Stock Market Prediction

CAPSTONE PROJECT

PREPARED BY

Kavya Kushnoor

Udacity - Machine Learning Nanodegree

Project Domain Background:

The main area of focus in this project is stock market trading data using publicly traded stocks. Stock market data is well documented and available using several resources like Google finance, Quandl, yahoo finance, etc. The major challenge is to clean this data and make it uniform for usage in analysis and prediction across the varied platforms. This area of focus is valued for the great business potential that it offers.

One of the major reference and foundation for this project has been the Machine Learning for Trading course offered at Udacity. The other references used for the research are mentioned below:

Related Academic Research:

- Machine Learning for Trading- Udacity Course <https://www.udacity.com/course/machine-learning-for-trading--ud501>
- Vatsal H. Shah. (2007). “Machine Learning Techniques for Stock Prediction” <http://artent.net/2012/08/30/machine-learning-techniques-for-stock-prediction/>
- Gurav, Uma & Sidnal, Nandini. (2018). Predict Stock Market Behavior: Role of Machine Learning Algorithms. 10.1007/978-981-10-7245-1_38. <https://www.researchgate.net/publication/322611175_Predict_Stock_Market_Behavior_Role_of_Machine_Learning_Algorithms>
- Hegazy, Osman & Soliman, Omar S. & Abdul Salam, Mustafa. (2013). A Machine Learning Model for Stock Market Prediction. International Journal of Computer Science and Telecommunications. 4. 17-23. <https://www.researchgate.net/publication/259240183_A_Machine_Learning_Model_for_Stock_Market_Prediction>

Problem Statement:

There are several challenges in analyzing stocks data; viz:

- Discrepancy and accuracy across varied online resources
- Metrics and statistical methods to use for portfolio valuation
- Missing values in the historic datasets
- Since various companies trade at varying rates, normalizing data is essential for a fair comparison

In order to address these discrepancies, pandas dataframes have been used to clean and tabulate data for analysis. The steps taken to solve this issue have the script in the attached Jupyter Notebook. The steps are:

Step 1: Normalize stock prices in the portfolio by dividing each value with the first one.
Step 2: Assign the allocated amount of investment by multiplying it with each of the normalized values.
Step 3: Calculate possible values by multiplying allocated amount with the starting value
Step 4: Get the sum of all the values in each stock to find the daily return value of the portfolio.
Step 5: Use python machine learning libraries and classes to run the analysis

Three types of models are used for making the prediction using machine learning, viz:

1. Random Forest Regressor
2. Decision Tree Model
3. Gradient Boosting Regressor

Evaluation Metrics:

The major evaluation metric used is Root mean square error. Besides that, we will evaluate the accuracy of the model using three commonly used methods:

1. Mean Absolute Error : It is the mean of the absolute value of the errors.

$$\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|$$

2. Mean Squared Error: It is the sum of the mean of the squared errors.

$$\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|^2$$

3. Root Mean Squared Error: It is the square-root of the sum of the mean of the squared errors.

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|^2}$$

We find these values by executing this script:

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 496.2985370004204
Mean Squared Error: 435980.02522676234
Root Mean Squared Error: 660.2878351346194
```

Data Exploration:

Quandl is the major source of stocks data for the portfolio. However, market data like S&P 500 is taken from investing.com as it required premium membership access in Quandl. The cleaned data files can be found in the Github repository.

For market data, visit: Market data
For portfolio data, visit: Portfolio data

The calculation script and explanations regarding the input, dimensionality, normalization and pre-processing of data can be found in the attached Jupyter Notebook. The data collected is over a period of 3.5 years, viz; July 1, 2015 to Dec 31, 2018.

Create a Dataframe with X & Y variables:

Column 1: Portfolio Values

We will now create a new dataframe that will contain all the X and Y variables that will be used in out predictive models. The cell below will add the dependent variable column 'Portfolio Val' which values are to be predicted.

Column 2: S&P 500 Values

Next, we will read the S&P500 values from the attached csv file 'S&P500.csv' and add it as our second column.

Column 3: Rolling mean of portfolio values

We will use a 5-day rolling mean for the values of 5 consecutive days on a rolling basis.

$$RollingMean = \frac{PortfolioValue(t)}{Mean(n:t)}$$

Column 4: Rolling standard deviation of portfolio values

We will use a 5-day rolling standard deviation for the values of 5 consecutive days on a rolling basis.

$$RollingStandardDeviation = \frac{PortfolioValue(t)}{StandardDeviation(n:t)}$$

Column 5: Upper Bollinger band

We will use the upper bollinger band which is the sum of the 5-day rolling mean and 2x of the 5-day rolling standard deviation.

$$UpperBand = RollingMean + 2 * RollingStandardDeviation$$

Column 6: Lower Bollinger band

We will use the lower bollinger band which is the difference of the 5-day rolling mean and 2x of the 5-day rolling standard deviation.

$$LowerBand = RollingMean - 2 * RollingStandardDeviation$$

Column 7: Bollinger bands

We will get bollinger bands which is 2x of the 5-day rolling standard deviation.

$$BollingerBands = \frac{PortfolioValue(t) - RollingAverage(n:t)}{2 * RollingStandardDeviation(t)}$$

	Portfolio Val	SP500	Rolling_mean_PV	Rolling_std_PV	upper_band_PV	lower_band_PV	BB
0	10000.000000	2077.42	13369.735506	104.566367	13578.868239	13160.602773	0.107968
1	10055.424566	2076.78	13369.735506	104.566367	13578.868239	13160.602773	0.107968
2	10029.035764	2068.76	13369.735506	104.566367	13578.868239	13160.602773	0.107968
3	10028.484013	2081.34	13369.735506	104.566367	13578.868239	13160.602773	0.107968
4	9863.244645	2046.68	9995.237797	76.346138	10147.930074	9842.545520	-0.864439
...	...	...	...	...	...	...	...
624	17983.258276	2679.25	18034.986713	109.541511	18254.069735	17815.903691	-0.236113
625	17953.896092	2684.57	17983.406597	50.021882	18083.450361	17883.362832	-0.294976
626	17990.031618	2683.34	17967.594727	19.109936	18005.814600	17929.374854	0.587048
627	17974.392222	2680.50	17973.523356	14.224080	18001.971516	17945.075196	0.030542
628	17856.439380	2682.62	17951.603518	54.907961	18061.419440	17841.787595	-0.866579

A Summary of the Dataframe of Variables¶

	Portfolio Val	SP500	Rolling_mean_PV	Rolling_std_PV	upper_band_PV	lower_band_PV	BB
count	629.000000	629.000000	625.000000	625.000000	625.000000	625.000000	625.000000
mean	13373.595923	2223.049141	13369.735506	104.566367	13578.868239	13160.602773	0.107968
std	2352.693243	207.102255	2335.808491	65.500885	2324.243397	2354.616353	0.503634
min	9500.286457	1829.080000	9839.985634	5.041378	10144.953111	9048.230215	-0.887889
25%	11281.620234	2071.180000	11271.397308	61.324932	11466.753995	11091.556441	-0.302416
50%	13302.268737	2168.270000	13303.927717	90.126445	13519.850888	13092.697880	0.202620
75%	15365.576956	2396.920000	15363.311964	135.399861	15604.631598	15041.415126	0.546905
max	18309.025416	2690.160000	18207.281931	528.000548	18504.266930	18027.006482	0.894274

Algorithm & Technique:

- Prepare and Split the Data
- Implement Decision Tree Model,
- Implement the GradientBoosting Regressor Model
- Implement the RandomForestREgressorModel

For each of the models above, we will implement the following steps:

- Fit the model and find score for train & test data
- We will guage the importance of each of the x variables using feature Importance of Independent Variables
- We will compare the predicted values y_pred with the actual output values y_test. Plot them in a table and visualize the comparison
- Evaluate the accuracy of the model using three commonly used methods:
    - Mean Absolute Error : It is the mean of the absolute value of the errors.
    - Mean Squared Error: It is the sum of the mean of the squared errors.
    - Root Mean Squared Error: It is the square-root of the sum of the mean of the squared errors.

Benchmark Model:

The final project follows this benchmark model as a foundation and enhances it.

Simple Linear regression was used for predictive analysis in the benchmark mdoel. It is examining two things:

a. Is the independent variable (predictor) accurate in predicting the dependent variable (outcome)?
b. Depending on the beta, how strong is the fit?

The regression estimates clarify the relationship between the two variables and the relationship between them.

Simple Linear regression is defined by the formula:

$$y = c + b*x,$$

where,
y = estimated dependent variable score
c = constant
b = regression coefficient
x = score on the independent variable

The linear regression algorithm fits a line on the data points which has the least resulting error.

Evaluation Metrics:

The major evaluation metric used is Root mean square error.

Project Design:

The reason for selecting Regression Analysis is because it uses the same principles of Capital Asset Pricing Model to predict stock prices.

Investopedia defines CAPM as:

“The Capital Asset Pricing Model (CAPM) describes the relationship between systematic risk and expected return for assets, particularly stocks. CAPM is widely used throughout finance for pricing risky securities and generating expected returns for assets given the risk of those assets and cost of capital.”

For more information, visit this post.

Capital Asset Pricing Model (CAPM): Regression Analysis:

In formal financial language, the regression model works with the same underlying principles as the Capital Asset Pricing Model (CAPM) equation. This is the reason why Regression analysis has been used as the model of choice.

$$r_p(t) = \beta_p r_m(t) + \alpha_p(t)$$

where,

$r_p(t)$: Return on portfolio for a given day t

β_p: Co-efficient

r_m: Market returns; viz S&P 500 values

$\alpha_p(t)$: Intercept

CAPM emphasizes that the return on any stock or portfolio is due to the markets. This is why $r_m(t)$ is the independent x-variable in this equation. Since we are predicting the returns on the portfolio value, it is the dependent y-variable in this equation, represented by $r_p(t)$.

The final project will build on this attached benchmark project. It will beat the benchmark project by using additional independent variables in three different models; viz :

- Multivariate Regression Analysis:
  The additional independent variables using stock prices will be:
    - *Bollinger Bands*

$$\textbf{Bollinger Bands} = \frac{\text{Price(t) - Simple Moving Average (t)}}{\text{2 * Standard Deviation (t)}}$$

where each of these values are taken in a given timeframe window of 20-21 days.

- *Simple Moving Average*

$$\textbf{Simple Moving Average} = \frac{\textbf{Price(t)}}{\textbf{Price[t - n:t].mean()}} - 1$$

where each of these values are at a given time t, and n which is the time frame window.

where each of these values are at a given time t, and time n which is the beginning of the time frame window.

The final project uses other methods to enhance the work done in the benchmark, viz:

- Decision Tree Model,
- GradientBoosting Regressor Model
- RandomForestRegressorModel

It also uses many more independent variables in each of the models above, along with the feature importance checked in each case.

Methodology:

Pre-Process the Data:

The Quandl dataframes have the following attributes:

- Date
- Open
- High
- Low
- Close
- Volume
- Ex-Dividend
- Split Ratio
- Adj. Open
- Adj. High
- Adj. Low
- Adj. Close
- Adj. Volume

A sample dataframe with all values for a single stock EXC is here:

Date	Open	High	Low	Close	Volume	Ex-Dividend	Split Ratio	Adj. Open	Adj. High	Adj. Low	EXC	Adj. Volume
2006-03-27	54.820	54.83	53.95	53.97	1652300.0	0.0	1.0	34.533570	34.539870	33.985518	33.998117	1652300.0
2006-03-28	53.975	54.41	53.30	53.65	2818800.0	0.0	1.0	34.001267	34.275293	33.576054	33.796535	2818800.0
2006-03-29	53.800	54.31	53.60	54.17	2045700.0	0.0	1.0	33.891027	34.212298	33.765038	34.124106	2045700.0
2006-03-30	53.800	54.03	52.79	53.19	3466700.0	0.0	1.0	33.891027	34.035914	33.254782	33.506760	3466700.0
2006-03-31	54.070	54.07	52.80	52.90	3627100.0	0.0	1.0	34.061112	34.061112	33.261082	33.324076	3627100.0

Of all these columns, we will use only ***Adj. Close***. This is because the trading data is very volatile and changes several times within a day. Adjusted closing price factors a stock's value to reflect any corporate actions such as dividends, rights offerings and stock splits.

Investopedia defines Adjusted Close as:

"The adjusted closing price amends a stock's closing price to reflect that stock's value after accounting for any corporate actions. It is often used when examining historical returns or doing a detailed analysis of past performance."

For more information, visit this post.

This is the reason why you will notice that each of the stock's adjusted price has been renamed to its ticker symbol in the table below.

On joining the individual stocks, we obtain a new dataframe. For sample data see table:

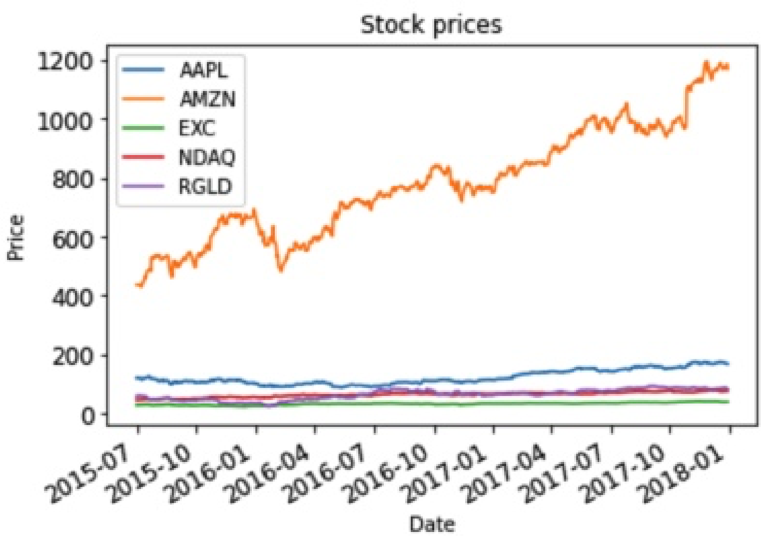
Date	AAPL	AMZN	EXC	NDAQ	RGLD
2015-07-01	121.243068	437.39	29.107162	46.691199	58.987619
2015-07-02	121.089838	437.71	29.400710	47.093792	60.214716
2015-07-06	120.668456	436.04	29.299803	46.739127	61.282387
2015-07-07	120.371574	436.72	30.061194	46.815812	59.548025
2015-07-08	117.383593	429.70	29.905246	45.876428	59.678465

A summary of this entire dataframe is below:

```
df.describe()
```

	AAPL	AMZN	EXC	NDAQ	RGLD
count	629.000000	629.000000	629.000000	629.000000	629.000000
mean	122.975465	778.805509	33.152518	64.564981	64.829244
std	24.597165	186.382364	4.108439	8.047604	17.093386
min	88.288161	429.700000	23.831237	45.876428	24.741683
25%	105.059007	625.900000	30.232317	59.958759	49.692324
50%	113.767235	766.770000	33.346851	66.043219	67.648195
75%	144.473805	948.430000	35.500436	69.816866	79.215683
max	176.420000	1195.830000	42.390000	79.270000	93.978947

On plotting this entire dataframe, we obtain the following results:

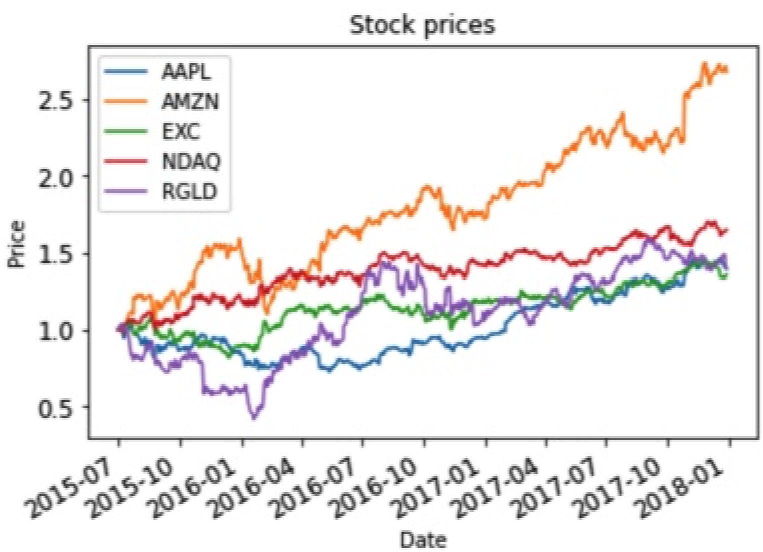


We notice the discrepancy since Amazon is trading at a higher range compared to the other stocks. To overcome this and make a fair comparison, we use data normalization. We will make them all begin at one fair point so that the comparison is on an equal footing. We will normalize price data, so that each value begins at 1.

```
def normalize_data(df):
    df = df/df.iloc[0, :]
    return df
df = normalize_data(df)
df.head()
```

Date	AAPL	AMZN	EXC	NDAQ	RGLD
2015-07-01	1.000000	1.000000	1.000000	1.000000	1.000000
2015-07-02	0.998736	1.000732	1.010085	1.008622	1.020803
2015-07-06	0.995261	0.996914	1.006618	1.001026	1.038903
2015-07-07	0.992812	0.998468	1.032777	1.002669	1.009500
2015-07-08	0.968167	0.982418	1.027419	0.982550	1.011712

On plotting the normalized data, we can observe that the stock prices move more closely together. See image below for visualization of the stocks movement.



Assign Weightages:

We will now use an investment value and assign weights to each stock in the portfolio. Assuming the ***investment value is $10,000*** and we assign weightages in the following order:

Stock	Weights
Apple	25%
Amazon	25%
Exelon Corporation	10%

Nasdaq	30%
Royal Gold Inc.	10%

We now assign the hypothetical investment amount of $10000 to this portfolio. The sum of all the stocks will provide a portfolio value at the end of each day. See table below for sample:

Date	AAPL	AMZN	EXC	NDAQ	RGLD	Portfolio Val
2015-07-01	2500.000000	2500.000000	1000.000000	3000.000000	1000.000000	10000.000000
2015-07-02	2496.840442	2501.829031	1010.085093	3025.867378	1020.802621	10055.424566
2015-07-06	2488.151659	2492.283774	1006.618342	3003.079450	1038.902539	10029.035764
2015-07-07	2482.030016	2496.170466	1032.776552	3008.006569	1009.500410	10028.484013
2015-07-08	2420.418641	2456.046092	1027.418847	2947.649353	1011.711712	9863.244645

A summary of this entire dataframe is below:

```
df.describe()
```

	AAPL	AMZN	EXC	NDAQ	RGLD	Portfolio Val
count	629.000000	629.000000	629.000000	629.000000	629.000000	629.000000
mean	2535.721572	4451.436411	1138.981482	4148.425086	1099.031372	13373.595923
std	507.187034	1065.309928	141.148745	517.074148	289.779223	2352.693243
min	1820.478525	2456.046092	818.741340	2947.649353	419.438576	9500.286457
25%	2166.288949	3577.470907	1038.655613	3852.466429	842.419570	11281.620234
50%	2345.850303	4382.644779	1145.657948	4243.404754	1146.820239	13302.268737
75%	2979.011651	5420.962985	1219.646102	4485.868853	1342.920516	15365.576956
max	3637.733744	6835.032808	1456.342620	5093.251034	1593.197851	18309.025416

Mean value of portfolio is: 13373.59. ***10% of mean portfolio value is: 1337.36.*** We will use this value to check the accuracy of the model by comparing it with RMSE later in the report.

We now have all the dependent variables ready in the form of the column ‘Portfolio Val’. For the independent variable, we will use market data that is derived from S&P500 daily values. This data is obtained from investing.com.

Implementation:

Prepare and Split the Data

Let us now split the rows in the dataset up into train and test sets using the code below.

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20)
```

Decision Tree Model

We will create a Decision Tree model using Scikit-learn. Next, we will evaluate the accuracy by using the score function. We will use this to enable a model for predicting the stock price value by learning the rules of decisions derived from the data.

```
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor().fit(x_train, y_train)
regressor.score(x_train, y_train), regressor.score(x_test, y_test)
```

```
(1.0, 0.997878347909761)
```

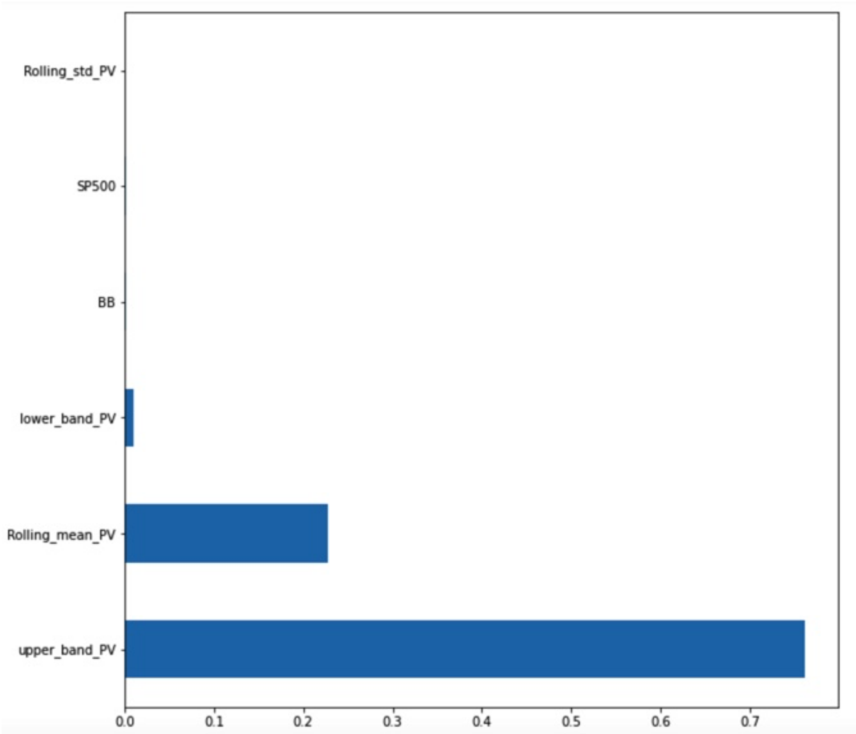
Feature Importance of Independent Variables

We will guage the importance of each of the x variables used in the DecisionTreeRegressor class. We will use the feature_importances_ property to get the importance scores for the features used in the input.

```
importance = regressor.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
    print(X_head.columns[i], ':', v)
```

The results suggest that 2 out of 6 variables are of relative importance. Next, we will plot a bar chart to visualize the same.

SP500 : 0.0006735943188128613
Rolling_mean_PV : 0.22689630024575835
Rolling_std_PV : 0.00012982130818200963
upper_band_PV : 0.7610221994661466
lower_band_PV : 0.010009647037572211
BB : 0.0012684376235279102



Gradient Boosting Regressor

We will create a Gradient Boosting Regressor model using Scikit-learn. Next, we will evaluate the accuracy by using the score function. Gradient Boosting is used to get a predictive model using an ensemble of relatively weaker predictive models.

```
model = GradientBoostingRegressor(random_state=0).fit(x_train, y_train)
model.score(x_train, y_train), model.score(x_test, y_test)
```

```
(0.9998678173167533, 0.9985472125032997)
```

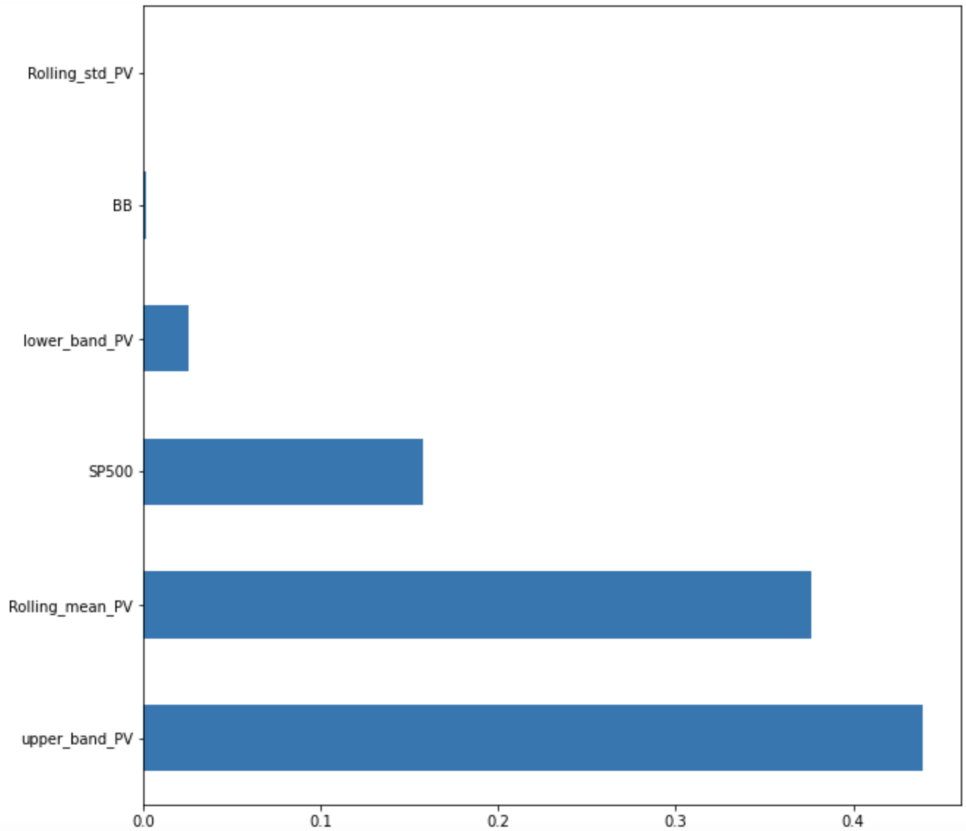
Feature Importance of Independent Variables

We will guage the importance of each of the x variables used in the GradientBoostingRegressor class. We will use the feature_importances_ property to get the importance scores for the features used in the input.

The results suggest that 4 out of 6 variables are of relative importance. Next, we will plot a bar chart to visualize the same.

```
importance = model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
    print(X_head.columns[i], ':', v)
```

SP500 : 0.1579961854242738
Rolling_mean_PV : 0.3765182172201925
Rolling_std_PV : 0.00016107226256443754
upper_band_PV : 0.4387701436536028
lower_band_PV : 0.02528940759103104
BB : 0.00126497384833529



Random Forest Regressor:

We will create a Random Forest Regressor model using Scikit-learn. Next, we will evaluate the accuracy by using the score function. It is a supervised learning model that uses ensemble regression learning. A Random Forest works by building several decision trees while training and resulting in the mean of all the classes for the prediction of all the decision trees.

```
model = RandomForestRegressor(random_state=0).fit(x_train, y_train)
model.score(x_train, y_train), model.score(x_test, y_test)
```

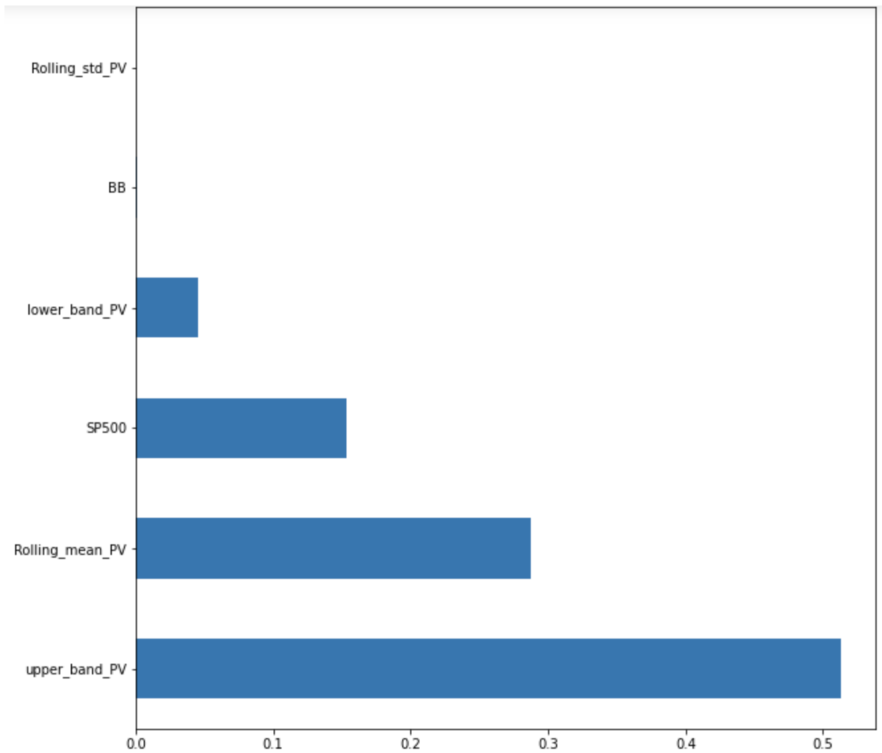
```
(0.9998116326114512, 0.9977983687350885)
```

Feature Importance of Independent Variables

We will guage the importance of each of the x variables used in the RandomForestRegressor class. We will use the feature_importances_ property to get the importance scores for the features used in the input.

The results suggest that 4 out of 6 variables are of relative importance. Next, we will plot a bar chart to visualize the same.

SP500 : 0.15348272179214484
Rolling_mean_PV : 0.28764912650160457
Rolling_std_PV : 0.00024029568367875013
upper_band_PV : 0.5125945558941888
lower_band_PV : 0.045016439131914575
BB : 0.0010168609964684998



Refinement:

The following steps are in consideration as further steps to further refine the project:

1. Upload data to AWS S3
2. Train the model and deploy it using AWS Sagemaker
3. Enable a web interface to enable users to pick custom stocks to create portfolio of choice (by using gradio.app)
4. Print dashboard for the portfolio that has been created for visualization

Model Evaluation & Validation:

We will compare the predicted values y_pred with the actual output values y_test and plot them in a table. In order to compare the predicted values y_pred with the actual output values y_test we will execute this script for each of the three models in consideration:

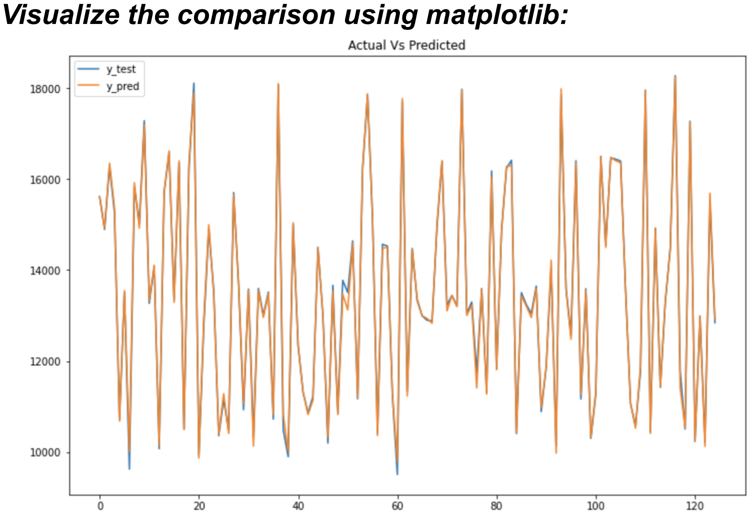
```
y_pred = regressor.predict(x_test)
```

```
dfYt = pd.DataFrame(y_test, columns=['y_test'])
dfYp = pd.DataFrame(y_pred, columns=['y_pred'])
dfYt = dfYt.join(dfYp)
dfYt['difference']=dfYt['y_test']-dfYt['y_pred']
dfYt = dfYt[['y_test', 'y_pred', 'difference']]
dfYt
```

Decision Tree Model

	y_test	y_pred	difference
0	15618.487867	15618.006799	0.481068
1	14896.203893	14923.104018	-26.900125
2	16292.208540	16353.551123	-61.342583
3	15261.746864	15365.576956	-103.830091
4	10705.228181	10677.029831	28.198350
...	...	...	...
120	10228.591291	10238.354460	-9.763169
121	12977.103005	12995.679791	-18.576786
122	10220.186922	10119.242847	100.944075
123	15515.917854	15694.102590	-178.184736
124	12842.000629	12935.290767	-93.290137

Visualize the comparison using matplotlib:



We will evaluate the accuracy of the model using three commonly used methods:

1. Mean Absolute Error : It is the mean of the absolute value of the errors.
2. Mean Squared Error: It is the sum of the mean of the squared errors.
3. Root Mean Squared Error: It is the square-root of the sum of the mean of the squared errors.

```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

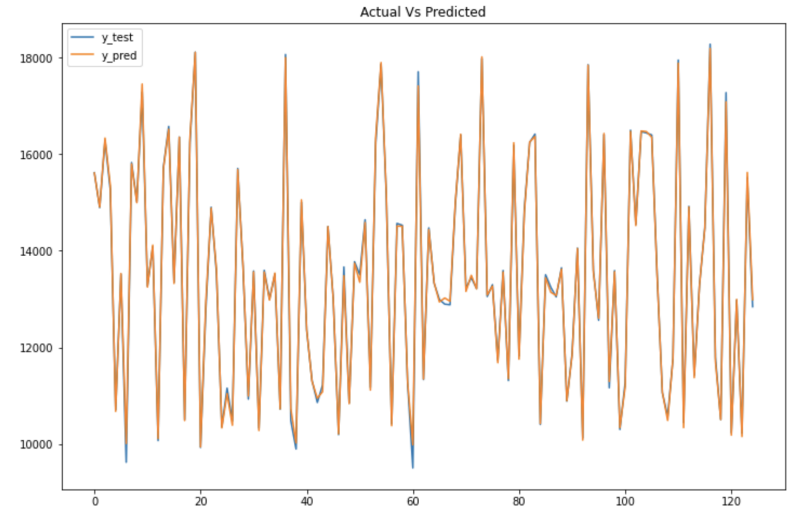
```
Mean Absolute Error: 72.34442825175127
Mean Squared Error: 11751.458504747217
Root Mean Squared Error: 108.4041443153684
```

RMSE is 131.53. The 10% of mean portfolio value is: 1337.36. Since RMSE is lower that 10% of the mean portfolio value, we can conclude that this model did a fairly good job of predicting values.

Gradient Boosting Regressor

	y_test	y_pred	difference
0	15618.487867	15612.551247	5.936620
1	14896.203893	14906.662555	-10.458662
2	16292.208540	16336.860664	-44.652124
3	15261.746864	15347.154033	-85.407169
4	10705.228181	10672.233066	32.995115
...	...	...	...
120	10228.591291	10179.488375	49.102916
121	12977.103005	12993.022065	-15.919060
122	10220.186922	10150.394601	69.792321
123	15515.917854	15624.430305	-108.512452
124	12842.000629	12988.220247	-146.219618

Visualize the comparison using matplotlib.



We will evaluate the accuracy of the model using three commonly used methods:

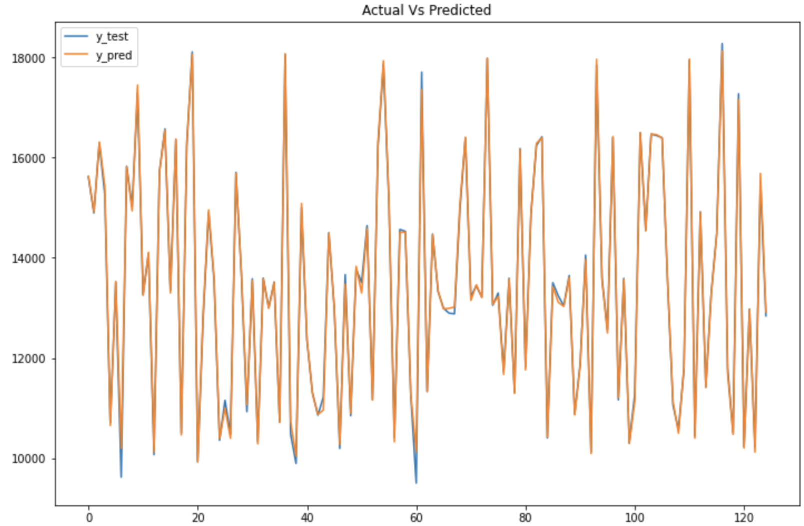
1. Mean Absolute Error : It is the mean of the absolute value of the errors.
2. Mean Squared Error: It is the sum of the mean of the squared errors.
3. Root Mean Squared Error: It is the square-root of the sum of the mean of the squared errors.

Mean Absolute Error: 57.14285478961184
Mean Squared Error: 8046.73492993216
Root Mean Squared Error: 89.70359485512361
RMSE is 80.14. The 10% of mean portfolio value is: 1337.36. Since RMSE is lower that 10% of the mean portfolio value, we can conclude that this model did a fairly good job of predicting values.

Random Forest Regressor

	y_test	y_pred	difference
0	15618.487867	15629.201037	-10.713170
1	14896.203893	14915.724844	-19.520951
2	16292.208540	16314.568286	-22.359746
3	15261.746864	15427.804661	-166.057796
4	10705.228181	10650.228211	54.999970
...	...	...	...
120	10228.591291	10198.174813	30.416477
121	12977.103005	12968.951082	8.151924
122	10220.186922	10118.709029	101.477894
123	15515.917854	15687.771474	-171.853620
124	12842.000629	12919.475905	-77.475276

Visualize this comparison using matplotlib.



We will evaluate the accuracy of the model using three commonly used methods:

1. Mean Absolute Error : It is the mean of the absolute value of the errors.
2. Mean Squared Error: It is the sum of the mean of the squared errors.
3. Root Mean Squared Error: It is the square-root of the sum of the mean of the squared errors.

Mean Absolute Error: 63.25899263875747
Mean Squared Error: 12194.449114156228
Root Mean Squared Error: 110.42847963345429

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

RMSE is 80.14. The 10% of mean portfolio value is: 1337.36. Since RMSE is lower that 10% of the mean portfolio value, we can conclude that this model did a fairly good job of predicting values.

Justification:

	Benchmark: Simple Linear Regression	Decision Tree Model	Gradient Boosting Regressor	Random Forest Regressor
Mean Absolute Error	496.30	72.34	57.14	63.26
Mean Squared Error	435980.02	11751.46	8046.73	12194.45
Root Mean Squared Error	660.28	108.40	89.70	110.43

We can see from the above Root Mean Squared Error scores that the three models used in the final project beat the benchmark project which used Simple Linear Regression. The three models used here can be ranked in the following order based on their RMSE values:

1. Gradient Boosting Regressor Model
2. Decision Tree Model
3. Random Forest Regressor