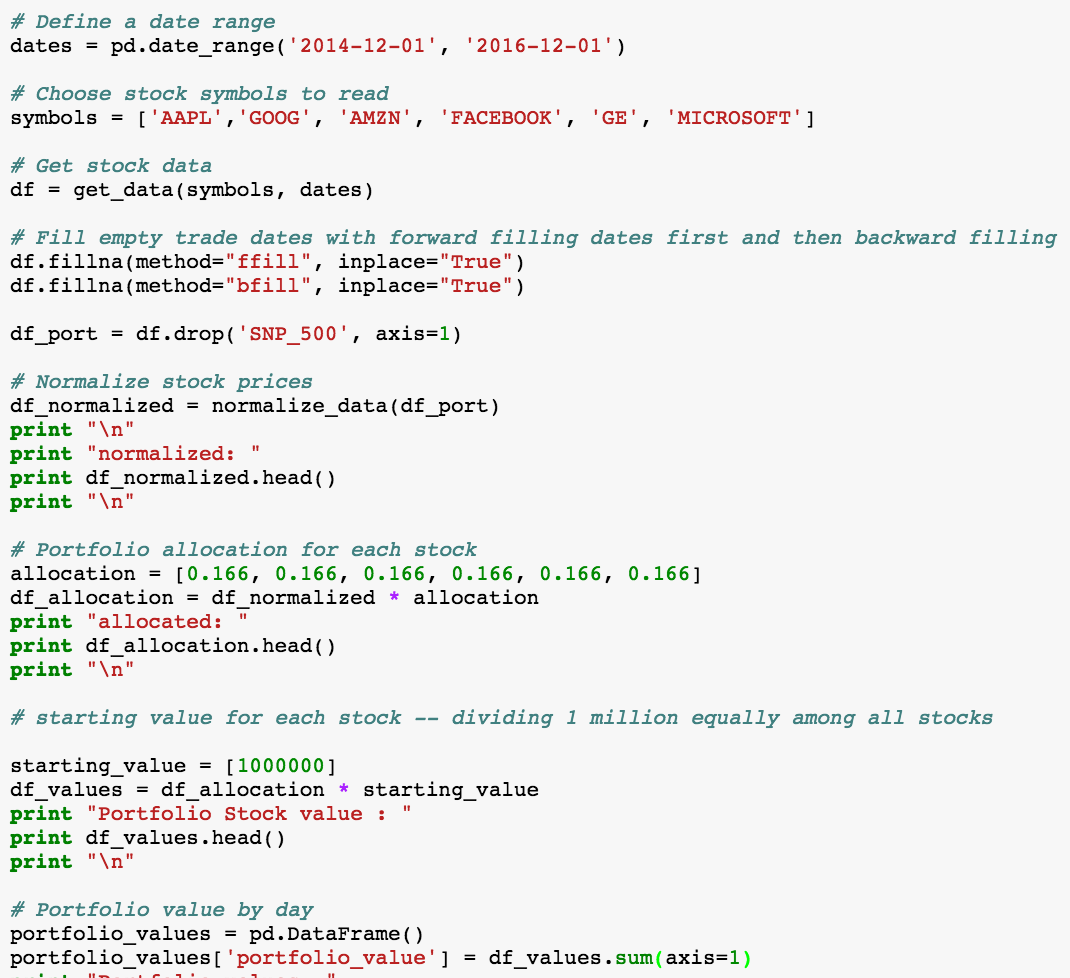
-2.23%

commulative return for MICROSOFT is:

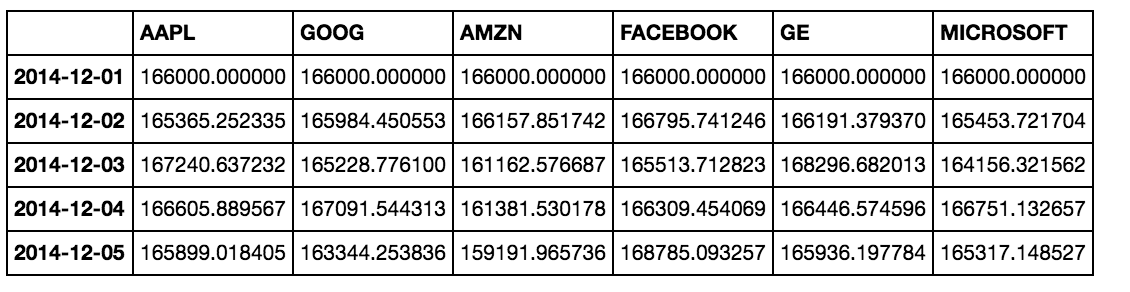
33.62%

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

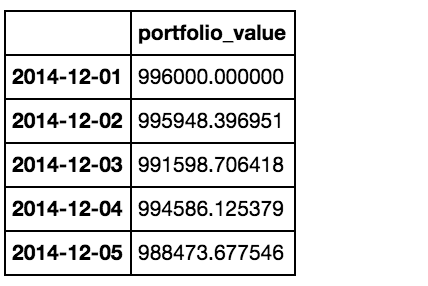
* Code snippet with equal allocation to a portfolio of Six stocks:

****

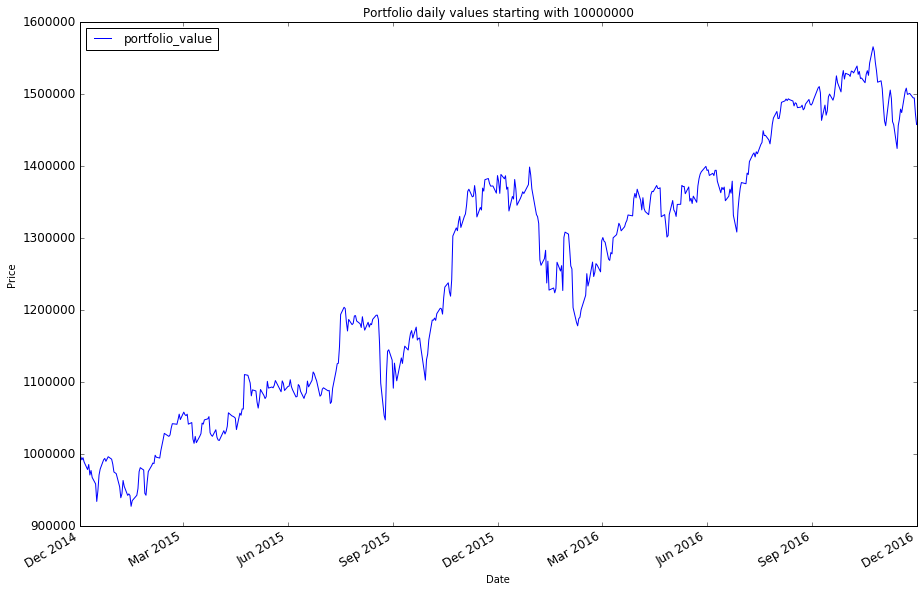
* Sample: Value of each Stock in the portfolio:

****

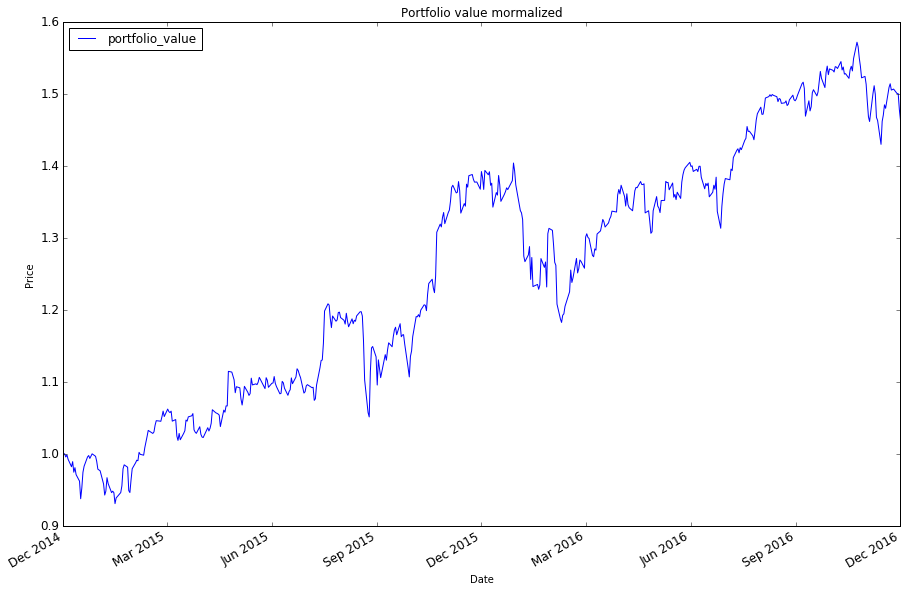
* Portfolio Value – summing up all the stock prices

****

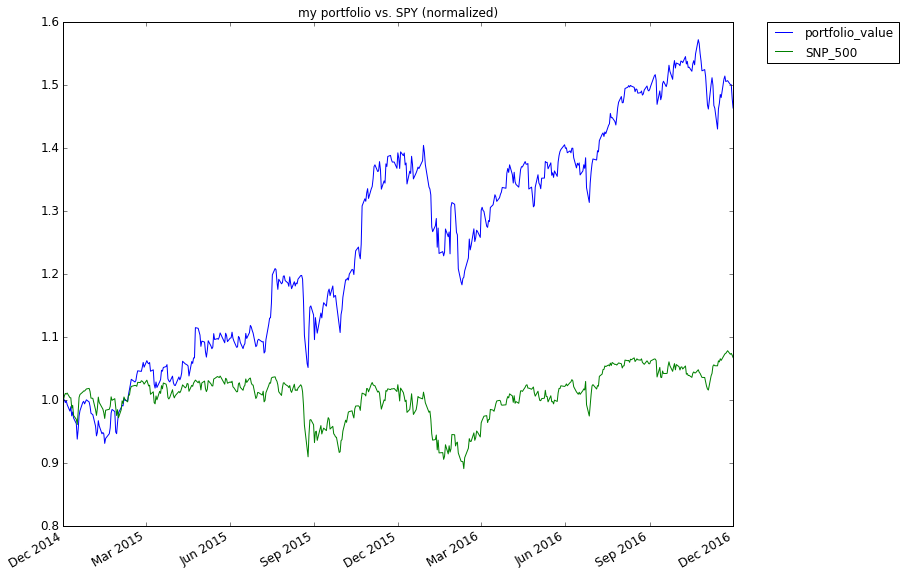
* Following is the plotted portfolio statistics:

****

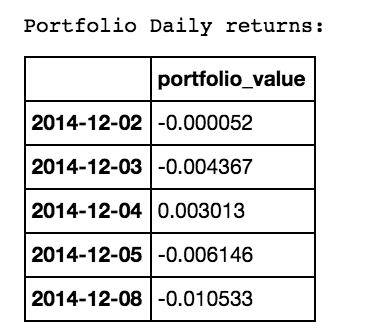
* Following is the normalized portfolio statistics:

****

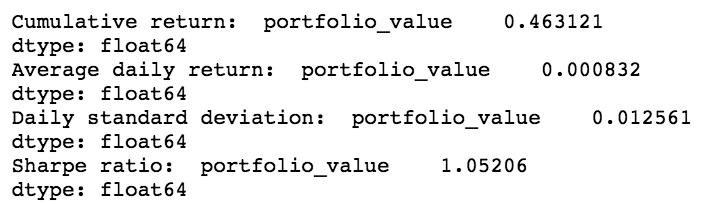
* Following is the portfolio value normalized plot vs normalized SNP\_500

****

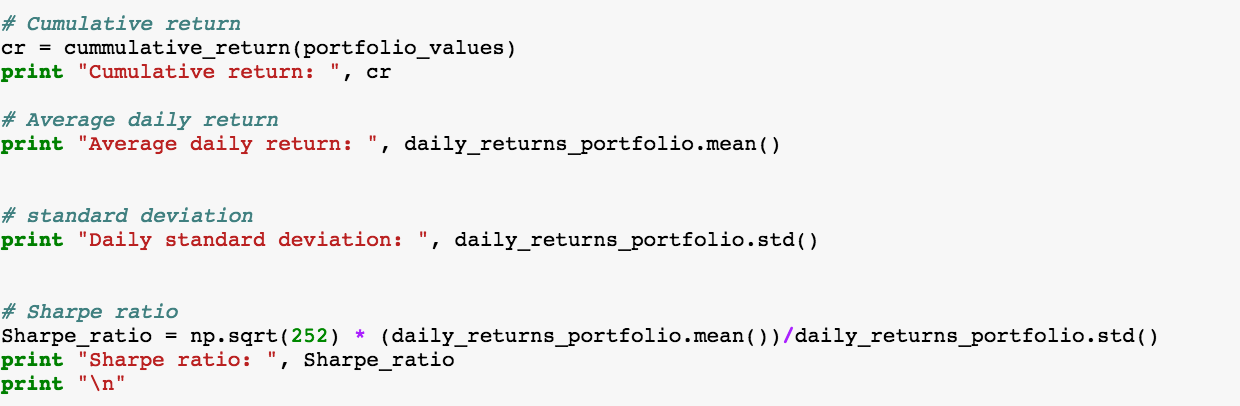
* Sample Data of Daily returns on the Portfolio

****

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

****

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

****

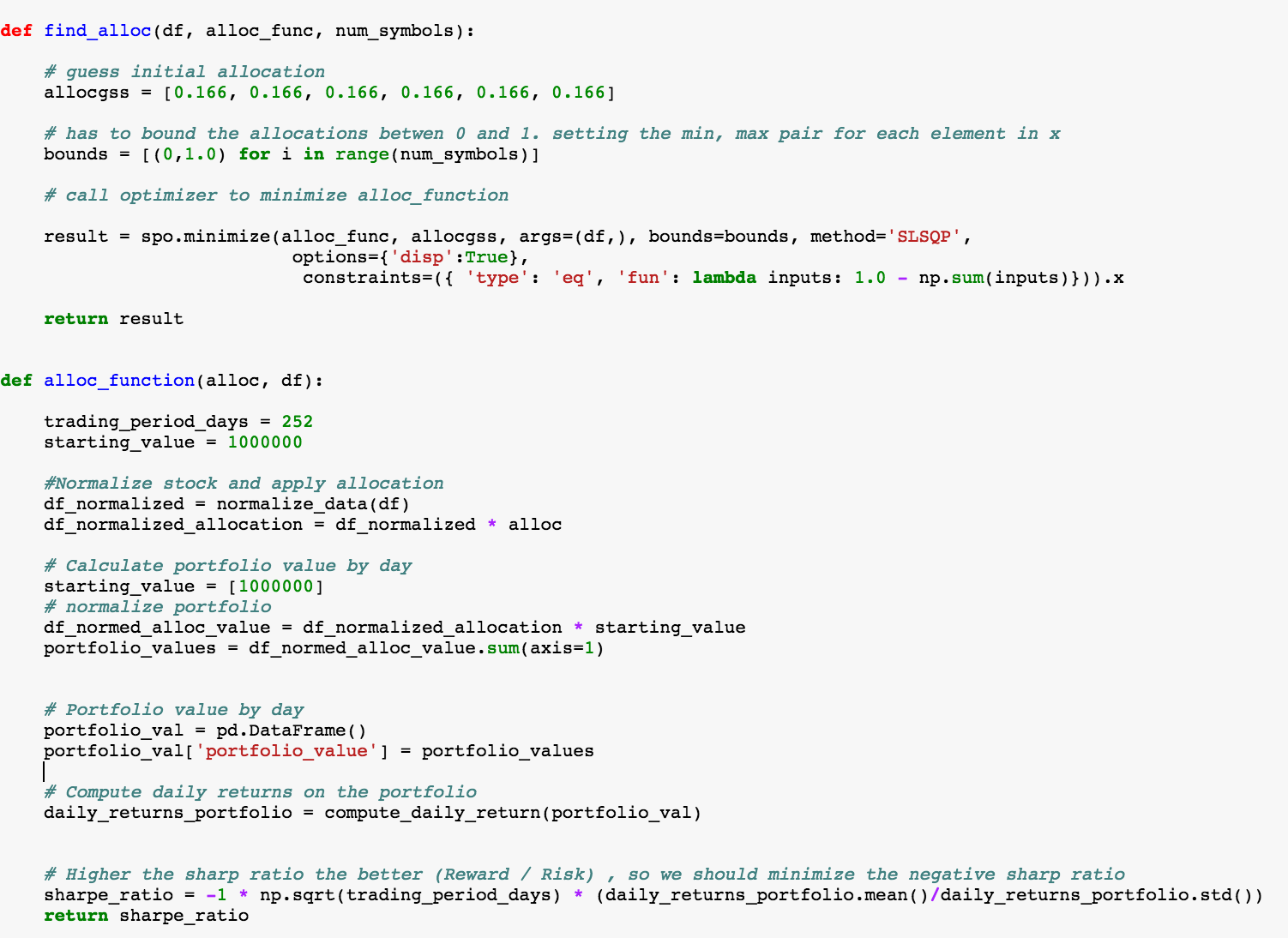
**HEADER 3:**

* **Optimizing the Portfolio by optimizing or minimizing the negative sharpe ratio**
* **plotting optimal portfolio statistics**
* **Scatter plot, co-relation between my portfolio daily return and SNP\_500 daily return, Beta value to see how reactive is my optimized portfolio compared to the market, alpha value to see how well it performs compared to the market**
* **histogram**

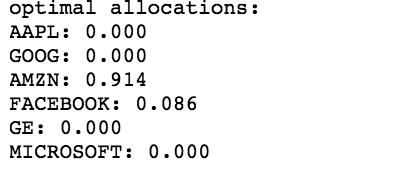
Sharpe Ratio is a measure for calculating risk-adjusted return, and this ratio has become the industry standard for such calculations

Source: <http://www.investopedia.com/terms/s/sharperatio.asp>

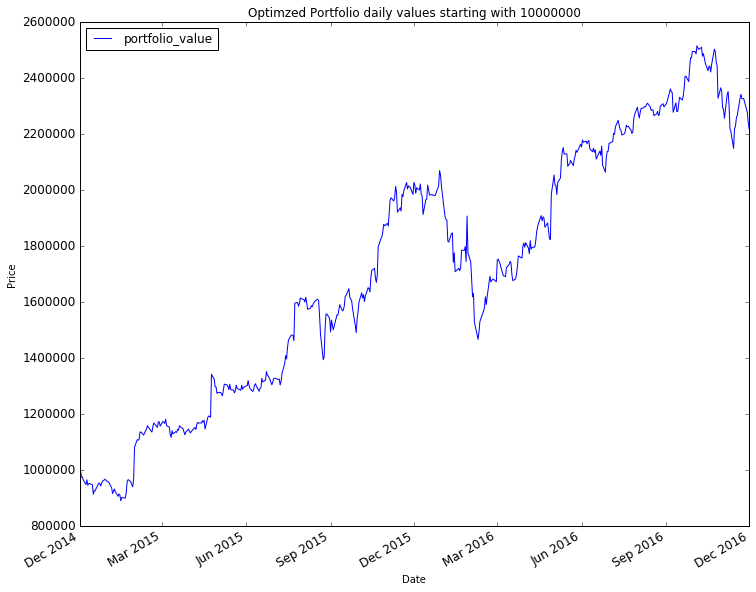
Code Snippet below to optimize sharpe ratio:

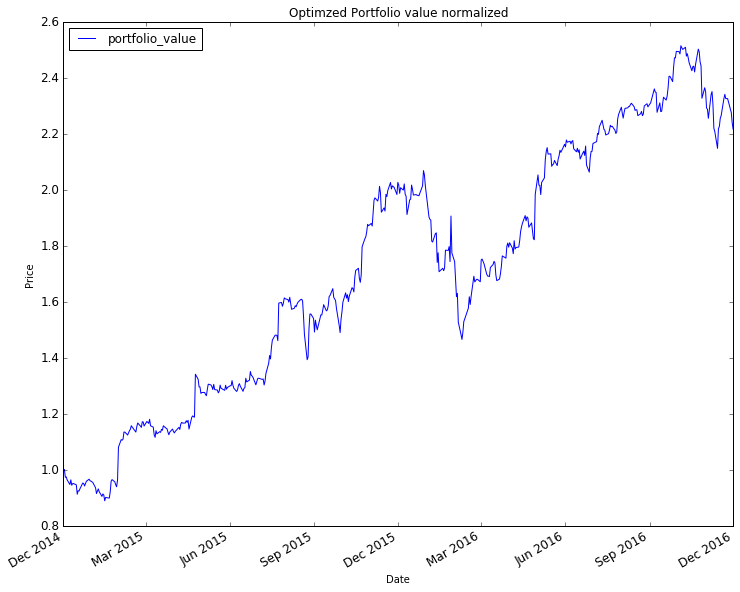


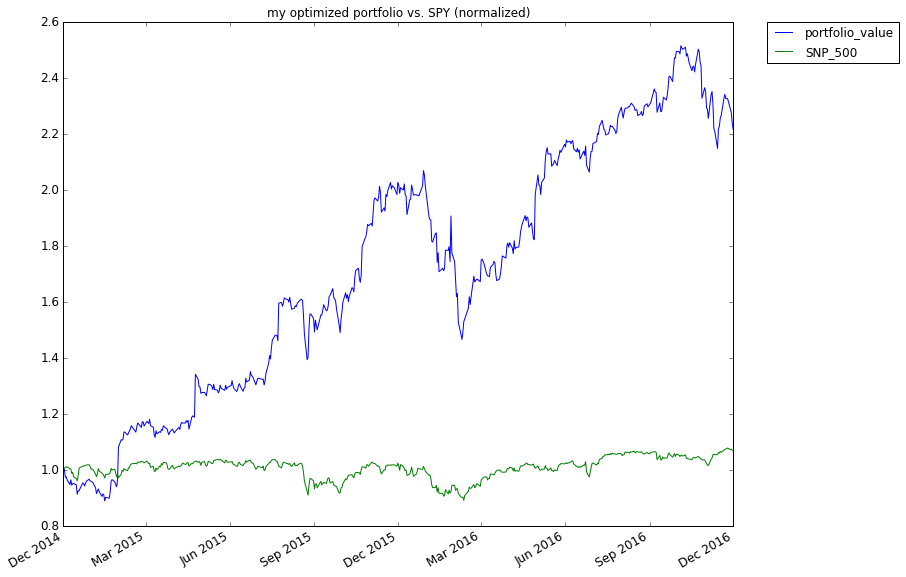
Optimal allocation below after Sharpe Ratio optimization:

****

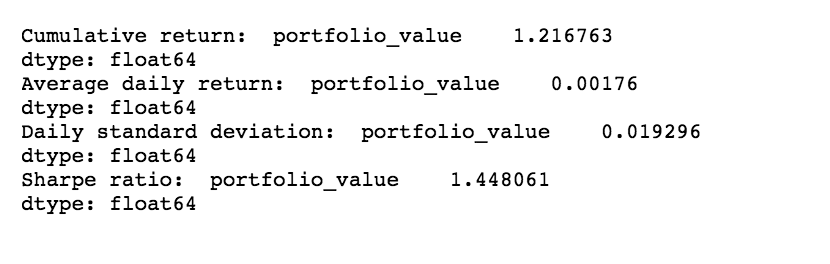
Plotted Optimal Portfolio Statistics:

****

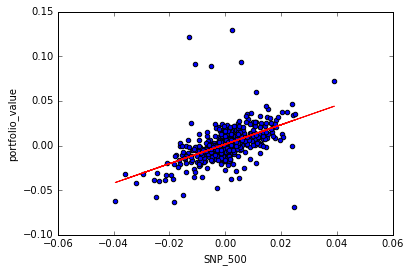
****

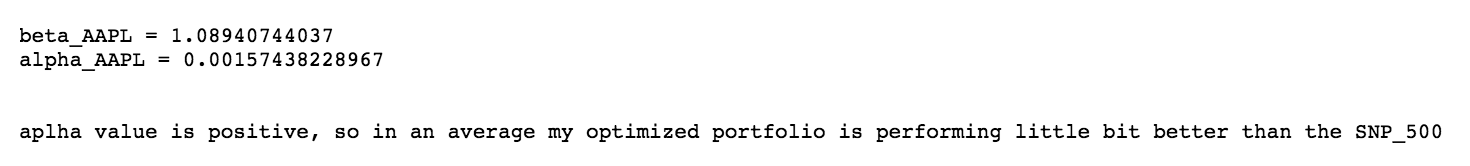
****

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

****

Scatter plot between my optimized portfolio and 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](https://en.wikipedia.org/wiki/Statistics), linear regression is an approach for modeling the relationship between a scalar [dependent variable](https://en.wikipedia.org/wiki/Dependent_variable) y and one or more [explanatory variables](https://en.wikipedia.org/wiki/Explanatory_variable) (or independent variables) denoted X. The case of one explanatory variable is called [simple linear regression](https://en.wikipedia.org/wiki/Simple_linear_regression). For more than one explanatory variable, the process is called multiple linear regression.

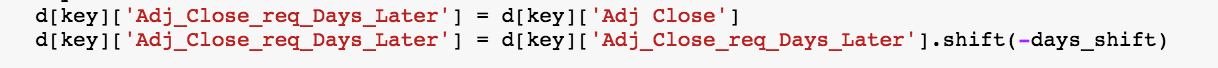
Source: <https://en.wikipedia.org/wiki/Linear_regression>

Multiple Linear Regression represented in the form: **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:



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.

**KNN Regression:**

The k-Nearest Neighbors algorithm (or k-NN for short) is a [non-parametric](https://en.wikipedia.org/wiki/Non-parametric_statistics) method used for [classification](https://en.wikipedia.org/wiki/Statistical_classification) and [regression](https://en.wikipedia.org/wiki/Regression_analysis). In both cases, the input consists of the k closest training examples in the [feature space](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/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.

\*\*Another reason I tried these two models as they were well discussed in the “Machine Learning for Trading” Course

**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.

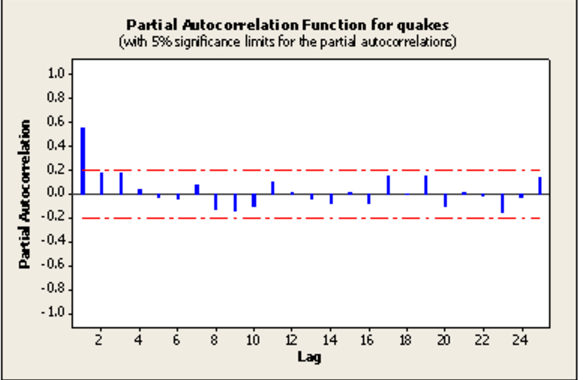
Source: Udacity “**Time Series Forecasting**” Course

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.

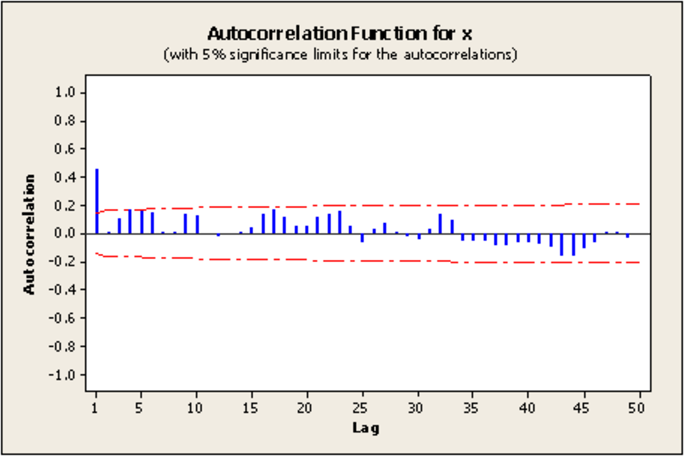
AR model: **Identification of an AR model is often best done with the PACF.** the theoretical PACF “shuts off” past the order of the model.  The phrase “shuts off” means that in theory the partial autocorrelations are equal to 0 beyond that point

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.

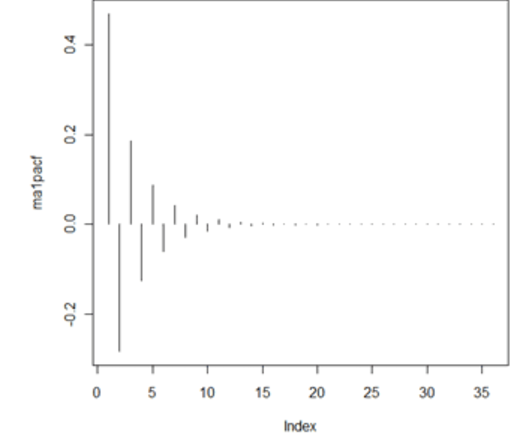
Sample PACF for an AR model where the PACF shuts off past the 1st order so, AR =1 in this case



Sample ACF where MA =1



Theoretical PACF where MA =1



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.

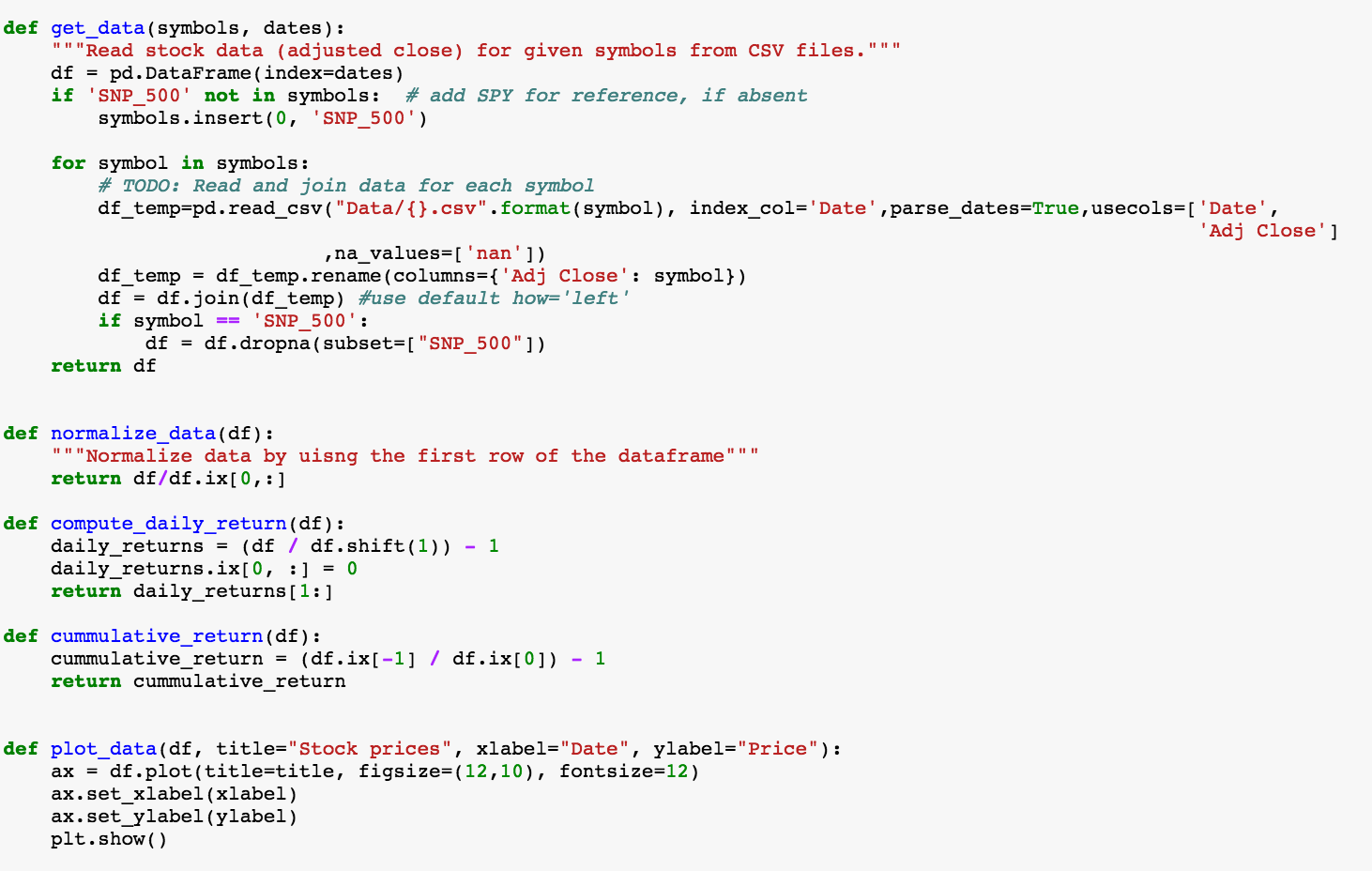
**3.Methodology:**

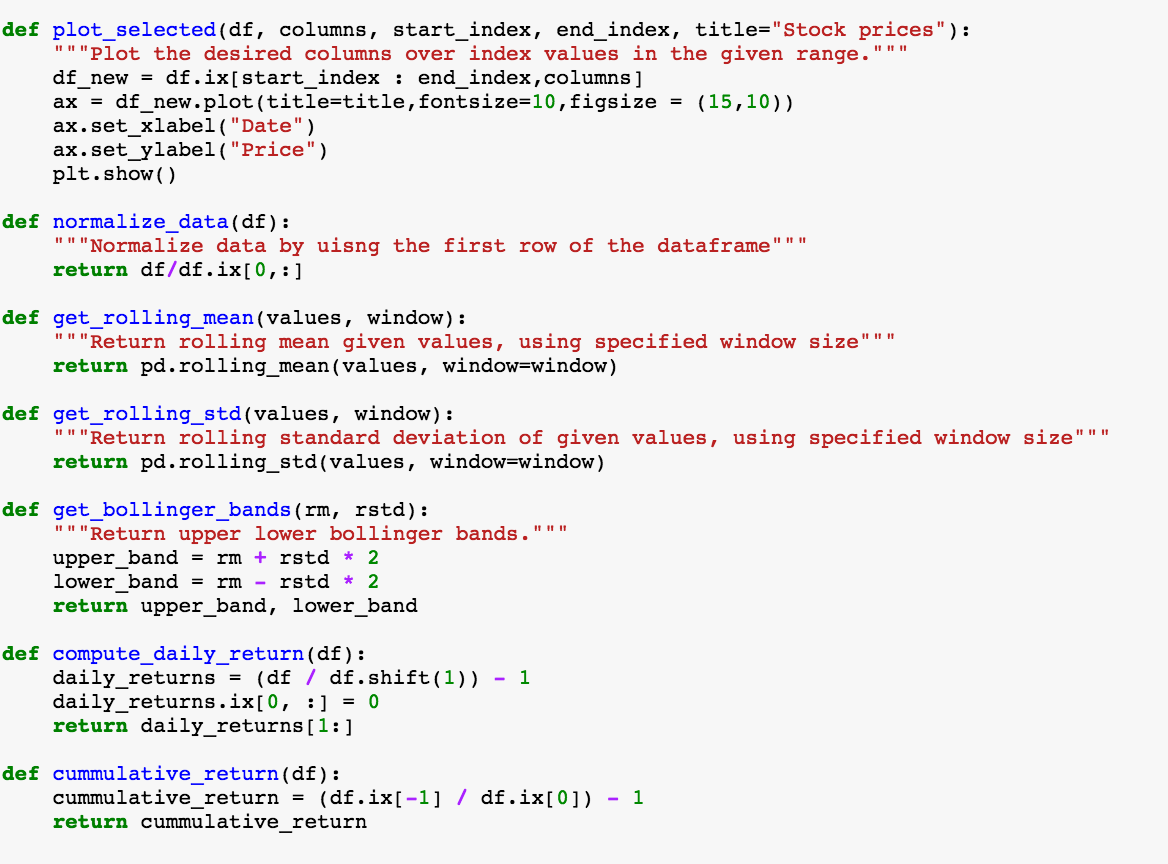
### **Data Preprocessing: (Please refer the All\_Analysis.ipyn ipython notebook “HEARDER 2”)**

I have chosen these stocks for training and prediction: 'AAPL','GOOG', 'AMZN', 'FACEBOOK', 'GE', 'MICROSOFT' as well as ‘SNP\_500’ index and ‘GLD’ 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:

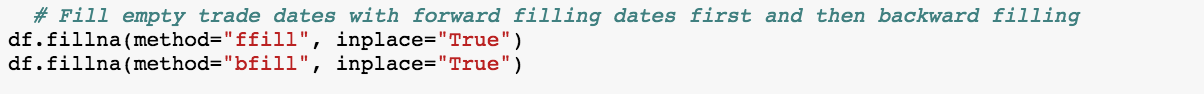
**\*\*\*** I have taken extensive help from the udacity free course “Machine Learning for Trading”

****

****

**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):



**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’.

Please refer the ipython notebook book: **All\_Analysis.ipynb** – **HEADER 2** and **HEADER 3**

**Implementation:**

Please refer the following headers in the ipython notebook book “**All\_Analysis.ipynb”** :

**# HEADER 4**

**- STOCK PRICE PREDICTION INTERFACE --- LINEAR REGRESSION**

**# HEADER 5**

**- STOCK PRICE PREDICTION INTERFACE --- KNN REGRESSION**

**#** **HEADER 6**

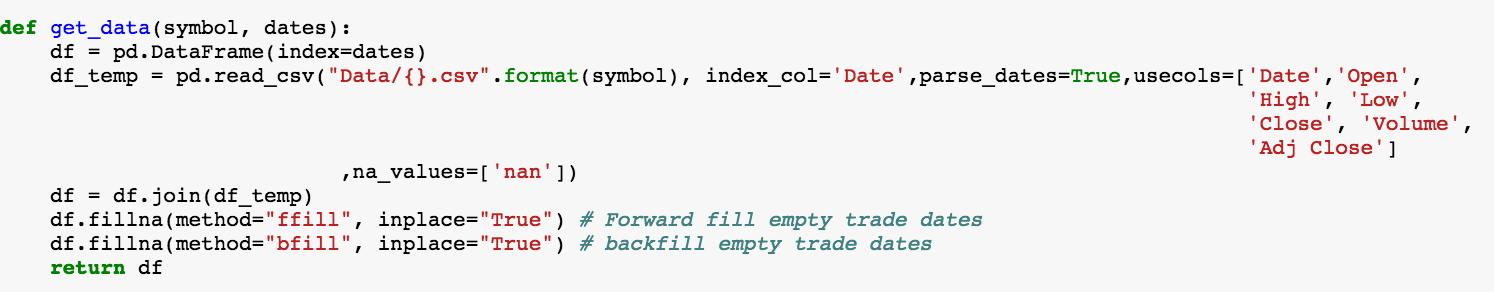
**- STOCK PRICE PREDICTION INTERFACE: ARIMA**

**LINEAR REGRESSION:**

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

classsklearn.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:](https://github.com/scikit-learn/scikit-learn/blob/14031f6/sklearn/linear_model/base.py" \l "L417)

[](https://github.com/scikit-learn/scikit-learn/blob/14031f6/sklearn/linear_model/base.py" \l "L417)

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”:**



**Training start date**

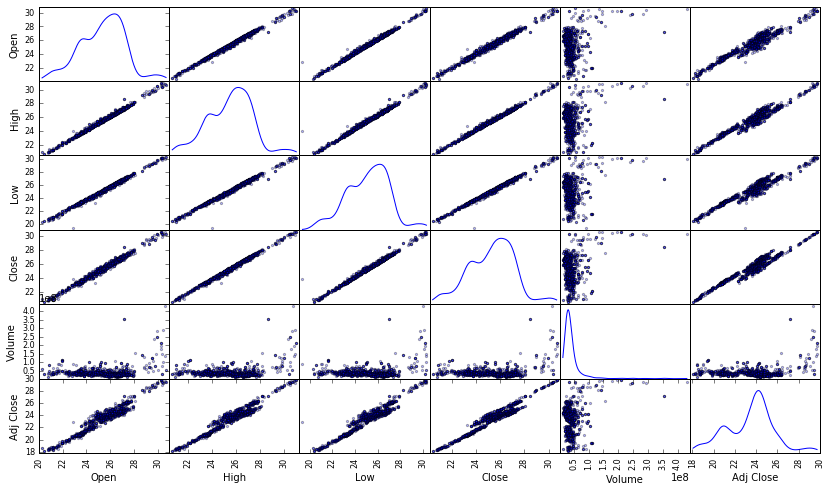
**Training end date**

**The number of days between the “Training end date” and the date in future on which we want to predict the stock prices**

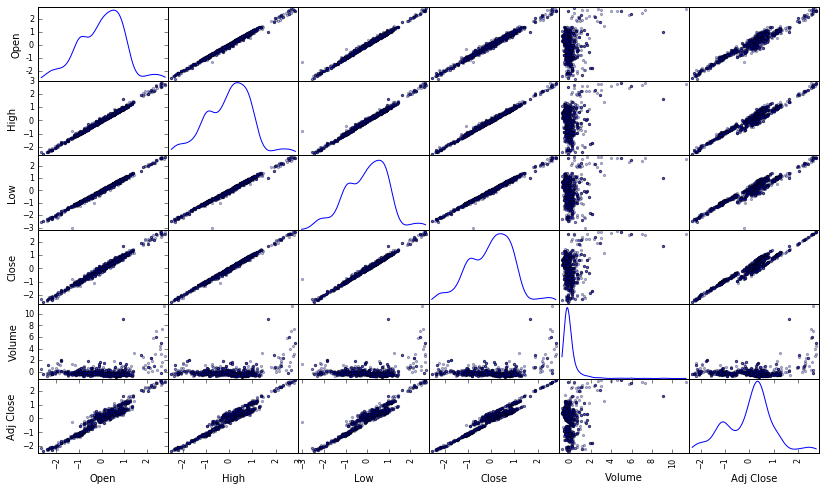
**The list of stock for which we want to build the trained models**

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.

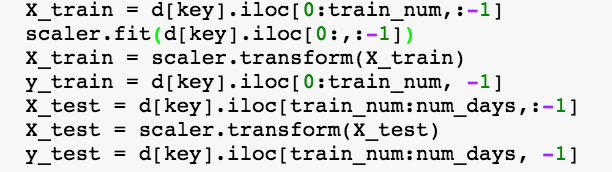
Below is the distribution of GE’s attributes before performing standardization:

****

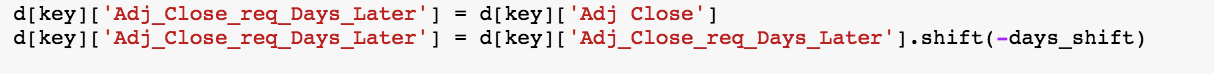
Below is the distribution of GE’s attributes after performing Standardization:

****

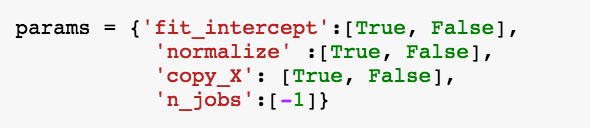
I am doing a manual test train on the data as it is advised in the course – “Machine Learning for trading” that we need to the split in the sequential in the stock price prediction world. The Sklearn’s test train split module performs randomized test train split, we must not get into a situation where we are testing on datasets which is prior to trained datasets. Please refer the below the code snippet to perform the manual test train split:



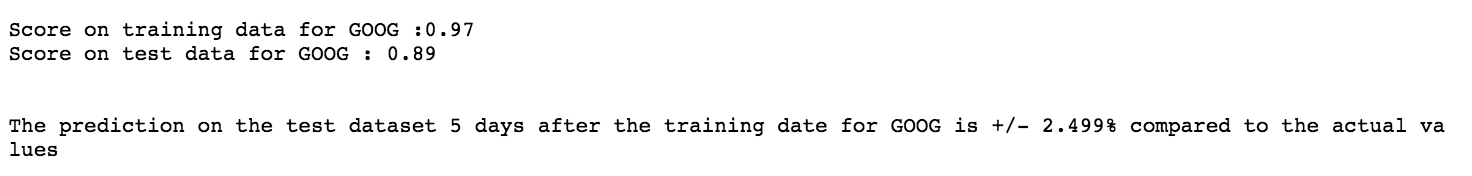
Just re-iterating, please also note that I am engineering the feature “Adj\_Close\_Req\_Days\_Later”. Here “days\_shift” is the number of days between the last training date and the date on which I want to predict the stock price in future :



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:

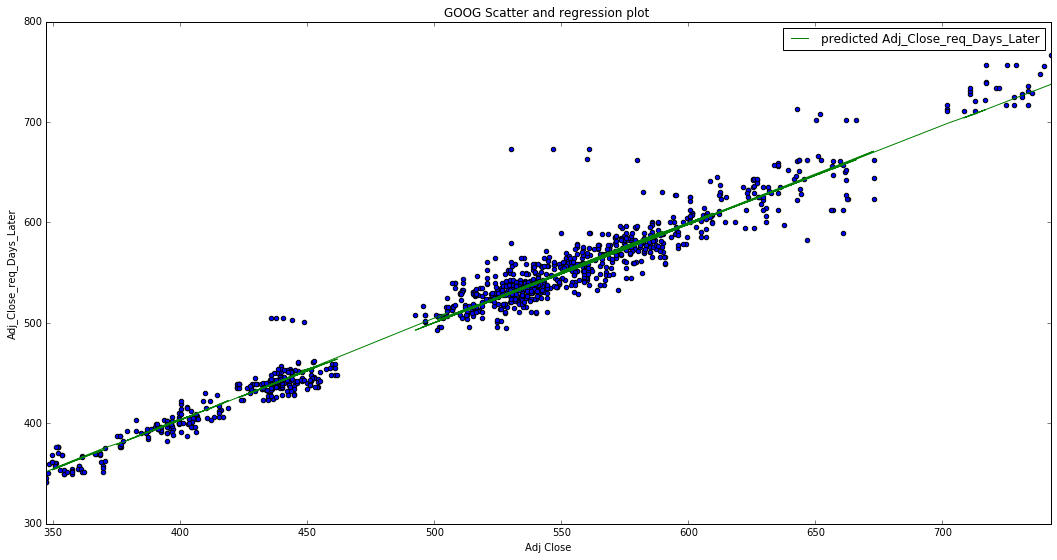


I printing out the training and test scores on the stocks, and the % +/- variation of the predicted stock value compared to the actual values. Additionally, I am also plotting the scatter plot between ‘Adj Close’ and ‘Adj\_Close\_req\_Days\_laters’ and the regression fitted line on the predicted ‘Adj\_Close\_req\_Days\_laters’.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

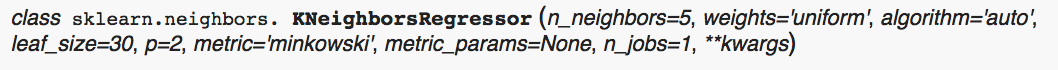


Scatter plot between ‘Adj Close’ and ‘Adj\_Close\_req\_Days\_laters’ and the regression fitted line on the predicted

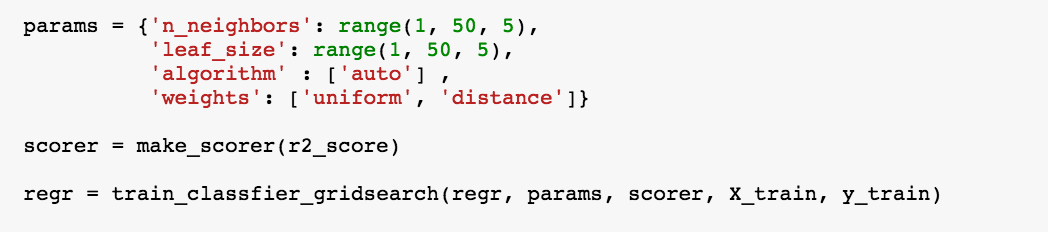
‘Adj\_Close\_req\_Days\_laters’ for Google



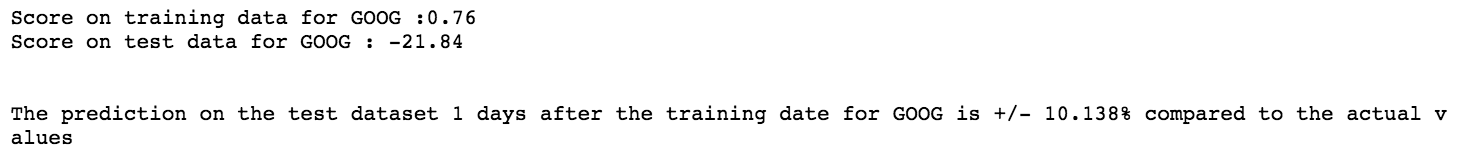
**KNN REGRESSION:**

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

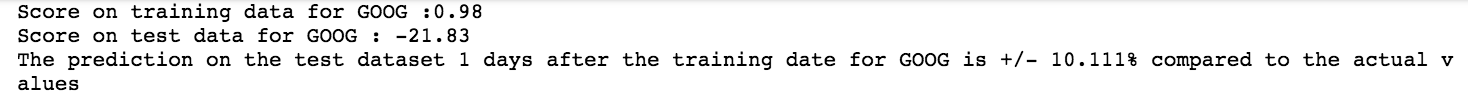
Additionally, since the R2 scores on test data for most of the stocks using the default parameters are terrible, I thought of using gridsearch to find the best tune parameters for KNN Regressor, please refer below the code snippet for hyper parameter tuning:



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



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:



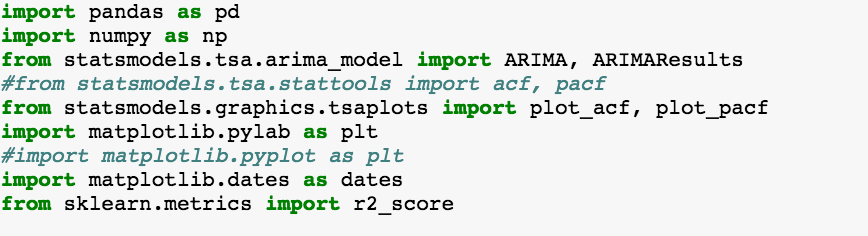
**ARIMA:** Please refer the “Algorithms and techniques” section of this document for an explanation of ARIMA**:**

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 “All\_Analysis.ipynb” notebook:

**# HEADER 6**

**- STOCK PRICE PREDICTION INTERFACE: ARIMA**”

Please note the libraries imported:

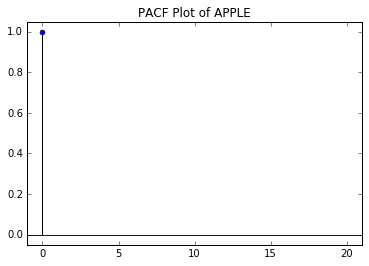


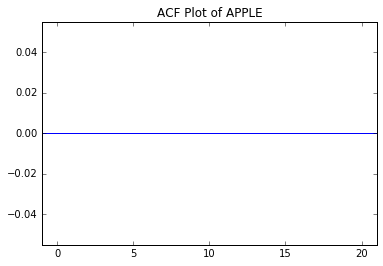
Please note for ARIMA I am just using ‘ADJ CLOSE’ as the attribute and since I am trying to predict the stock price one ahead I am adjusting the ‘ADJ CLOSE’ as below:



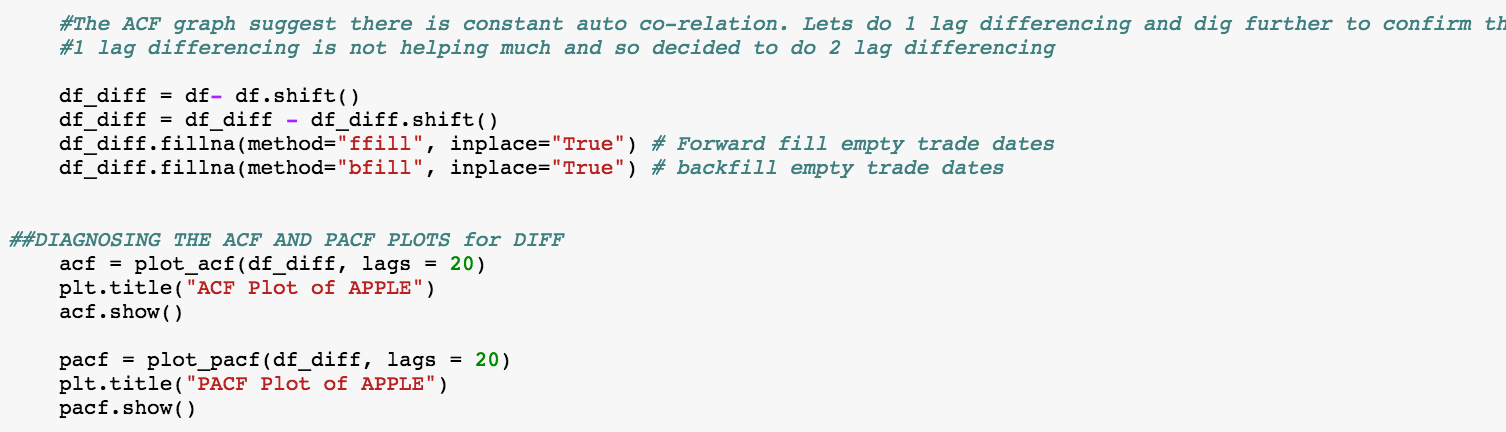
I started with diagnosing the AR, I and MA terms for the ARIMA model, please refer below the code snippet as well the ACF and PACF plots:

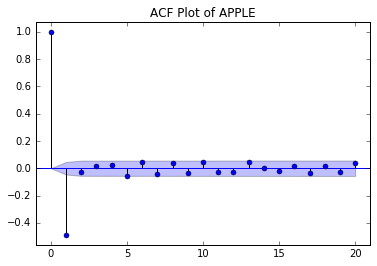


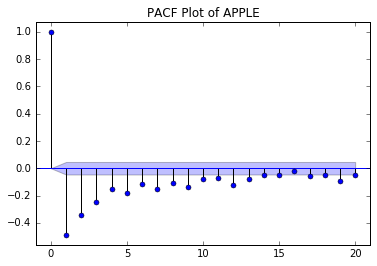




The ACF plot above suggests there is no auto-correlation, so I decided to do 1 lag differencing which is also, not helping much and went ahead with 2 lag differencing. Please refer the code snippet as well the ACF and PACF plots below:







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:



|  |
| --- |
|  |

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

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>

**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 :0.97

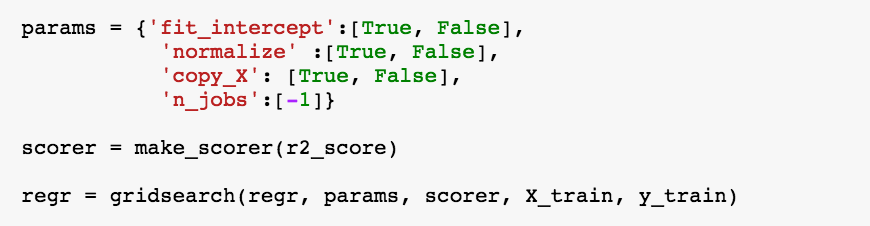
Score on test data for GOOG: 0.89

Score with grid search:

Score on training data for GOOG :0.97

Score on test data for GOOG: 0.89

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



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

**4.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 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 or less 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.

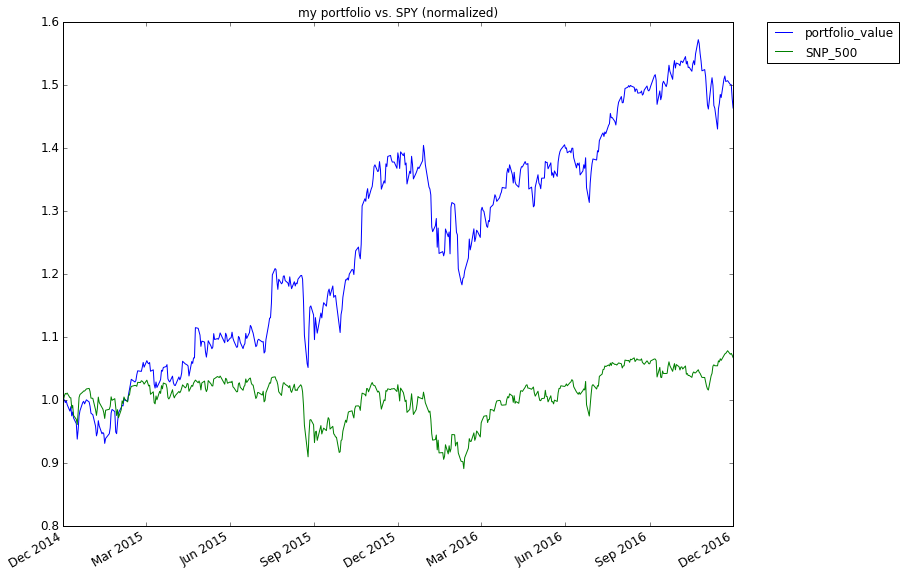
Non-Seasonal ARIMA showed some promising results, for Apple Stock its R2 score on the train dataset **1 day** out in future is **0.88** but the score on the unseen test dataset is **0.47** and missed the benchmark of 0.6 on test dataset. I could have tried out the Season ARIMA to bring in seasonality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

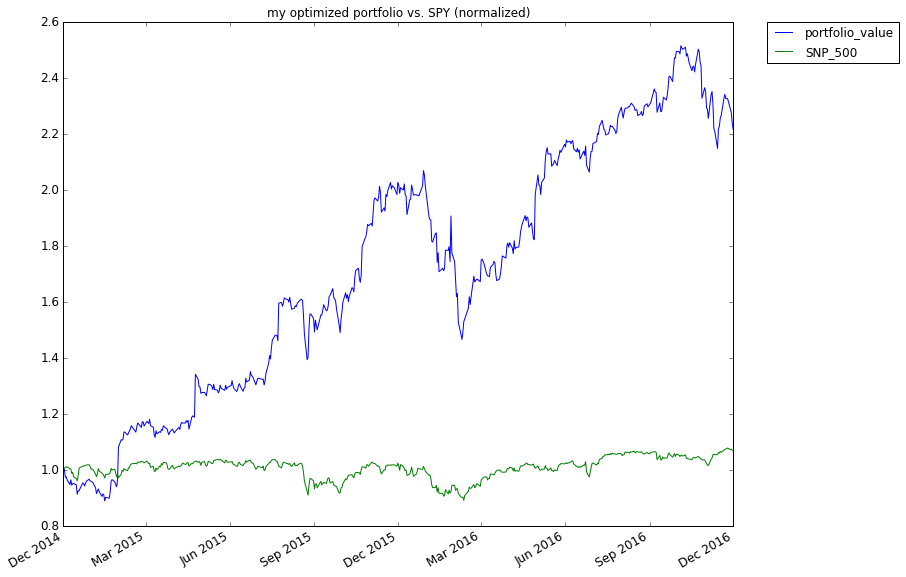
**5. Conclusion: Free form visualization, Reflection and Improvement:**

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 = ['AAPL', 'GOOG', 'AMZN', 'FACEBOOK', 'GE', 'MICROSOFT'], allocation = [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 = ['AAPL', 'GOOG', 'AMZN', 'FACEBOOK', 'GE', 'MICROSOFT'], allocation = [0.0, 0.0, 0.914, 0.086, 0.0, 0.0]

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

****

****

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. I believe a real time self-learning robotic system which take inputs from its environment is more relevant in the world of stock price prediction than an algorithm which only learns from history and predicts the future. I am starting with the “**Self- Driving Car**” Nano degree beginning 16th of Feb, I am confident that I would gain good expertise in **Deep learning** and **Robotics** after completing the Nanodegree, which I can definitely apply in the stock price prediction problem as well.