Data Engineering Project

Group 3

Gold Price Prediction

Mentor-Prof. Romi Banerjee

Group Members:

- **Arjun Bhattad(B22AI051)**
- **Dev Pandya(B22AI016)**
- **Chirag Kumar(B22AI056)**

Technologies Used:

- Python
- Jupyter Notebook
- Numpy, Pandas, Sklearn, Matplotlib, Seaborn, Sql Connector
- MySQL
- Docker
- Github
- HTML, CSS, Javascript

Important Links:

- Github Repository: https://github.com/ArjunBhattad2004/Gold_Price_Prediction/tree/main
- Project Website: https://goldpricepredictiondeproject.netlify.app/
- Dockerhub Repository: https://hub.docker.com/repository/docker/arjunbhattad/gold-price-prediction/general

Progress Timeline:

- Lab-3: Data Collection, Testing, Comparison of Datasets
- Lab-4: Data Wrangling, Database Creation, Data Preprocessing
- Lab-5: Connection of Database to the Project, Implementation and Comparison of Models, Dockerization of Project, Webpage Implementation

Kaggle Dataset

I. Definition

Project Overview

Gold prices fluctuate due to a variety of factors, including currency exchange rates, stock market trends, and geopolitical conditions. Accurate prediction of gold prices can help investors make informed decisions about buying or selling.

This part of the project focuses on using the **Kaggle Dataset**, containing historical gold price data, to predict the **Adjusted Closing Price** of gold. Various machine learning algorithms are applied, and their performance is compared to identify the best solution for gold price prediction.

Problem Statement

The challenge is to predict the future adjusted closing price of gold accurately. The task is modeled as a regression problem, as the target variable (gold price) is continuous.

Steps Followed in the Project

1. Data exploration and visualization.
2. Statistical analysis and feature engineering.
3. Normalizing and preparing data for machine learning.
4. Splitting the dataset into training and testing sets.
5. Applying machine learning algorithms, including benchmark models.
6. Comparing results based on evaluation metrics.
7. Refining models using feature selection.

Evaluation Metrics

1. **Root Mean Square Error (RMSE):** Measures the standard deviation of prediction errors. Lower RMSE indicates better model performance.
2. **R^2 Score:** Indicates the proportion of variance explained by the model. Higher R^2 implies better predictions.

II. Analysis

Data Exploration

The Kaggle Dataset includes features like:

- **Gold Price (GLD):** Historical open, high, low, close, adjusted close, and volume.
- **Economic Indicators:** S&P 500 index, crude oil prices, EUR/USD exchange rate.

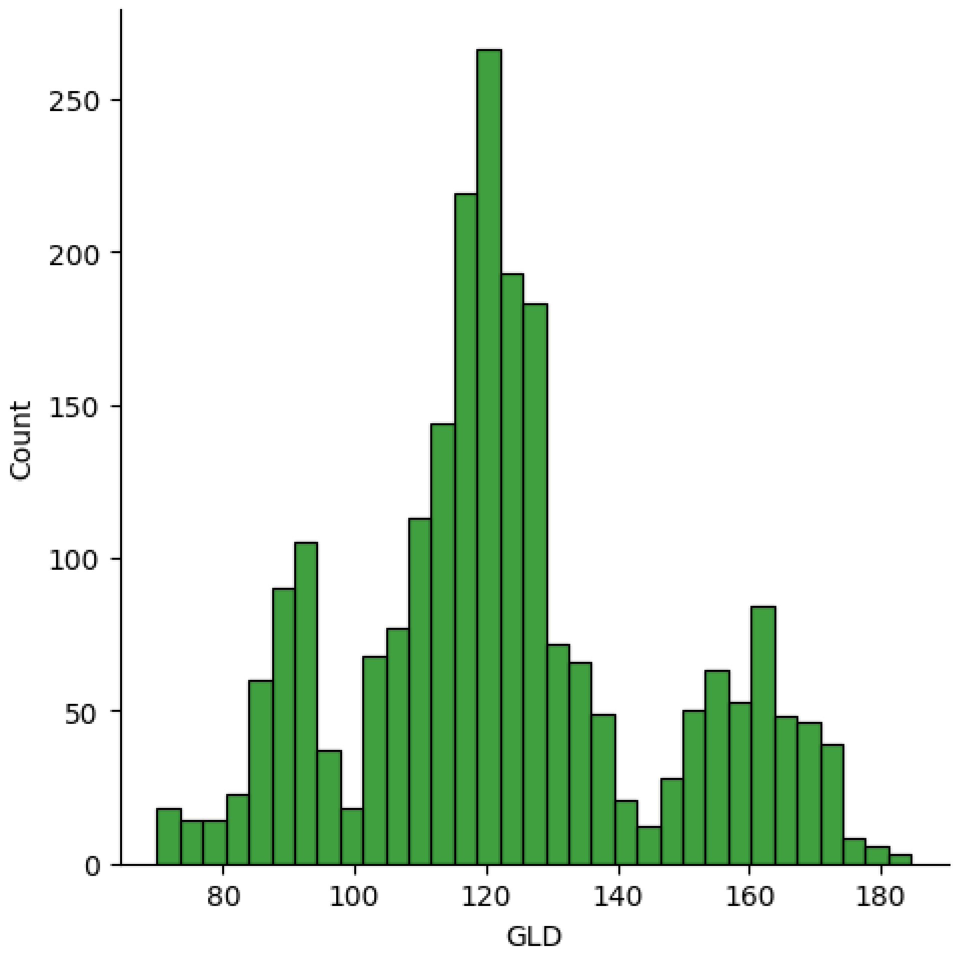
Features are analyzed for missing values, trends, and correlations with the target variable.

Visualizations:

- Line plots for historical gold prices.
- Heatmaps to identify correlations between features and the target.

Statistical Analysis

- Mean, standard deviation, and kurtosis are calculated to understand the data distribution.
- Correlation analysis identifies key predictors influencing gold prices.



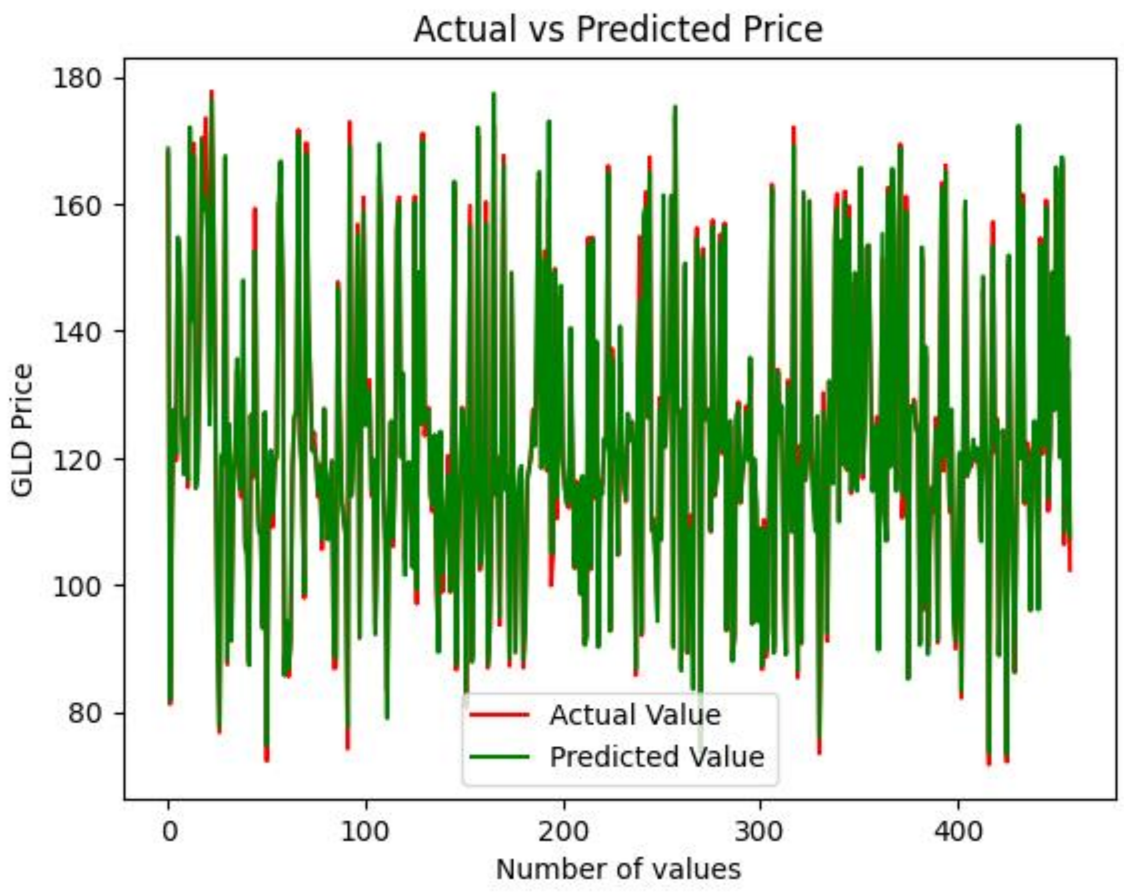
:

III. Results

Random Forest Regressor Model Evaluation

Key Takeaways:

1. Gold prices are influenced by a combination of market indices, crude oil prices, and currency exchange rates.
2. Random Forest Regreessor provides robust predictions for gold prices.



FINAL_USO Dataset

I. Definition

Project Overview

Historically, gold had been used as a form of currency in various parts of the world including USA. In present times, precious metals like gold are held with central banks of all countries to guarantee re-payment of foreign debts, and also to control inflation which results in reflecting the financial strength of the country. Recently, emerging world economies, such as China, Russia, and India have been big buyers of gold, whereas USA, South Africa, and Australia are among the big seller of gold.

Forecasting rise and fall in the daily gold rates, can help investors to decide when to buy (or sell) the commodity. But Gold prices are dependent on many factors such as prices of other precious metals, prices of crude oil, stock exchange performance, Bonds prices, currency exchange rates etc.

We in this project would forecast gold rates using the most comprehensive set of features and would apply various machine learning algorithms for forecasting and compare their results. We also identify the attributes that highly influence the gold rates.

We would use SPDR Gold Trust (GLD) Exchange Traded Fund data downloaded from https://finance.yahoo.com in the date range of **18/11/2004** to **01/01/2019**. We would try to predict **Adjusted Close price** of GLD ETF, the detail about the data would be described in the **Dataset and Input** section below.

Problem Statement

The challenge of this project is to accurately predict the future adjusted closing price of Gold ETF across a given period of time in the future. The problem is a regression problem, because the output value which is the adjusted closing price in this project is continuous value.

Various studies have been conducted by researchers to forecast gold rates using different machine learning algorithms with varying degrees of success but until recently the ability to build these models has been restricted to academics. Now with libraries like **Scikit-learn** anyone can build powerful predictive models.

For this project I will use different linear, ensemble and boosting machine learning models to predict the adjusted closing price of the SPDR Gold Trust (GLD) ETF using a dataset of past prices from **18/11/2004** to **01/01/2019**

Steps to be followed during Project

1. Exploring Gold ETF closing prices
2. Perform statistical analysis
3. Perform Feature Engineering
4. Normalizing the data
5. Splitting the dataset
6. Implement benchmark machine learning model and different solution models
7. Compare benchmark model with different machine learning models based on evaluation metrics described below.
8. Perform Feature Selection
9. Compare Feature selected models with full feature models

Metrics

I would use **Root Mean Square Error** and **R2 score** as my evaluation metrics. Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are. The formula for calculating RMSE is given below

$$RMSE = \sqrt{\overline{(f-o)^2}}$$

Where f is the Predicted or forecasted value and o is observed or Actual value.

RMSE is always non-negative, and a value of 0 (almost never achieved in practice) would indicate a perfect fit to the data.

R-squared is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression.

R-squared is always between 0 and 100%:

- 0% indicates that the model explains none of the variability of the response data around its mean.
- 100% indicates that the model explains all the variability of the response data around its mean.

So, when we apply both evaluation metrics to our benchmark and solution models , we would choose the one which has lower RMSE value and higher R2 score value.

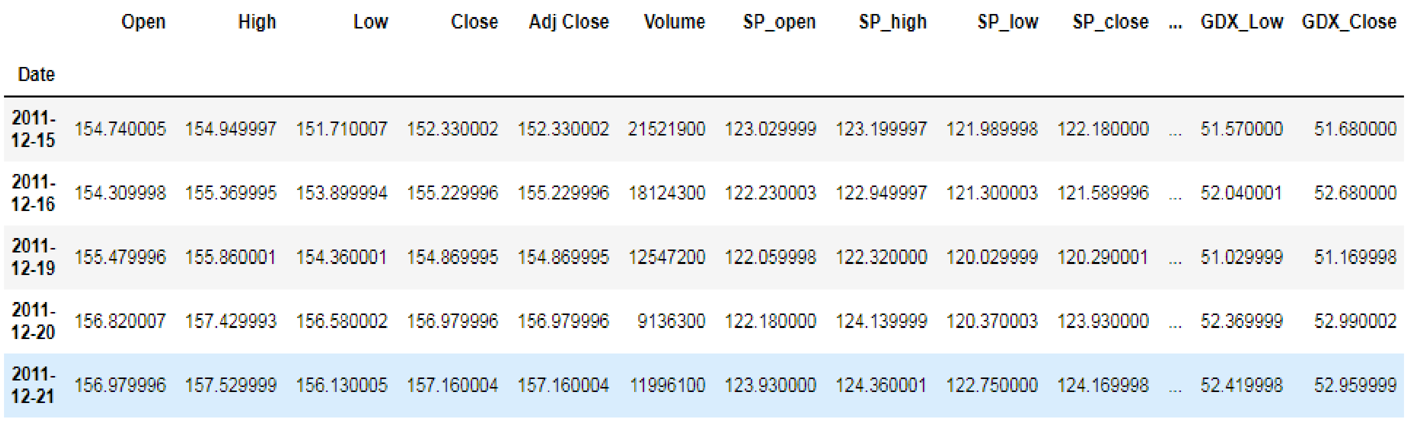
II. Analysis

Data Exploration

Data for this study is collected from November 18th 2011 to January 1st 2019 from various sources. The data has 1718 rows in total and 80 columns in total. Data for attributes, such as Oil Price, Standard and Poor's (S&P) 500 index, Dow Jones Index US Bond rates (10 years), Euro USD exchange rates, prices of precious metals Silver and Platinum and other metals such as Palladium and Rhodium, prices of US Dollar Index, Eldorado Gold Corporation and Gold Miners ETF were gathered.

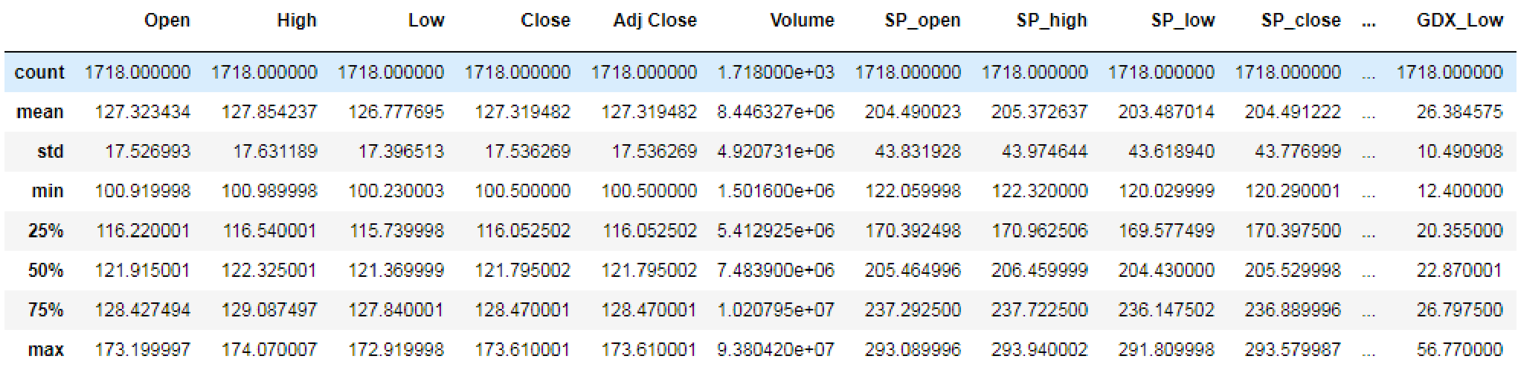
The historical data of Gold ETF fetched from Yahoo finance has 7 columns, Date, Open, High, Low, Close, Adjusted Close and Volume, the difference between **Adjusted Close** and **Close** is that closing price of a stock is the price of that stock at the close of the trading day. Whereas the adjusted closing price takes into account factors such as dividends, stock splits and new stock offerings to determine a value. We would use Adjusted Close as our outcome variables which is the value we want to predict. 'SP_open', 'SP_high', 'SP_low', 'SP_close', 'SP_Ajclose', 'SP_volume' of **S&P 500 Index**, 'DJ_open','DJ_high', 'DJ_low', 'DJ_close', 'DJ_Ajclose', 'DJ_volume' of **Dow Jones Index**, 'EG_open', 'EG_high', 'EG_low', 'EG_close', 'EG_Ajclose', 'EG_volume' of **Eldorado Gold Corporation (EGO)**, 'EU_Price','EU_open', 'EU_high', 'EU_low', 'EU_Trend' of **EUR USD Exchange rate**, 'OF_Price', 'OF_Open', 'OF_High', 'OF_Low', 'OF_Volume', 'OF_Trend' of **Brent Crude oil Futures,** 'OS_Price', 'OS_Open', 'OS_High', 'OS_Low', 'OS_Trend', of **Crude Oil WTI USD,** 'SF_Price', 'SF_Open', 'SF_High', 'SF_Low', 'SF_Volume', 'SF_Trend' of **Silver Futures**, 'USB_Price', 'USB_Open', 'USB_High','USB_Low', 'USB_Trend' of **US Bond Rate data**, 'PLT_Price', 'PLT_Open', 'PLT_High', 'PLT_Low','PLT_Trend' of **Platinum Price**, 'PLD_Price', 'PLD_Open', 'PLD_High', 'PLD_Low','PLD_Trend' of **Palladium price** 'RHO_PRICE' of **Rhodium Prices** 'USDI_Price', 'USDI_Open', 'USDI_High','USDI_Low', 'USDI_Volume', 'USDI_Trend' of **US dollar Index Price**, 'GDX_Open', 'GDX_High', 'GDX_Low', 'GDX_Close', 'GDX_Adj Close', 'GDX_Volume' of **Gold Miners ETF**, 'USO_Open','USO_High', 'USO_Low', 'USO_Close', 'USO_Adj Close', 'USO_Volume' of **Oil ETF USO** are used as feature engineered variables. Date is a kind of categorical data, and we won't use to train, we would use date to sort the data by making it as index.

Below is the snapshot of the data.



	Open	High	Low	Close	Adj Close	Volume	SP_open	SP_high	SP_low	SP_close	...	GDX_Low	GDX_Close
Date													
2011-12-15	154.740005	154.949997	151.710007	152.330002	152.330002	21521900	123.029999	123.199997	121.989998	122.180000	...	51.570000	51.680000
2011-12-16	154.309998	155.369995	153.899994	155.229996	155.229996	18124300	122.230003	122.949997	121.300003	121.589996	...	52.040001	52.680000
2011-12-19	155.479996	155.860001	154.360001	154.869995	154.869995	12547200	122.059998	122.320000	120.029999	120.290001	...	51.029999	51.169998
2011-12-20	156.820007	157.429993	156.580002	156.979996	156.979996	9136300	122.180000	124.139999	120.370003	123.930000	...	52.369999	52.990002
2011-12-21	156.979996	157.529999	156.130005	157.160004	157.160004	11996100	123.930000	124.360001	122.750000	124.169998	...	52.419998	52.959999

Below is the preview of detailed summary of data such as count, mean, standard deviation etc.



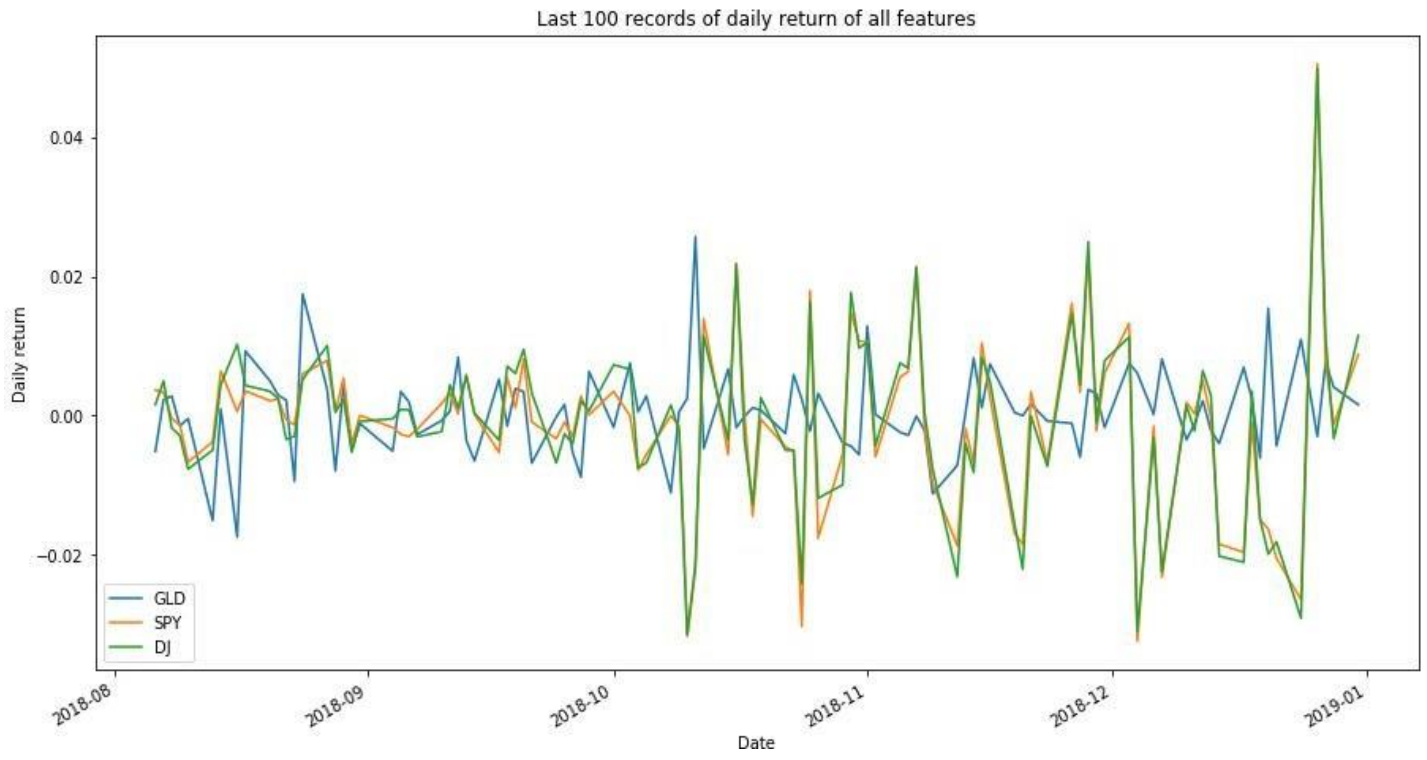
	Open	High	Low	Close	Adj Close	Volume	SP_open	SP_high	SP_low	SP_close	...	GDX_Low
count	1718.000000	1718.000000	1718.000000	1718.000000	1718.000000	1.718000e+03	1718.000000	1718.000000	1718.000000	1718.000000	...	1718.000000
mean	127.323434	127.854237	126.777695	127.319482	127.319482	8.446327e+06	204.490023	205.372637	203.487014	204.491222	...	26.384575
std	17.526993	17.631189	17.396513	17.536269	17.536269	4.920731e+06	43.831928	43.974644	43.618940	43.776999	...	10.490908
min	100.919998	100.989998	100.230003	100.500000	100.500000	1.501600e+06	122.059998	122.320000	120.029999	120.290001	...	12.400000
25%	116.220001	116.540001	115.739998	116.052502	116.052502	5.412925e+06	170.392498	170.962506	169.577499	170.397500	...	20.355000
50%	121.915001	122.325001	121.369999	121.795002	121.795002	7.483900e+06	205.464996	206.459999	204.430000	205.529998	...	22.870001
75%	128.427494	129.087497	127.840001	128.470001	128.470001	1.020795e+07	237.292500	237.722500	236.147502	236.889996	...	26.797500
max	173.199997	174.070007	172.919998	173.610001	173.610001	9.380420e+07	293.089996	293.940002	291.809998	293.579987	...	56.770000

Exploratory Visualization

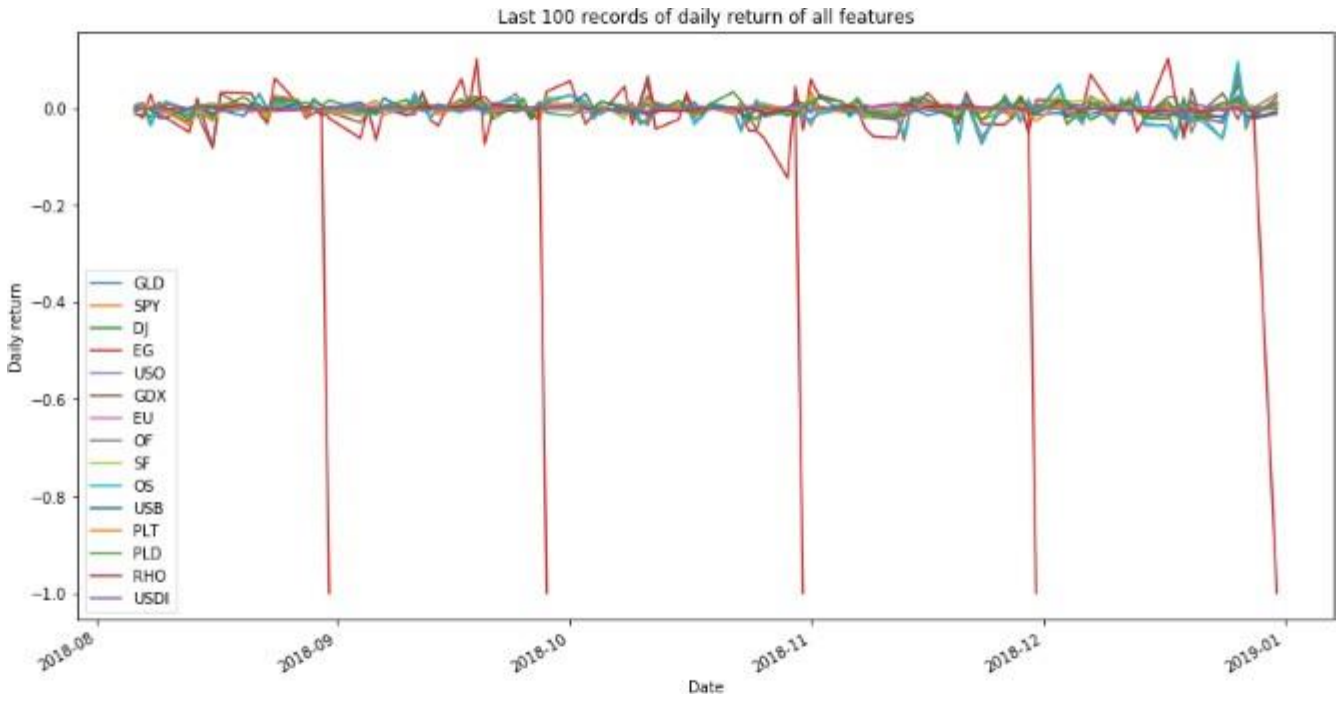
Below image shows the daily return of stock indexes with Gold ETF. In the image it is easily understood that price trend of Standards and Poors (S&P 500) Index and Dow Jones Index is somewhat same. But in some places Gold price shows reverse trend with respect to S&P and Dow Jones.

To visualize the data I have used **matplotlib** library.

Following is the snapshot of the plotted data:



Below image shows the daily returns of last records of all the predictor variables in the dataset.



Technical Indicators

Apart from other features which I have already mentioned above, I have added following technical indicators which I feel help as a feature for prediction of Gold price

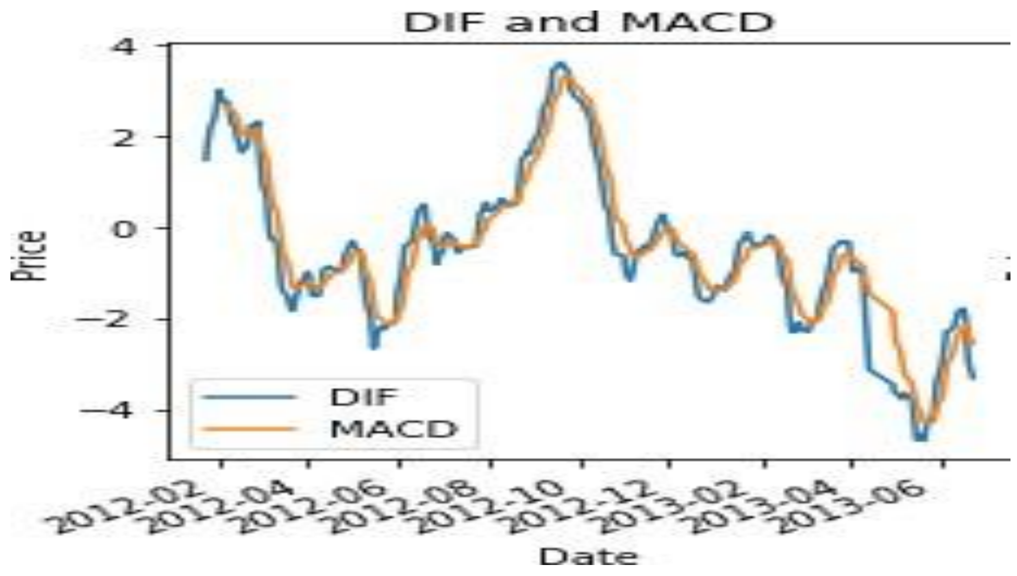
1. **MACD :** The **M**oving **A**verage **C**onvergence-**D**ivergence (MACD) is one of the most powerful and well-known indicators in technical analysis. The indicator is comprised of two exponential moving averages (EMAs) that help measure momentum in a security. The MACD is simply the difference between these two moving averages plotted against a centerline, where the centerline is the point at which the two moving averages are equal.

$$\boxed{\begin{array}{l} \boldsymbol{DIF} = \boldsymbol{EMA}(\text{close},12) - \boldsymbol{EMA}(\text{close},26) \\ \\ \boldsymbol{MACD} = \boldsymbol{EMA}(\boldsymbol{DIF},9) \end{array}}$$

   MACD is calculated by first calculating **DIF** which is the difference of 12 days of **Exponential Moving Average (EMA)** of Adjusted close price and 26 days **EMA**. After calculating **DIF MACD** is calculated which is 9 days **EMA** of **DIF**

   Pseudocode

```
def calculate_MACD(df, nslow=26, nfast=12):
    emaslow = df.ewm(span=nslow, min_periods=nslow, adjust=True, ignore_na=False).mean()
    emafast = df.ewm(span=nfast, min_periods=nfast, adjust=True, ignore_na=False).mean()
    dif = emafast - emaslow
    MACD = dif.ewm(span=9, min_periods=9, adjust=True, ignore_na=False).mean()
    return dif, MACD
```

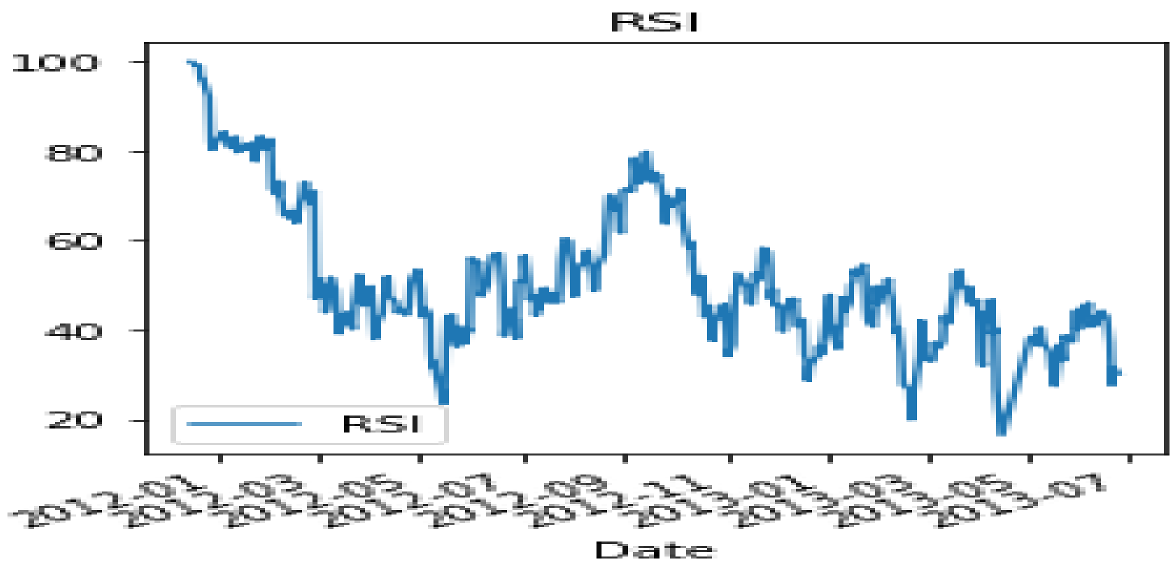


2. **RSI :** The **R**elative **S**trength **I**ndex (RSI) is another well known momentum indicators that's widely used in technical analysis. The indicator is commonly used to identify overbought and oversold conditions in a security with a range between 0 (oversold) and 100 (overbought). RSI can also be used to identify the general trend. The basic formula is:

RSI = 100 - [100 / (1 + (Average of Upward Price Change / Average of Downward Price Change))]

Pseudocode:

```
def calculate_RSI(df, periods=14):
    # wilder's RSI
    delta = df.diff()
    up, down = delta.copy(), delta.copy()

    up[up < 0] = 0
    down[down > 0] = 0

    rUp = up.ewm(com=periods,adjust=False).mean()
    rDown = down.ewm(com=periods, adjust=False).mean().abs()

    rsi = 100 - 100 / (1 + rUp / rDown)
    return rsi
```



3. **Simple Moving Average (SMA): S**imply takes the sum of all of the past closing prices over a time period and divides the result by the total number of prices used in the calculation.

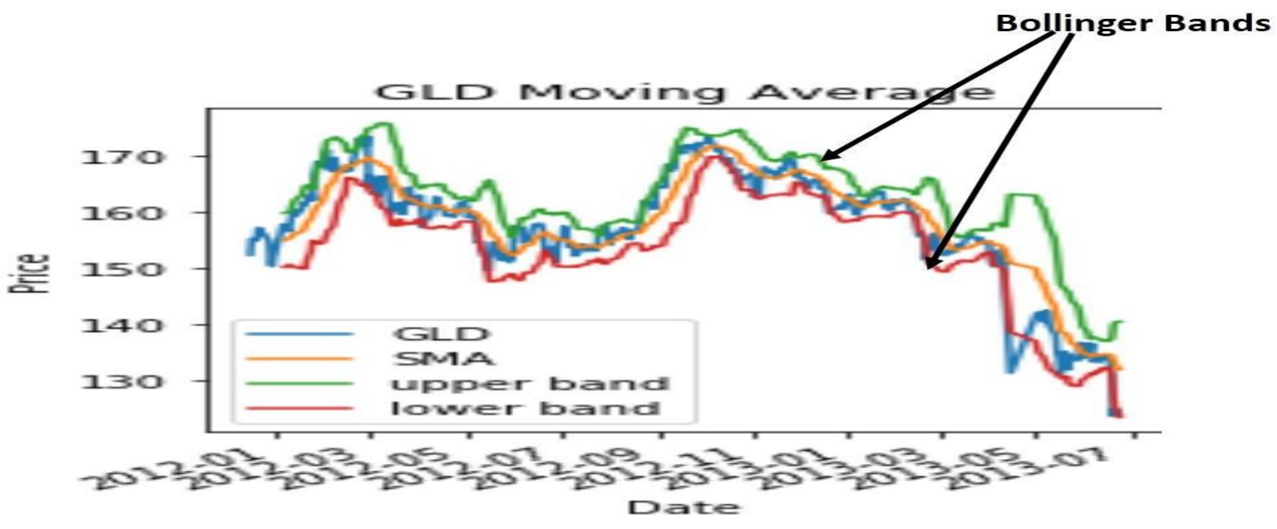
SMA = (A1+A2+A3+…..An)/n

Here **An** is the adjusted close price of period **n**, **n** is the number of total period in our case it is **15 days**.

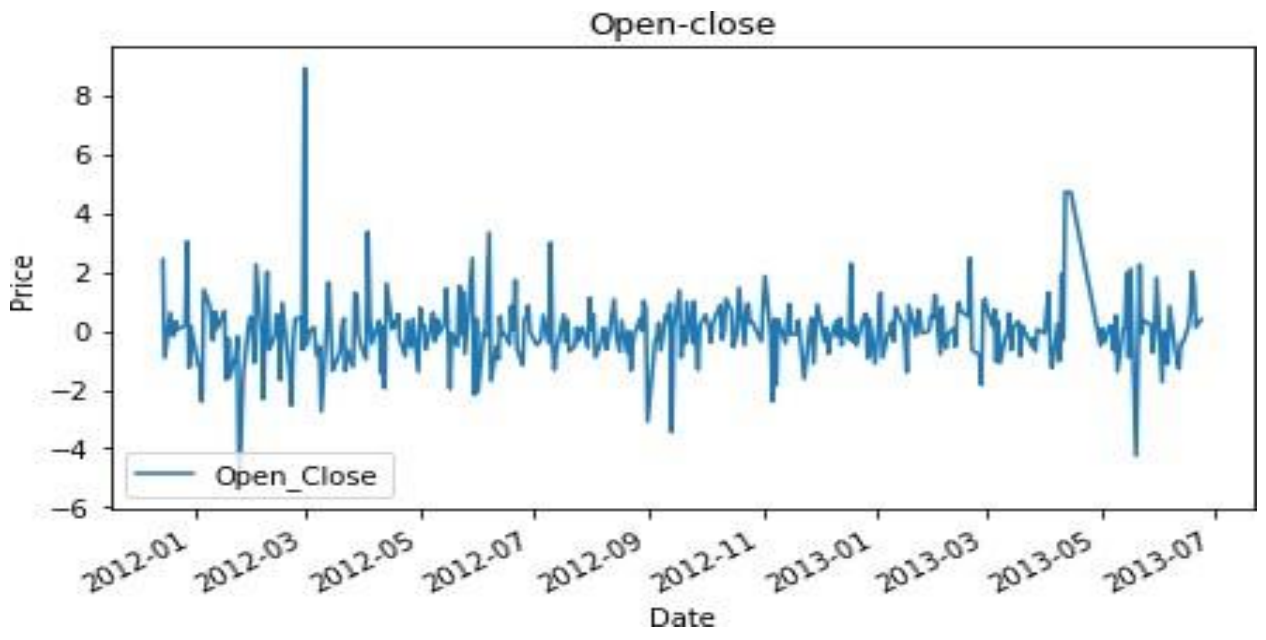
4. **Bollinger Bands:** Bollinger Bands are a technical analysis tool developed by John Bollinger in the 1980s for trading stocks. The bands comprise a volatility indicator that measures the relative high or low of a security's price in relation to previous trades. Volatility is measured using standard deviation, which changes with increases or decreases in volatility.

   Bollinger Bands are comprised of three lines: upper, middle and lower band. The middle band is a moving average. The upper and lower bands are positioned on either side of the moving average band.

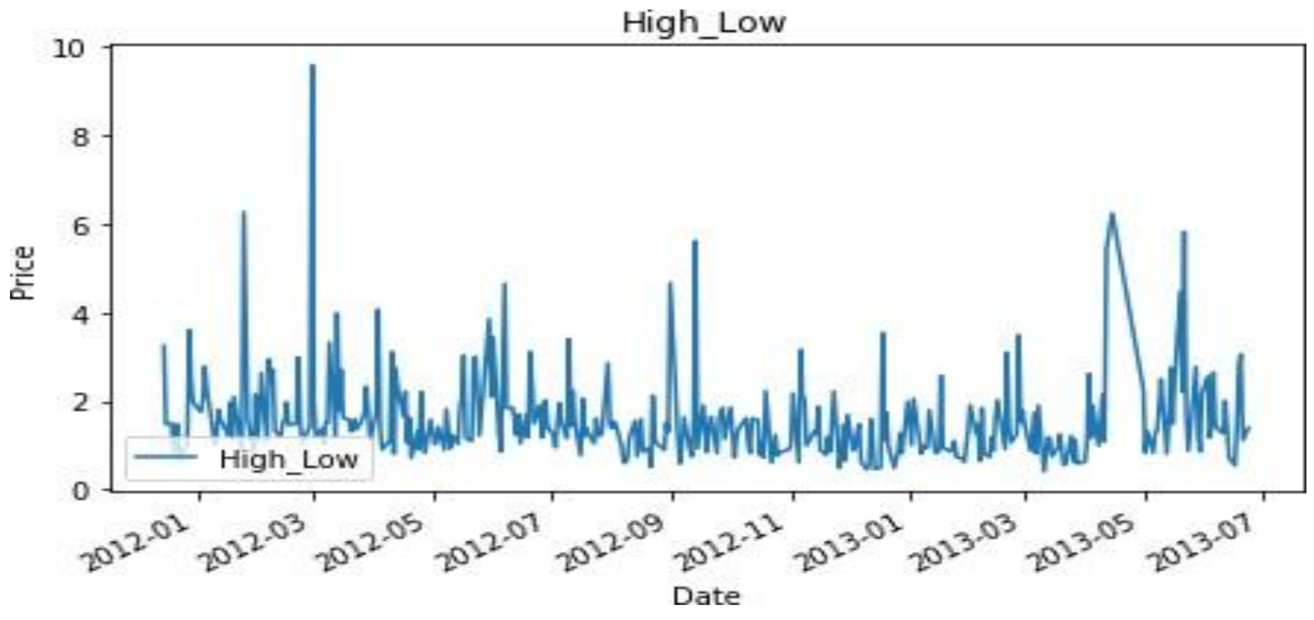
   Upper Band is 2 standard deviation above the middle band whereas lower band is 2 standard deviation below the middle band.



5. **Open – Close** : It is calculated by taking difference between Open prices and Closing prices of stocks.

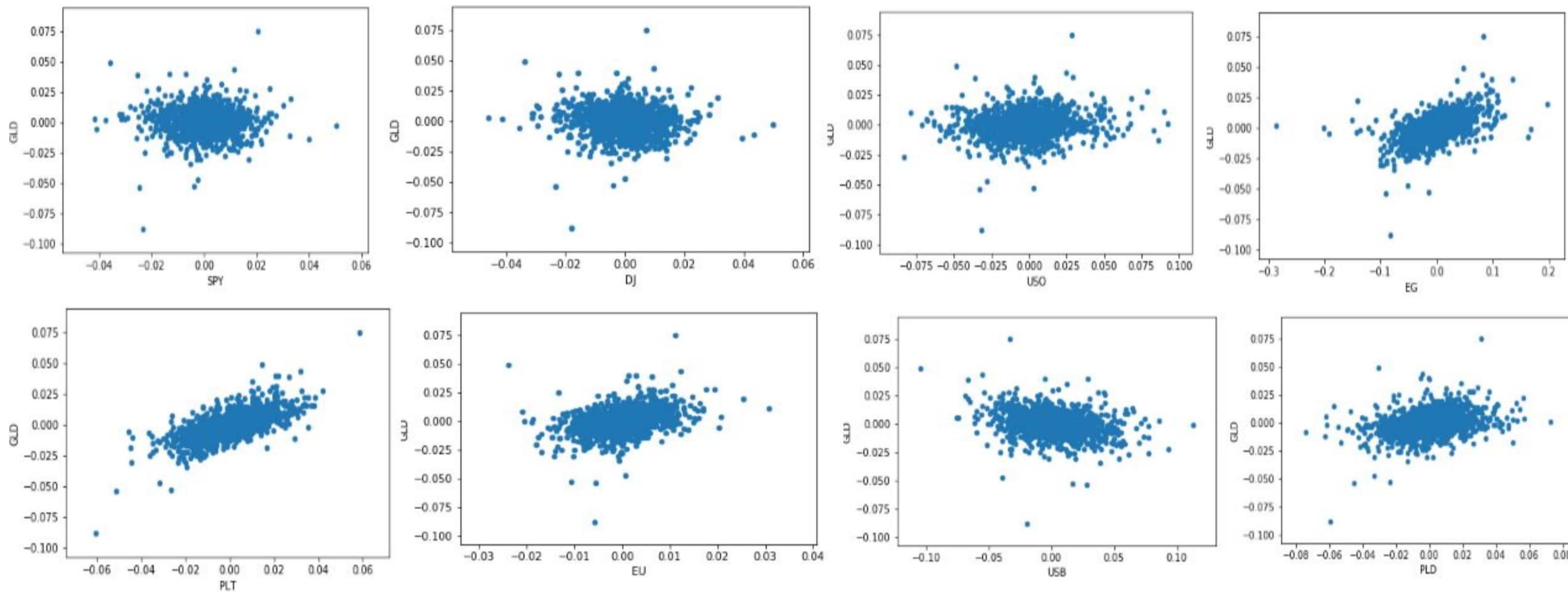


6. **High – Low :** It is calculated by taking difference between High and Low prices.



Scatterplots

Below scatterplot shows relationship between Gold Adjusted price and different features such as Stock indexes SPY 500, Dow Jones Index, Prices of Platinum, Palladium, United States 10 year Bonds, Euro-USD Exchange rate etc.



Statistical Measures

Kurtosis is a statistical measure that is used to describe the distribution. Whereas skewness differentiates extreme values in one versus the other tail, kurtosis measures extreme values in either tail.

Distributions with large kurtosis exhibit tail data exceeding the tails of the normal distribution (e.g., five or more standard deviations from the mean).

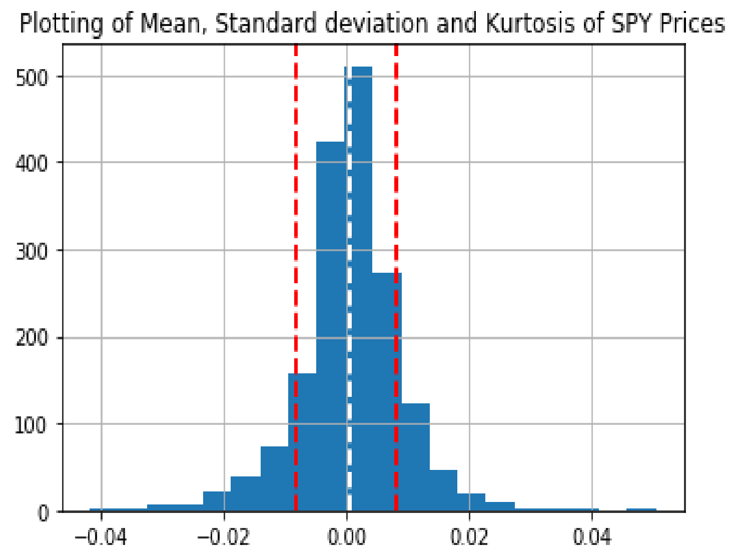
Distributions with low kurtosis exhibit tail data that is generally less extreme than the tails of the normal distribution.

We have calculated mean, standard deviation and kurtosis of all the stock indexes and Gold rates in terms of Adjusted close price.

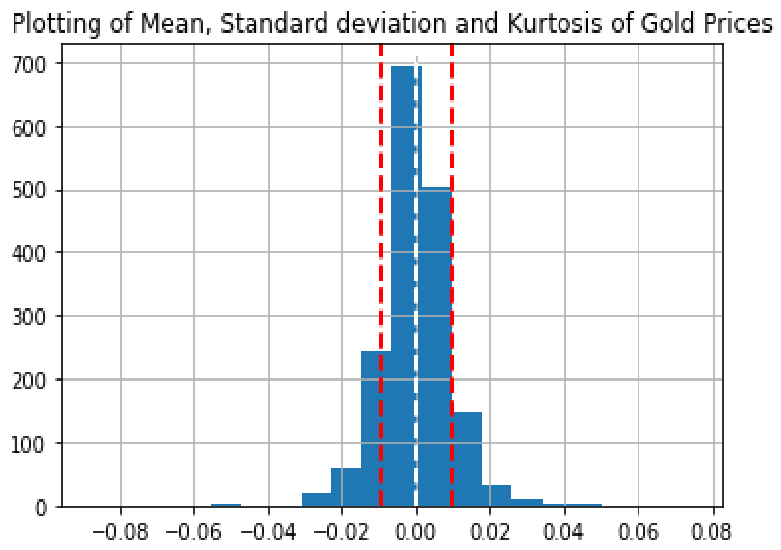
Positive Kurtosis means more weights in the tail

Negative Kurtosis It has as much data in each tail as it does in the peak.

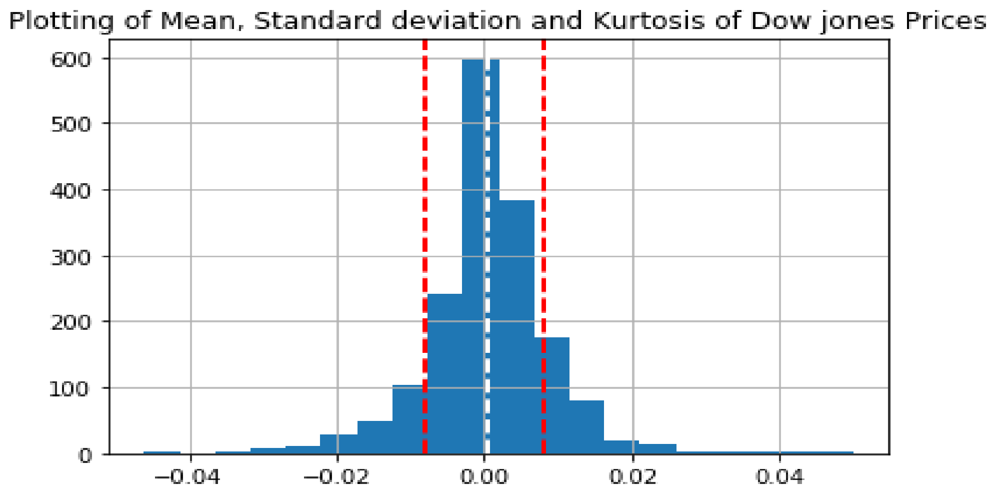
```
Mean= 0.0005366024364688845
Standard Deviation= 0.008262309911393529
Kurtosis= 3.4557859039745225
```



```
Mean= -8.656986121281953e-05
Standard Deviation= 0.009611536167006395
Kurtosis= 8.60658492491834
```



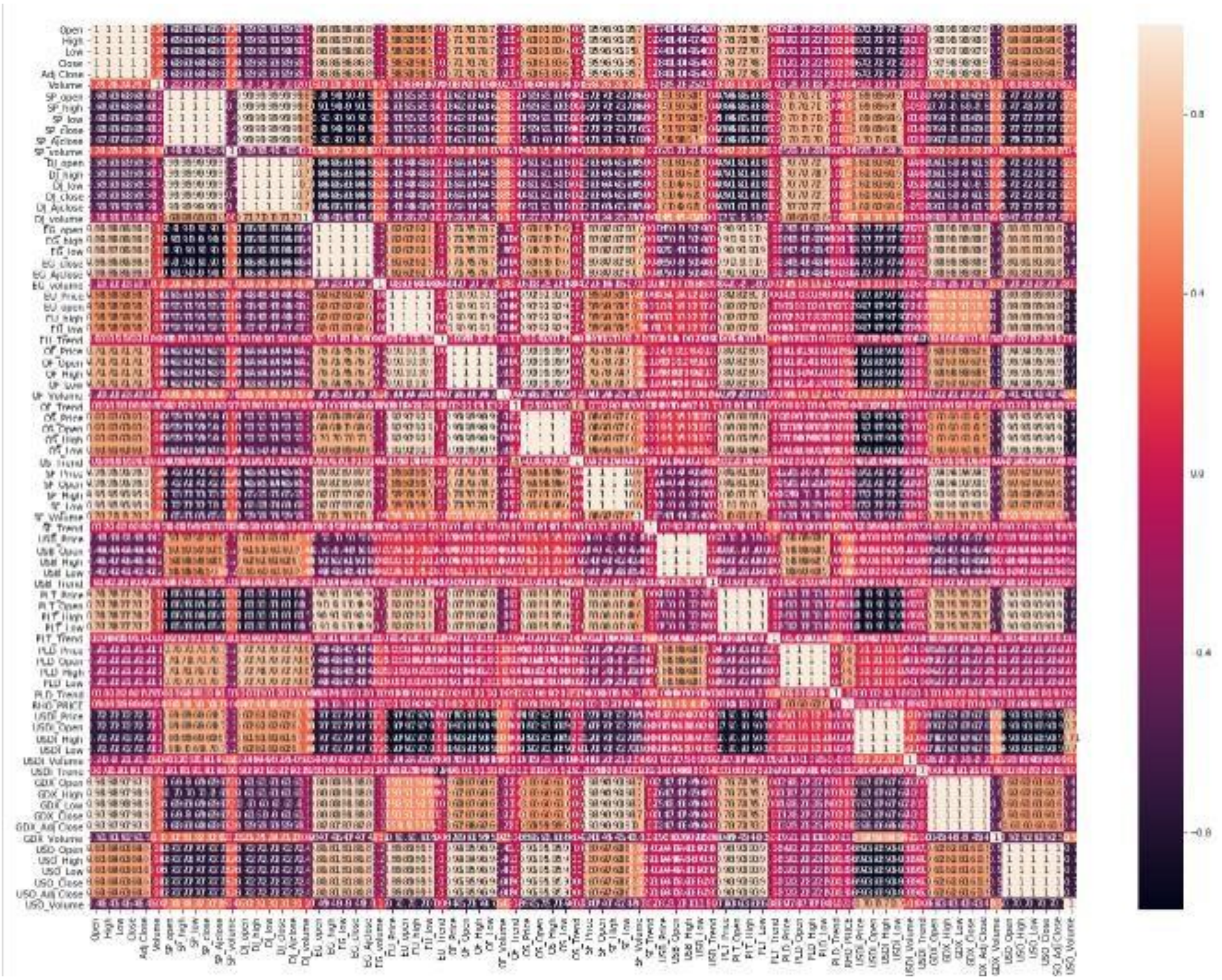
```
Mean= 0.00042663952187518026
Standard Deviation= 0.00815178011451231
Kurtosis= 3.832719336260693
```



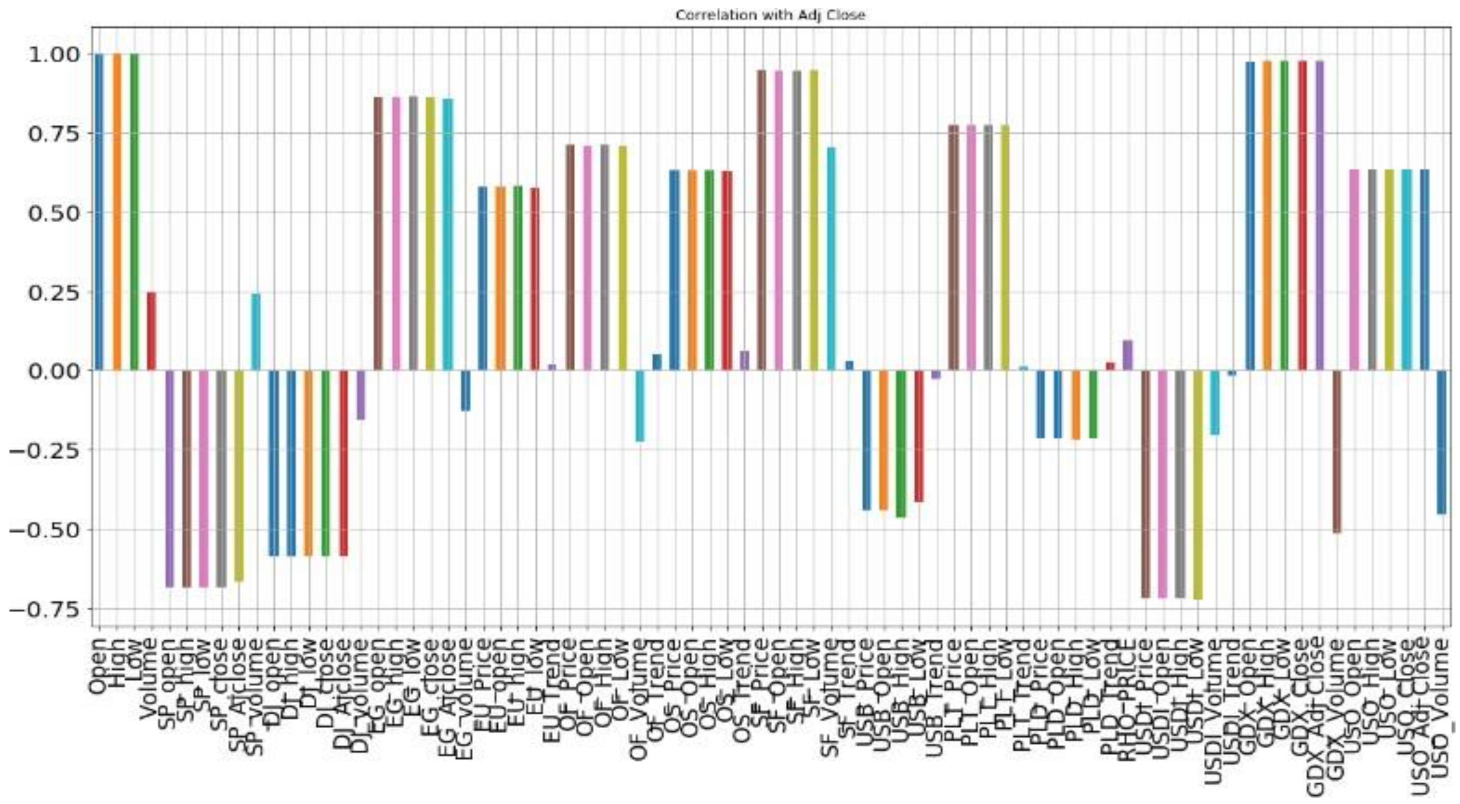
Correlation Analysis

In this step we will explore which feature variables are highly correlated with target variable which Adjusted Close price.

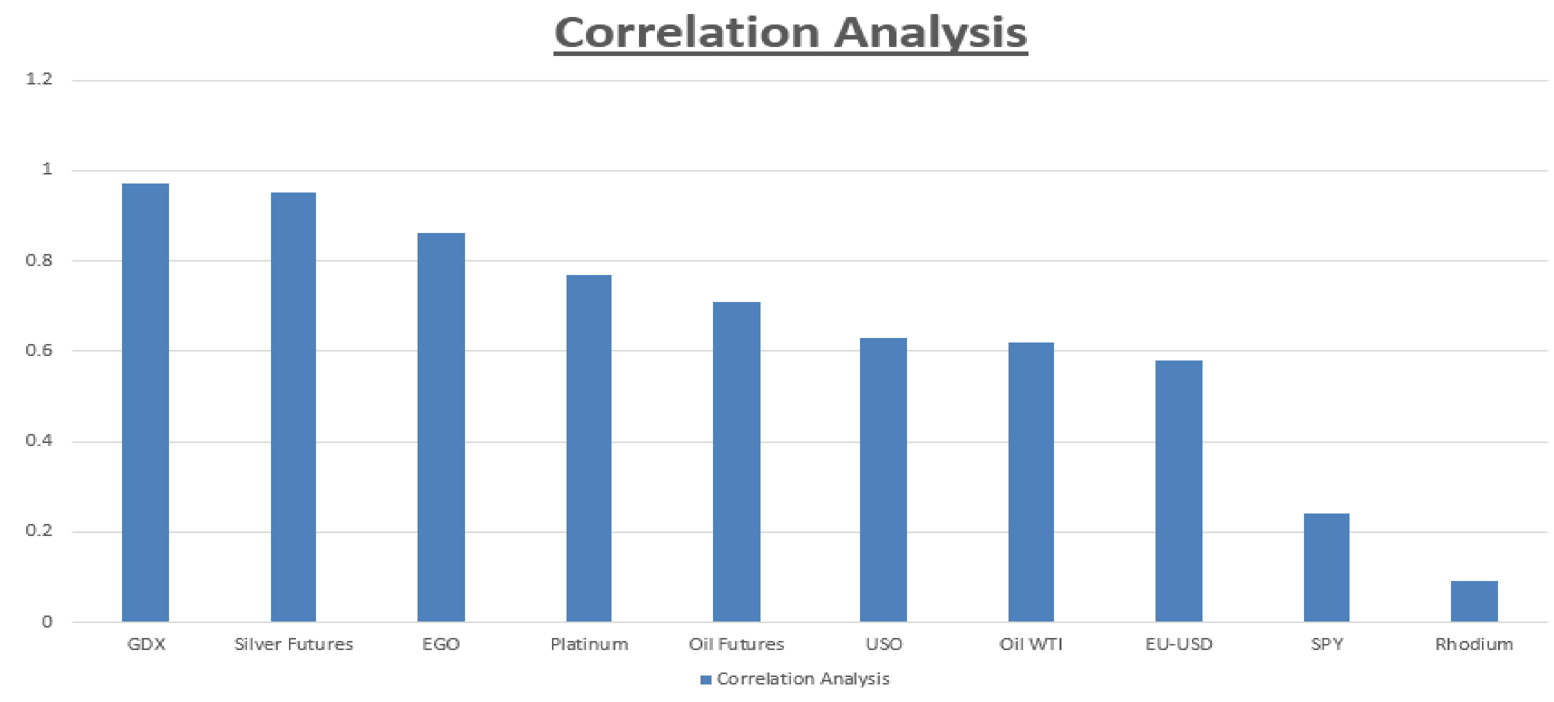
First we have plotted correlation matrix. As there are so many features in the dataset so it is very difficult to visibly observe the correlation among variables.



Next we will calculate and plot correlation bar graph taking all the feature variables against target variable. Values in the range of -1 to +1 where negative value indicates that particular feature is negatively correlated with target variable and positive value indicates that feature is positively correlated with target variable and 0 value indicates no correlation exist between that feature and target variable.



Below image shows the summarized correlation between different features and target variable:



Algorithms & Techniques

The goal of this project is to study time-series data and explore as many options as possible to accurately predict the Gold rates in future.

We would use TimeSeriesSplit function in scikit-learn to split the data into training set and testing set, it could split the whole dataset into several packs and in each packs, the indices of testing set would be higher than training set. By doing this can prevent look ahead bias, which means the model would not use future data to train itself.

I separate the last **90 days** data as the validation dataset, to test the models by the data they have never seen.

We have used different machine learning algorithms to see if they are able to accurately predict Gold prices.I have used 7 different machine learning algorithms i.e, Support Vector Regressor (SVR), Random Forest Regressor, Lasso CV, Ridge CV, Bayesian Ridge, Gradient Boosting Regressor and Stochastic Gradient Descent (SGD) and then we ensemble top three best performing algorithms and compare their performance with other algorithms.

1. **Support Vector Regressor (SVR):**

The model produced by Support Vector Regression depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction.

I have used SVR with (**kernel='linear'**)

2. **Random Forest Regressor:**

Random Forest is a supervised learning algorithm.It creates a forest and makes it somehow random. The forest it builds, is an **ensemble** of Decision Trees, most of the time trained with the "**bagging**" method. The general idea of the bagging method is that a combination of learning models increases the overall result.

Random decision forests correct for decision trees habit of **overfitting** to their training set.

I have used two parameters **n_estimators=50 (default value =10)**, the number of trees in the forest**.** and **random_state=0,** random_state is the seed used by the random number generator.

3. **Lasso CV:**

The Lasso is a linear model that estimates sparse coefficients. It is useful in some contexts due to its tendency to prefer solutions with fewer parameter values, effectively reducing the number of variables upon which the given solution is dependent.

- Performs **L1 regularization**, i.e. adds penalty equivalent to absolute value of the magnitude of coefficients
- Minimization objective = **LS Obj + α * (sum of absolute value of coefficients)**

Note that here '**LS Obj**' refers to 'least squares objective', i.e. the linear regression objective without regularization.

I have used three parameters **n_alphas=1000**, Number of alphas along the regularization path **max_iter=3000**, the maximum number of iterations and **random_state=0.**

Here, the alpha parameter controls the degree of sparsity of the coefficients estimated.

4. **Ridge CV:**

Ridge regression addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of coefficients. The ridge coefficients minimize a penalized residual sum of squares,

$$\min_{w} ||Xw - y||_2^2 + \alpha ||w||_2^2$$

Here, α≥0 is a complexity parameter that controls the amount of shrinkage: the larger the value of α, the greater the amount of shrinkage and thus the coefficients become more robust to collinearity.

I have used the parameter **gcv_mode='auto',** flag indicating which strategy to use when performing Generalized Cross-Validation.

5. **Bayesian Ridge:**

Bayesian regression techniques can be used to include regularization parameters in the estimation procedure: the regularization parameter is not set in a hard sense but tuned to the data at hand.

I have used default parameters in this algorithm.

6. **Gradient Boosting Regressor:**

Gradient Tree Boosting or Gradient Boosted Regression Trees (GBRT) is a generalization of boosting to arbitrary differentiable loss functions.

GradientBoostingRegressor supports a number of different loss functions for regression which can be specified via the argument loss; the default loss function for regression is least squares ('ls').

I have used following parameters:
n_estimators=70, the number of boosting stages to perform**. learning_rate=0.1,** learning rate shrinks the contribution of each tree by learning_rate, **max_depth=4,** the maximum depth limits the number of nodes in the tree**, random_state=0, loss='ls',** 'ls' refers to least squares regression.

7. **Stochastic Gradient Descent (SGD):**

SGD stands for Stochastic Gradient Descent: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate).

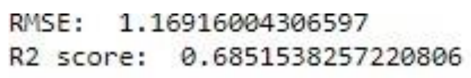
The regularizer is a penalty added to the loss function that shrinks model parameters towards the zero vector using either the squared euclidean norm L2 or the absolute norm L1 or a combination of both (Elastic Net).

I have used following parameters :

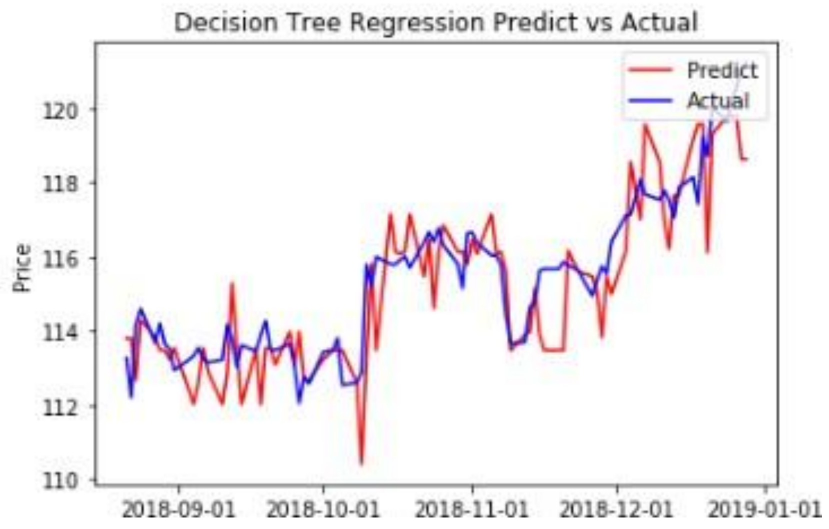
max_iter=1000, the maximum number of passes over the training data (aka epochs).
tol=1e-3, the stopping criterion. If it is not None, the iterations will stop when (loss > previous_loss - tol)**,**
loss='squared_epsilon_insensitive', 'epsilon_insensitive' ignores errors less than epsilon and is linear past that; this is the loss function used in SVR. 'squared_epsilon_insensitive' is the same but becomes squared loss past a tolerance of epsilon.
penalty='l1', the penalty (aka regularization term) to be used**,**
alpha=0.1, Constant that multiplies the regularization term.

Benchmark Model

I would choose **Decision Tree Regressor** and use default arguments as the benchmark model. I would use the same data and features in my solution model detailed above. The result of the benchmark model on validation set is shown below :



```
RMSE:  1.16916004306597
R2 score:  0.6851538257220806
```



From the above result we can see that the model's performance is somewhat good actually on validation set, the model somewhat predicted the prices well. Next we will see the performance of our solution models.

III. Methodology

Data Preprocessing

We had normalized the data using sklearn's MinMaxScaler Function.It transformed the data in the range of 0 to 1.

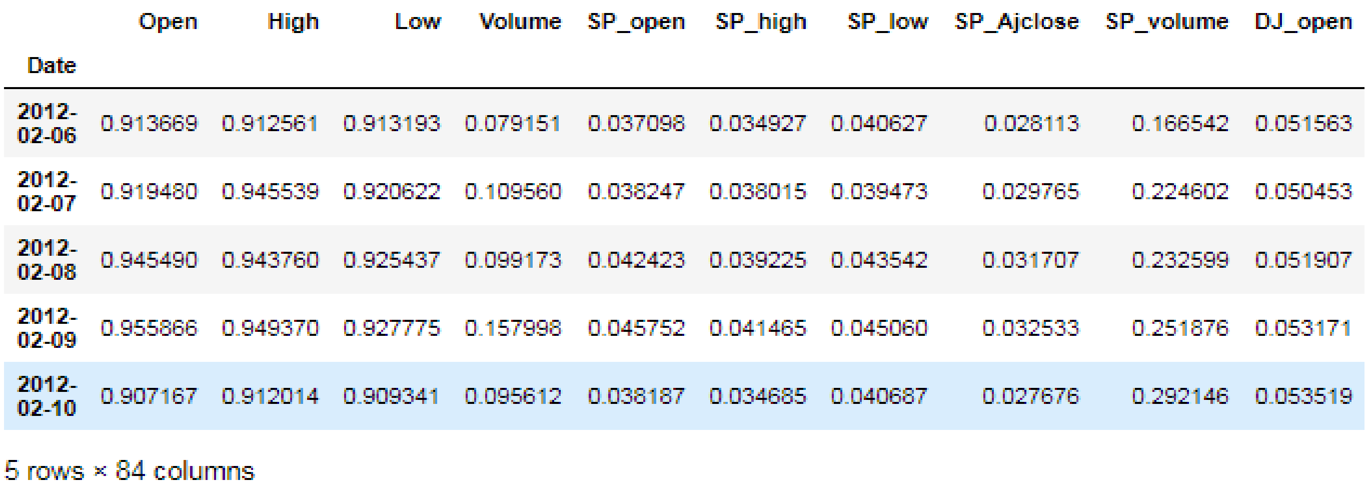
The transformation is given by:

X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))

X_scaled = X_std * (max - min) + min

where min, max = feature_range.

The below image represents the snapshot of the normalized data.

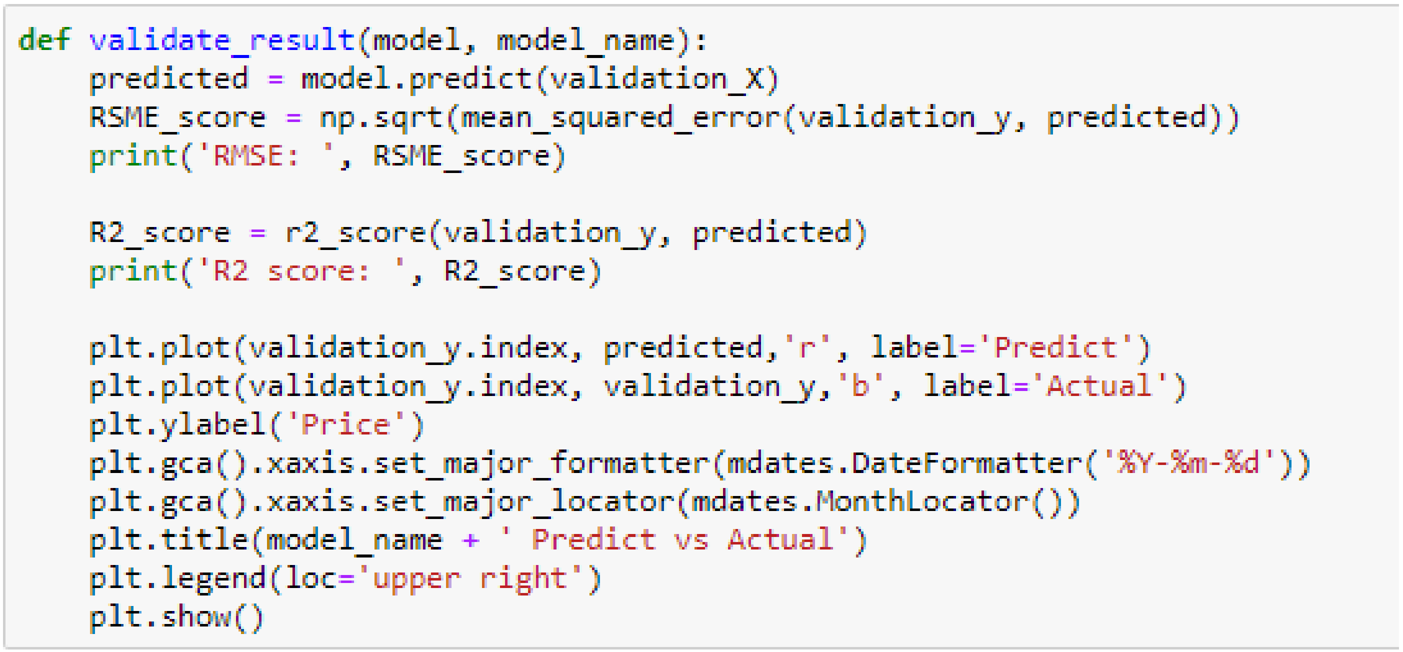


	Open	High	Low	Volume	SP_open	SP_high	SP_low	SP_Ajclose	SP_volume	DJ_open
Date										
2012-02-06	0.913669	0.912561	0.913193	0.079151	0.037098	0.034927	0.040627	0.028113	0.166542	0.051563
2012-02-07	0.919480	0.945539	0.920622	0.109560	0.038247	0.038015	0.039473	0.029765	0.224602	0.050453
2012-02-08	0.945490	0.943760	0.925437	0.099173	0.042423	0.039225	0.043542	0.031707	0.232599	0.051907
2012-02-09	0.955866	0.949370	0.927775	0.157998	0.045752	0.041465	0.045060	0.032533	0.251876	0.053171
2012-02-10	0.907167	0.912014	0.909341	0.095612	0.038187	0.034685	0.040687	0.027676	0.292146	0.053519

5 rows × 84 columns

Implementation

I have developed a function for validation of models.



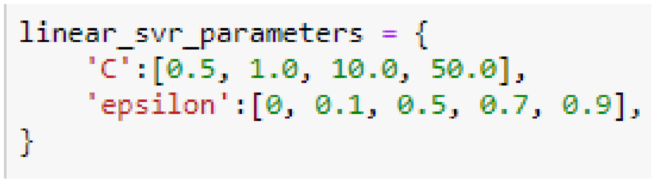
```
def validate_result(model, model_name):
    predicted = model.predict(validation_X)
    RSME_score = np.sqrt(mean_squared_error(validation_y, predicted))
    print('RMSE: ', RSME_score)

    R2_score = r2_score(validation_y, predicted)
    print('R2 score: ', R2_score)

    plt.plot(validation_y.index, predicted,'r', label='Predict')
    plt.plot(validation_y.index, validation_y,'b', label='Actual')
    plt.ylabel('Price')
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
    plt.gca().xaxis.set_major_locator(mdates.MonthLocator())
    plt.title(model_name + ' Predict vs Actual')
    plt.legend(loc='upper right')
    plt.show()
```

validate_result take a model and the model's name as input, it would use validation set to see the performance of model, and output the RMSE error and R2 score and also plot the results.

We train our solution models, and use **GridSearchCV** to adjust the parameters. Below are the performances of models, their parameters and the ranking of **RMSE** error.

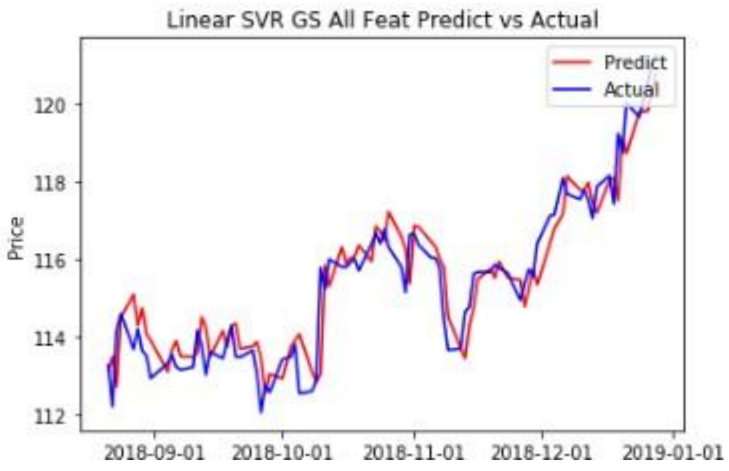
Support Vector Regressor

The result of default parameter is not ideal, it didn't learn well how to predict price. So, I have tried to tune the hyper parameters using **GridSearchCV** to adjust the parameters to see whether we could improve the model performance. The parameters I have tuned using Gridsearch is shown below.



```
linear_svr_parameters = {
    'C':[0.5, 1.0, 10.0, 50.0],
    'epsilon':[0, 0.1, 0.5, 0.7, 0.9],
}
```

```
RMSE:   0.7416902627042532
R2 score:   0.8732944477456932
```



Result after gridsearch improves a lot and it has predicted the prices well.

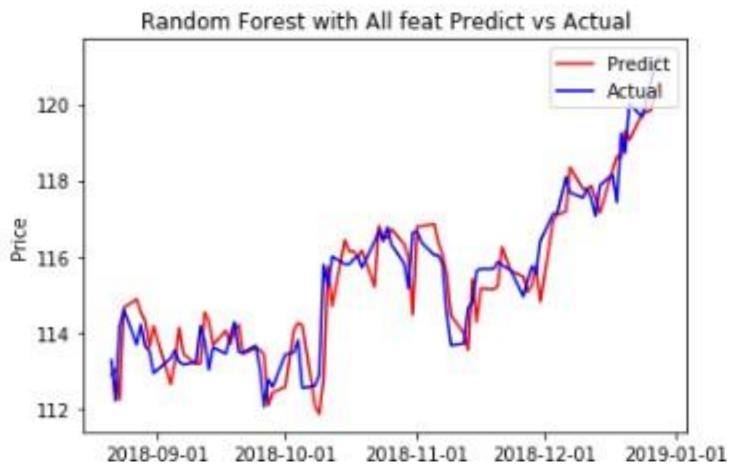
Comparison of Default Parameter SVR and Hyper parameter tuned SVR

Eval Metric	**SVR (default parameters)**	**Tuned SVR**
RMSE	0.8136	0.7416
R2 Score	0.8474	0.8732

Random Forest Regressor

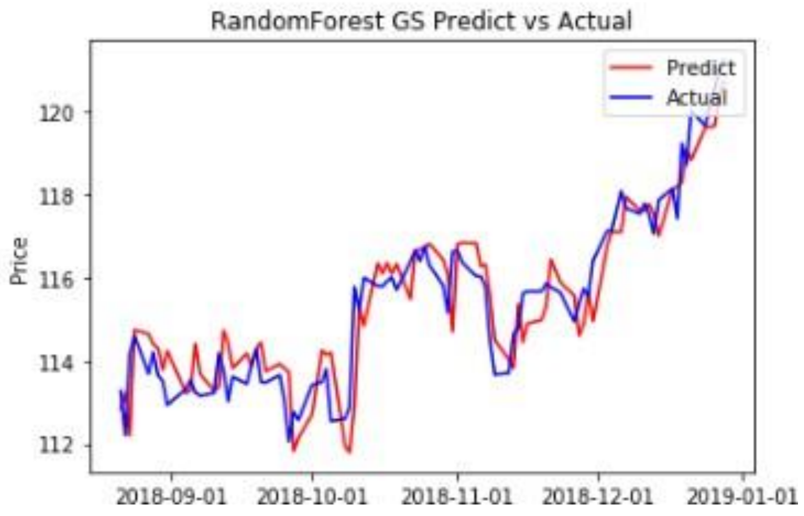
After applying random forest regressor with **n_estimators =50**, we have achieved following result :

```
RMSE:   0.809652076705479
R2 score:   0.8490102869916533
```



After applying random Forest regressor with default parameters we have tried to tune the hyper parameters using GridsearchCV to see if the performance would improve.Below is the result after applying GridsearchCV on Random forest.

```
RMSE:   0.8209975294625336
R2 score:   0.8447490766439336
```



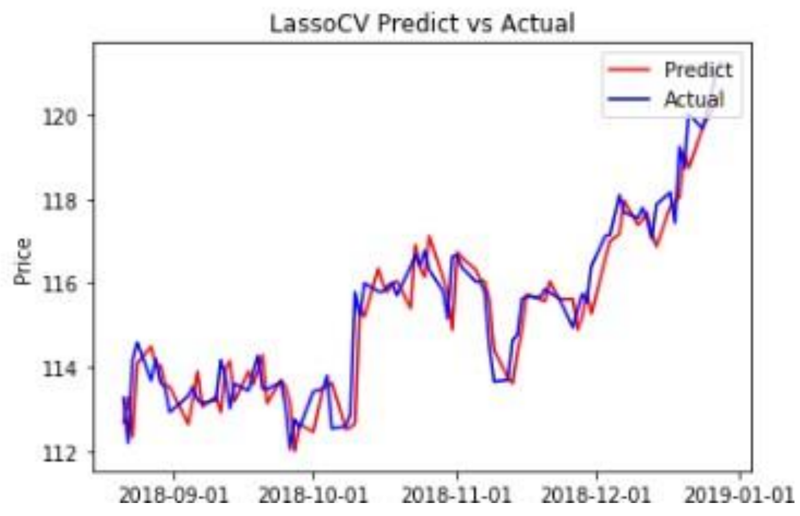
Comparison of Default Parameter SVR and Hyper parameter tuned SVR

Eval Metric	**Random Forest (Default Parameters)**	**Random forest after GridsearchCV**
RMSE	0.8096	0.8209
R2 Score	0.8490	0.8447

Lasso CV

By default, Lasso CV performs Cross-Validation, so I have tried to tune some parameters i.e., n_alpha = 1000 perform better, and set the max_iter = 3000, because if the max_iter value is low, the model would reach the limit before convergence.

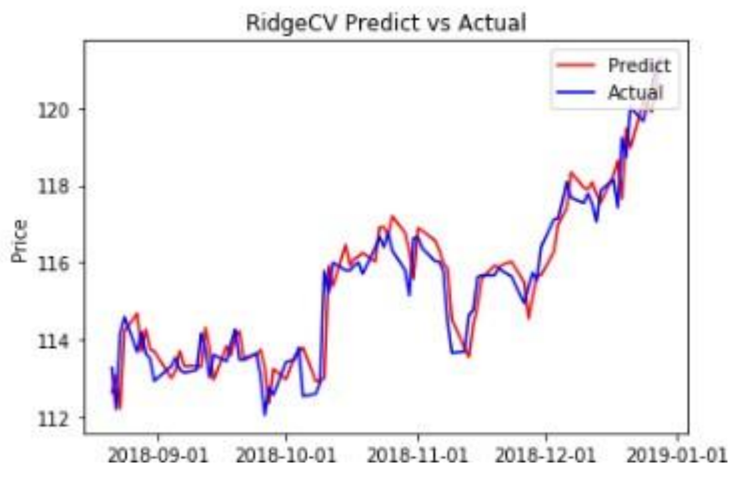
```
RMSE:  0.7159068365647187
R2 score:  0.8819506742756609
```



RidgeCV

It performs generalized cross validation. I have used the parameter **gcv_mode='auto',** flag indicating which strategy to use when performing Generalized Cross-Validation.

```
RMSE:  0.7186272522528762
R2 score:  0.8810518047954405
```

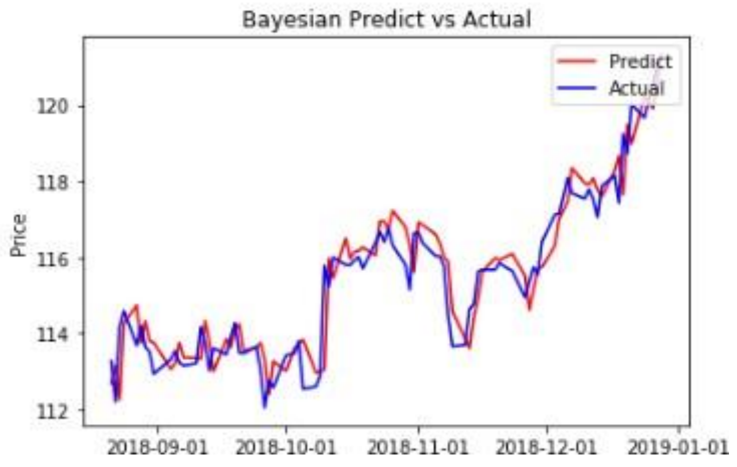


Bayesian Ridge

Bayesian regression techniques can be used to include regularization parameters in the estimation procedure: the regularization parameter is not set in a hard sense but tuned to the data at hand.

I have used default parameters in this algorithm.

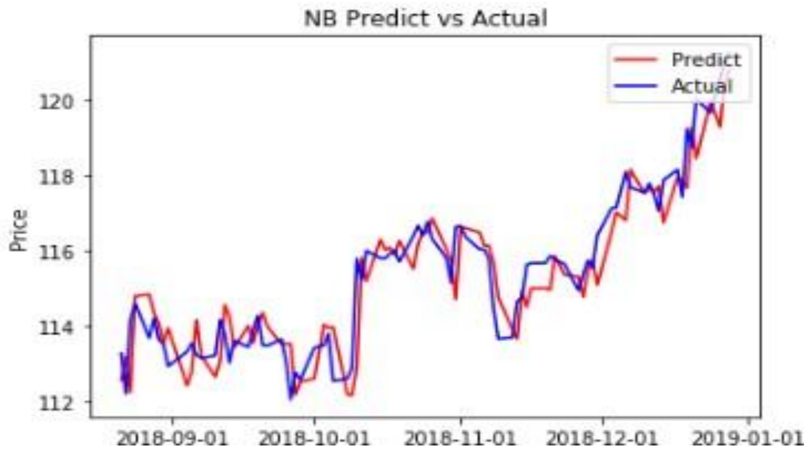
```
RMSE:  0.7195639481910471
R2 score:  0.8807415162414403
```



Gradient Boosting Regressor

Gradient Tree Boosting or Gradient Boosted Regression Trees (GBRT) is a generalization of boosting to arbitrary differentiable loss functions.
I have used following parameters:
n_estimators=70, learning_rate=0.1, max_depth=4, random_state=0, loss='ls', 'ls' refers to least squares regression.

```
RMSE:  0.8094931831292773
R2 score:  0.8490695443986888
```



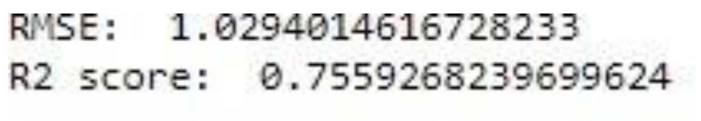
<u>SGD</u>

SGD stands for Stochastic Gradient Descent: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate).

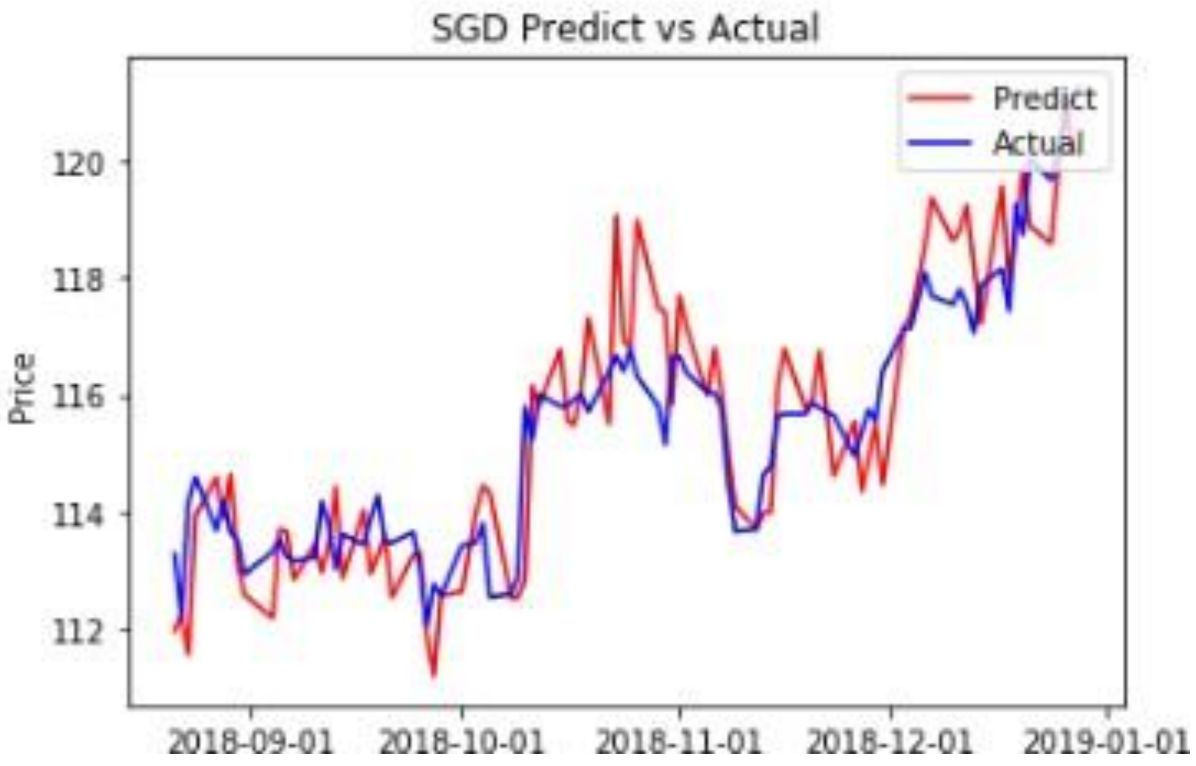
The regularizer is a penalty added to the loss function that shrinks model parameters towards the zero vector using either the squared euclidean norm L2 or the absolute norm L1 or a combination of both (Elastic Net).

I have used following parameters :

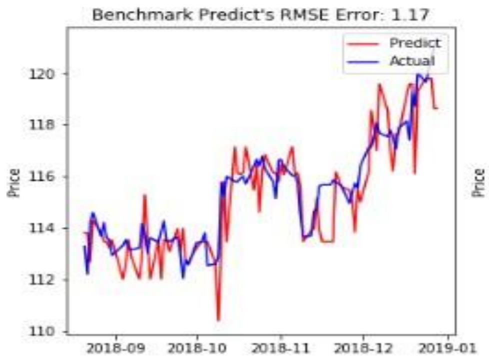
max_iter=1000, tol=1e-3, loss='squared_epsilon_insensitive', penalty='l1' and **alpha=0.1**

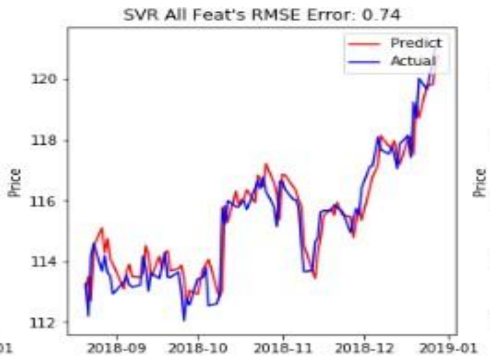


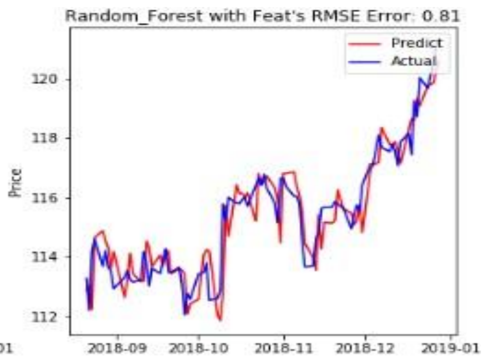
```
RMSE:  1.0294014616728233
R2 score:  0.7559268239699624
```

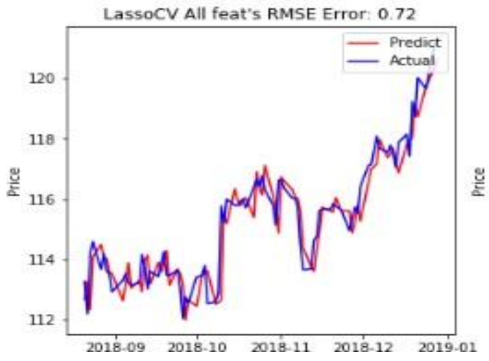


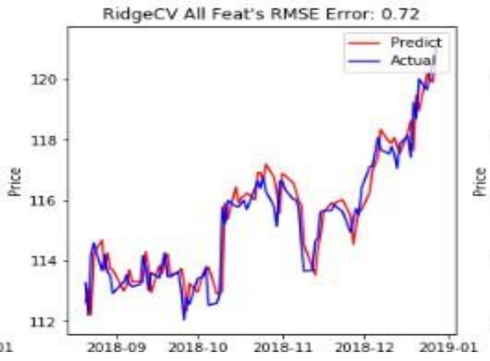
Review of Benchmark & Solution Models

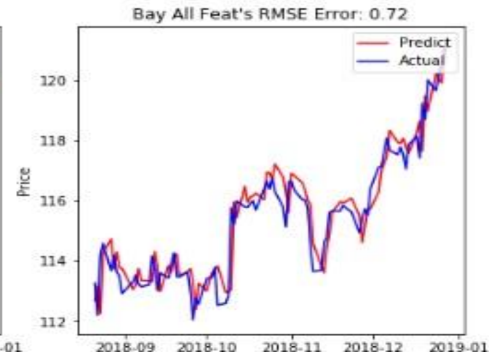


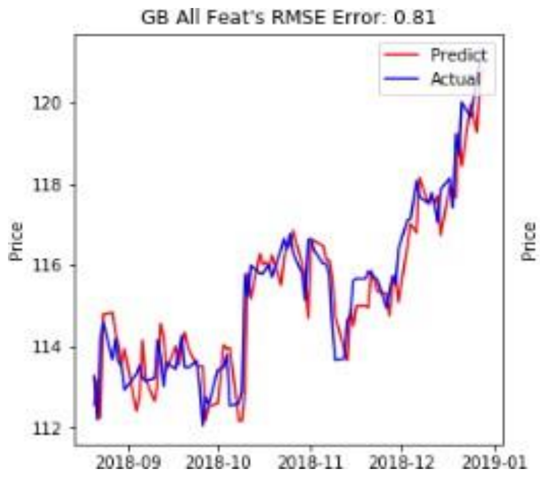


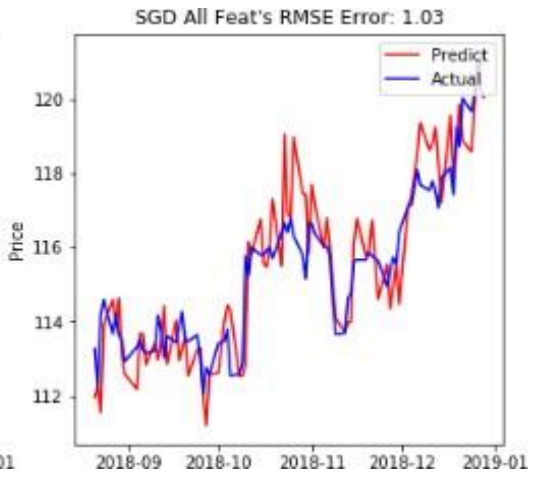




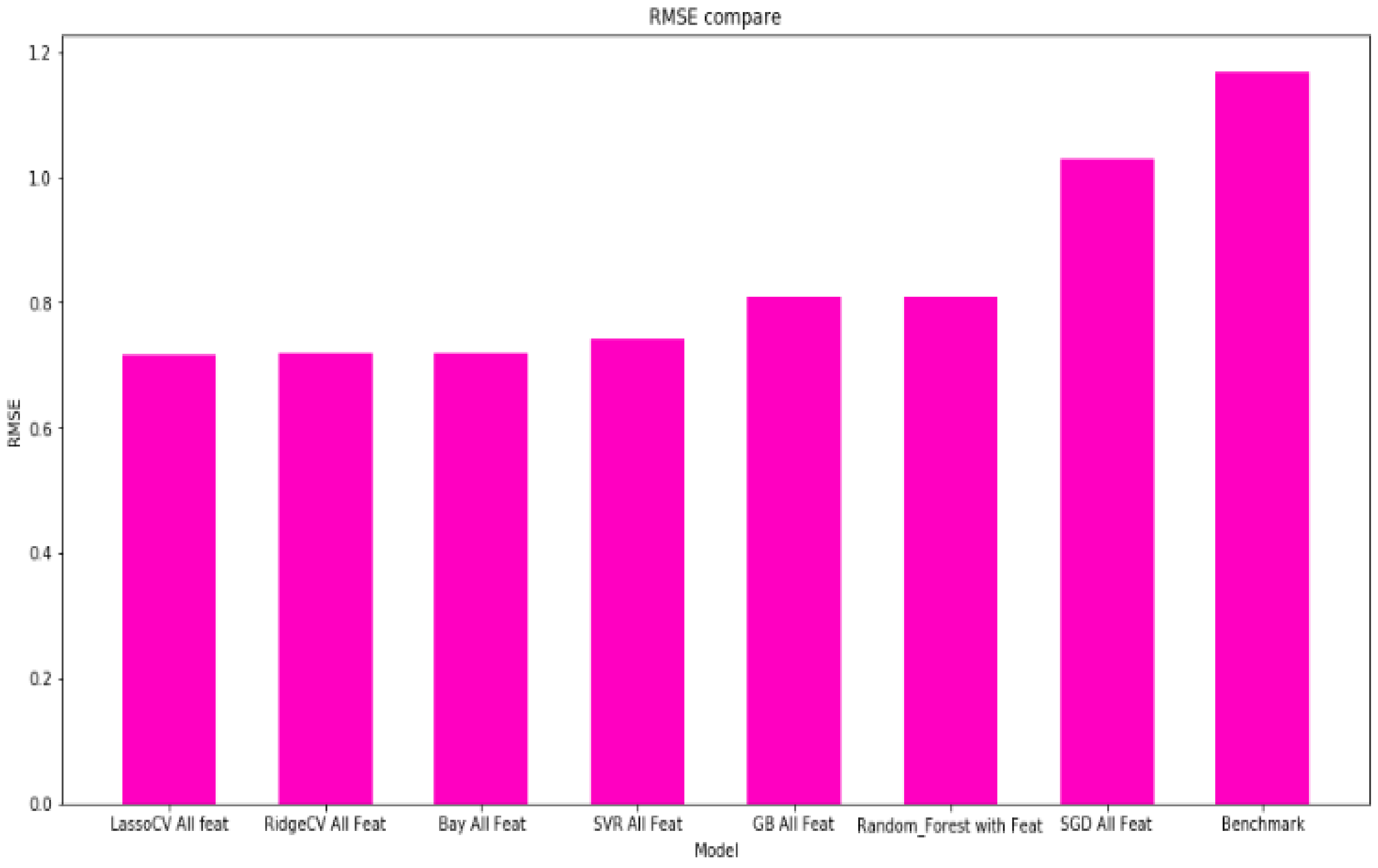








Comparison of RMSE of Benchmark & Solution models



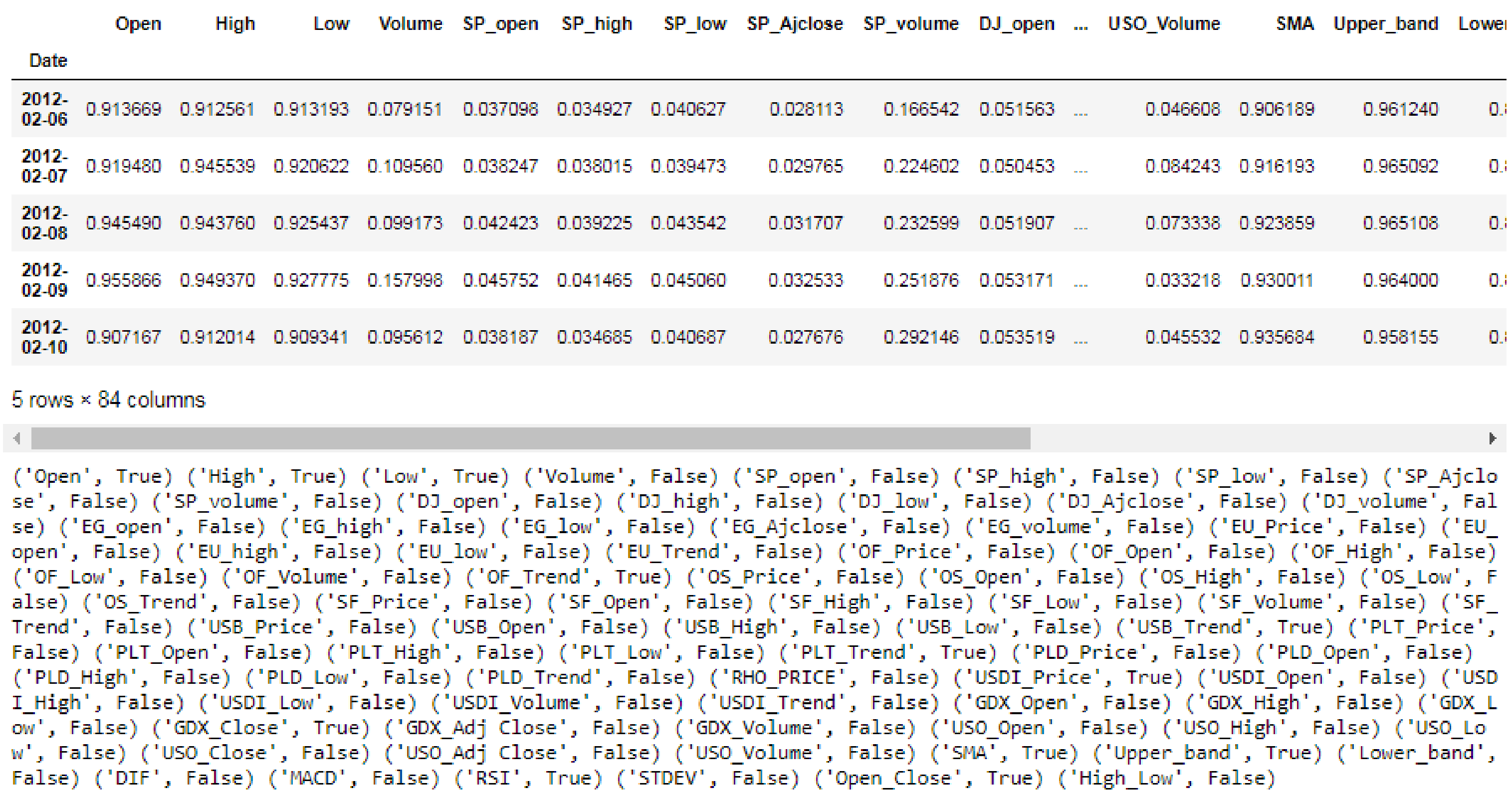
We can see that the RSME errors of models are close except SGD and benchmark model which is decision tree regressor, Benchmark model has highest RMSE error next SGD has second highest RMSE error whereas LassoCV, RidgeCV and Bayesian Ridge has nearly same performance having RMSE error in the range of 0.70 to 0.72.

Refinement

I have choosen sklearn's **SelectFromModel** for feature selection. It selects weights based on feature importance weights so we have to input those model in **SelectFromModel** which has feature importance attribute. In our case I have choosen LassoCV because it has feature importance attribute and it has the best performance in terms of RMSE.

So, I have kept only those features who has greater importance while others are discarded.

Below is the snapshot of the image showing feature supporting attributes in terms of true or false.

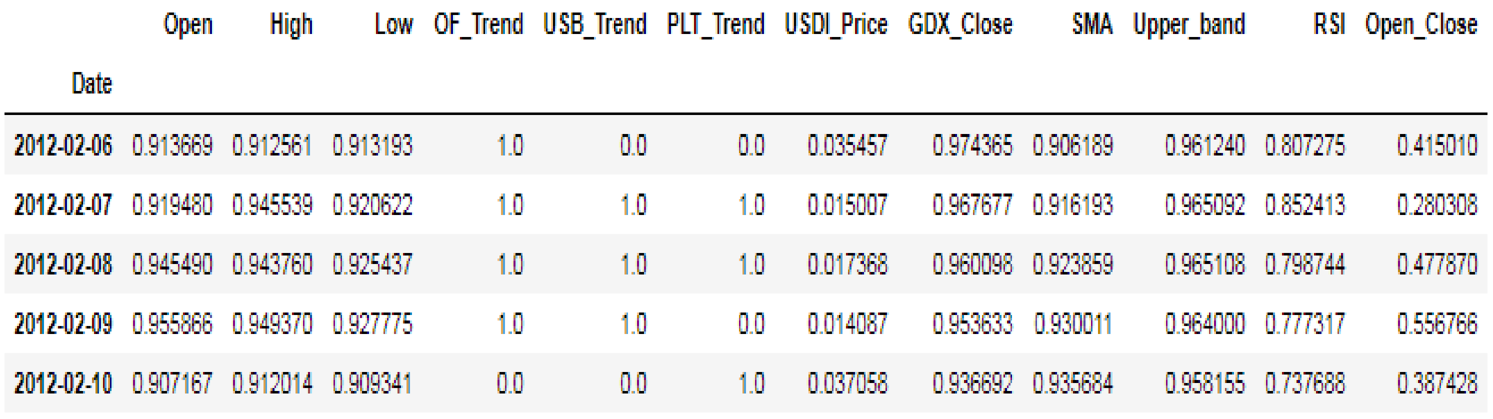


	Open	High	Low	Volume	SP_open	SP_high	SP_low	SP_Ajclose	SP_volume	DJ_open	...	USO_Volume	SMA	Upper_band	Lowe
Date															
2012-02-06	0.913669	0.912561	0.913193	0.079151	0.037098	0.034927	0.040627	0.028113	0.166542	0.051563	...	0.046608	0.906189	0.961240	0.
2012-02-07	0.919480	0.945539	0.920622	0.109560	0.038247	0.038015	0.039473	0.029765	0.224602	0.050453	...	0.084243	0.916193	0.965092	0.
2012-02-08	0.945490	0.943760	0.925437	0.099173	0.042423	0.039225	0.043542	0.031707	0.232599	0.051907	...	0.073338	0.923859	0.965108	0.
2012-02-09	0.955866	0.949370	0.927775	0.157998	0.045752	0.041465	0.045060	0.032533	0.251876	0.053171	...	0.033218	0.930011	0.964000	0.
2012-02-10	0.907167	0.912014	0.909341	0.095612	0.038187	0.034685	0.040687	0.027676	0.292146	0.053519	...	0.045532	0.935684	0.958155	0.

5 rows × 84 columns

```
('Open', True) ('High', True) ('Low', True) ('Volume', False) ('SP_open', False) ('SP_high', False) ('SP_low', False) ('SP_Ajclo
se', False) ('SP_volume', False) ('DJ_open', False) ('DJ_high', False) ('DJ_low', False) ('DJ_Ajclose', False) ('DJ_volume', Fal
se) ('EG_open', False) ('EG_high', False) ('EG_low', False) ('EG_Ajclose', False) ('EG_volume', False) ('EU_Price', False) ('EU_
open', False) ('EU_high', False) ('EU_low', False) ('EU_Trend', False) ('OF_Price', False) ('OF_Open', False) ('OF_High', False)
('OF_Low', False) ('OF_Volume', False) ('OF_Trend', True) ('OS_Price', False) ('OS_Open', False) ('OS_High', False) ('OS_Low', F
alse) ('OS_Trend', False) ('SF_Price', False) ('SF_Open', False) ('SF_High', False) ('SF_Low', False) ('SF_Volume', False) ('SF_
Trend', False) ('USB_Price', False) ('USB_Open', False) ('USB_High', False) ('USB_Low', False) ('USB_Trend', True) ('PLT_Price',
False) ('PLT_Open', False) ('PLT_High', False) ('PLT_Low', False) ('PLT_Trend', True) ('PLD_Price', False) ('PLD_Open', False)
('PLD_High', False) ('PLD_Low', False) ('PLD_Trend', False) ('RHO_PRICE', False) ('USDI_Price', True) ('USDI_Open', False) ('USD
I_High', False) ('USDI_Low', False) ('USDI_Volume', False) ('USDI_Trend', False) ('GDX_Open', False) ('GDX_High', False) ('GDX_L
ow', False) ('GDX_Close', True) ('GDX_Adj Close', False) ('GDX_Volume', False) ('USO_Open', False) ('USO_High', False) ('USO_Lo
w', False) ('USO_Close', False) ('USO_Adj Close', False) ('USO_Volume', False) ('SMA', True) ('Upper_band', True) ('Lower_band',
False) ('DIF', False) ('MACD', False) ('RSI', True) ('STDEV', False) ('Open_Close', True) ('High_Low', False)
```

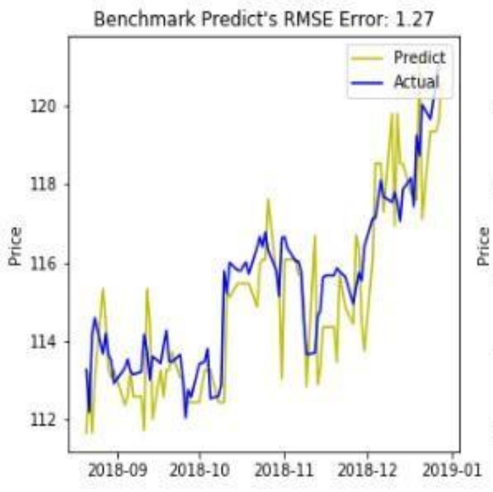
So we will create a new dataset with selected features and which looks like as below snapshot.

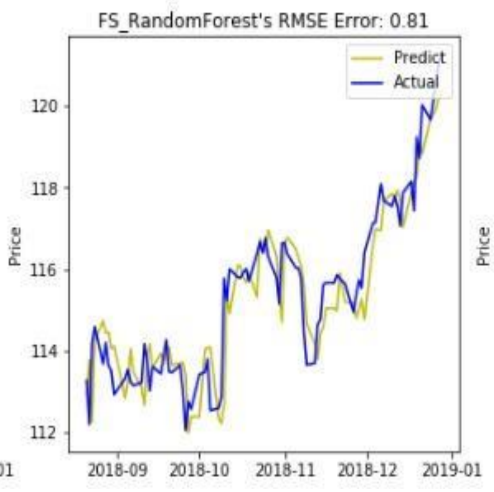


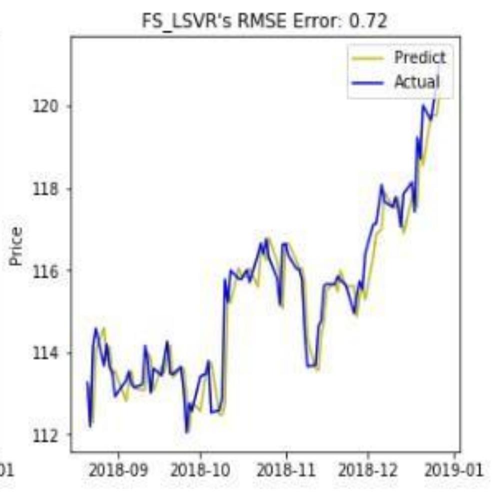
	Open	High	Low	OF_Trend	USB_Trend	PLT_Trend	USDI_Price	GDX_Close	SMA	Upper_band	RSI	Open_Close
Date												
2012-02-06	0.913669	0.912561	0.913193	1.0	0.0	0.0	0.035457	0.974365	0.906189	0.961240	0.807275	0.415010
2012-02-07	0.919480	0.945539	0.920622	1.0	1.0	1.0	0.015007	0.967677	0.916193	0.965092	0.852413	0.280308
2012-02-08	0.945490	0.943760	0.925437	1.0	1.0	1.0	0.017368	0.960098	0.923859	0.965108	0.798744	0.477870
2012-02-09	0.955866	0.949370	0.927775	1.0	1.0	0.0	0.014087	0.953633	0.930011	0.964000	0.777317	0.556766
2012-02-10	0.907167	0.912014	0.909341	0.0	0.0	1.0	0.037058	0.936692	0.935684	0.958155	0.737688	0.387428

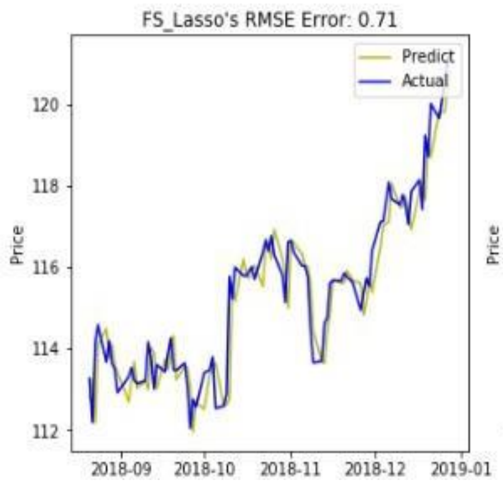
After creating new dataset with selected features we will train benchmark and all the solution models and validate it by selected features dataset to see the performance in comparison to original features dataset.

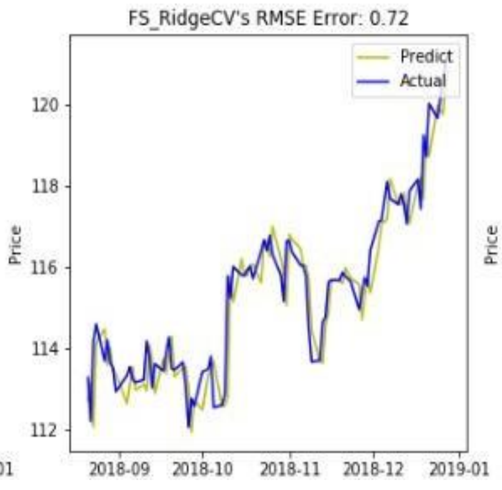
Review of Benchmark & Solution Models

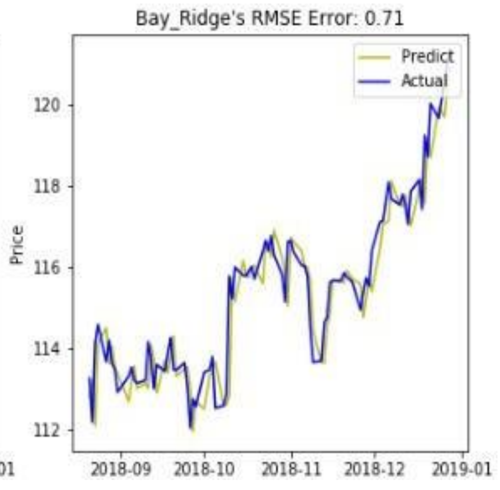


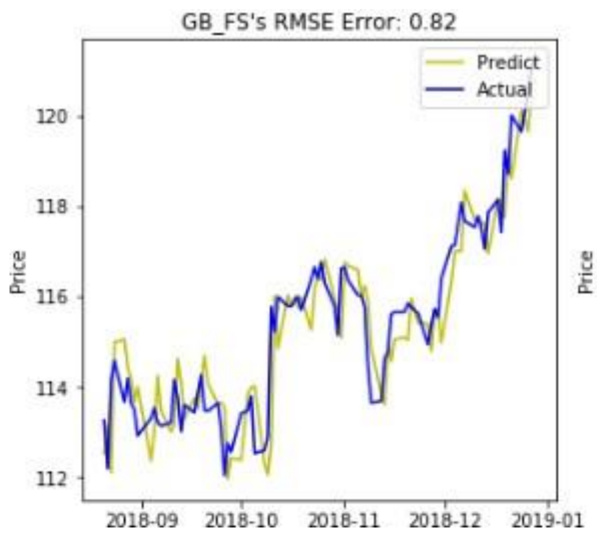


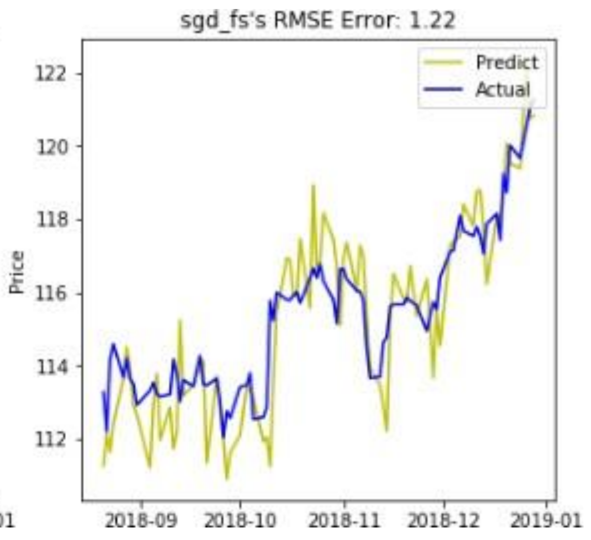




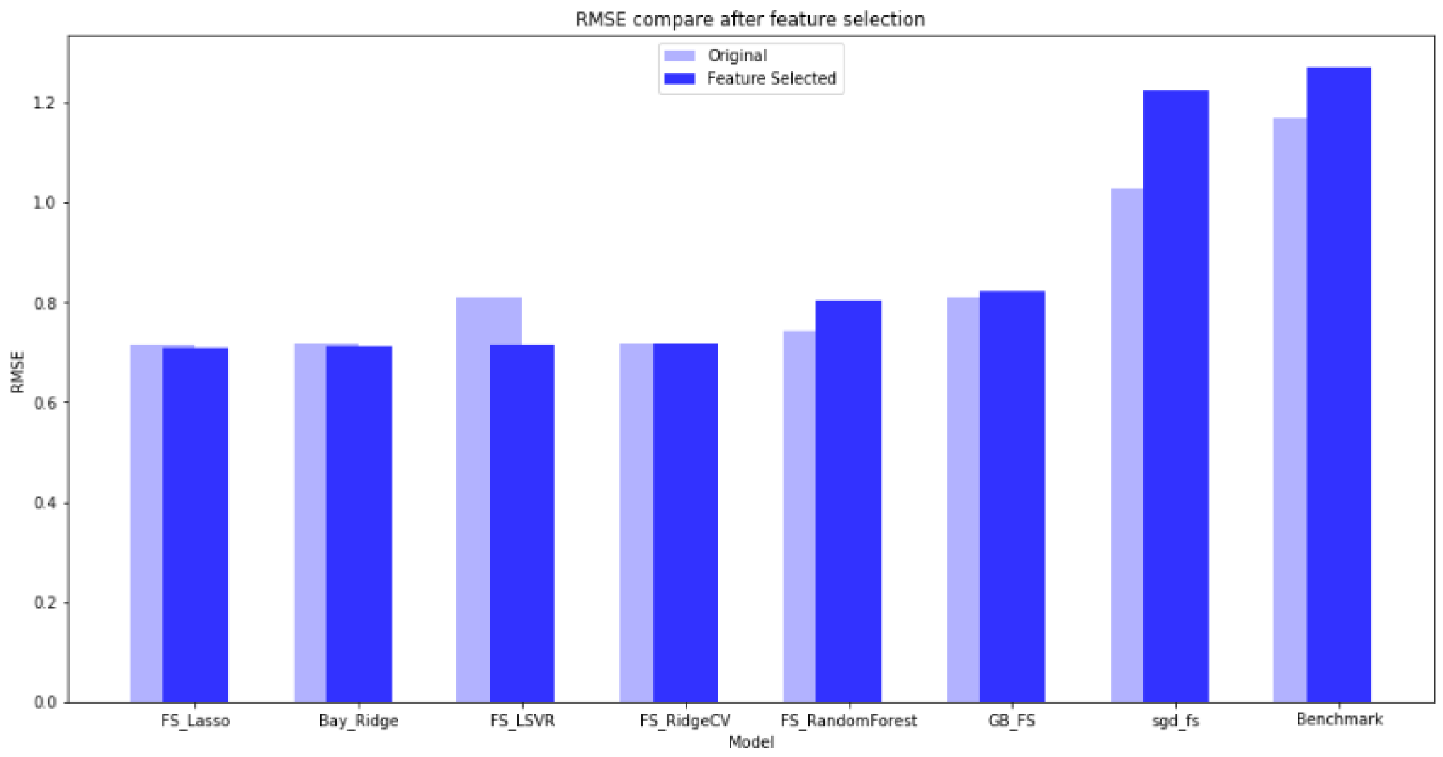








Comparison of RMSE of Feature selected & Original Feature Models



As we have seen from the above plot 3 Feature selected models performs better in RMSE error reduction and Feature selected Linear SVR is the best as it has RMSE of 0.716 in feature selected model and 0.741 with all features model. Also Lasso cv and Bayesian Ridge performs slightly better from original features model whereas Ridge cv shows no improvement from features model whereas four model performance degrades after feature selection in which benchmark model has highest RMSE error and SGD model degrades most in comparison to others.

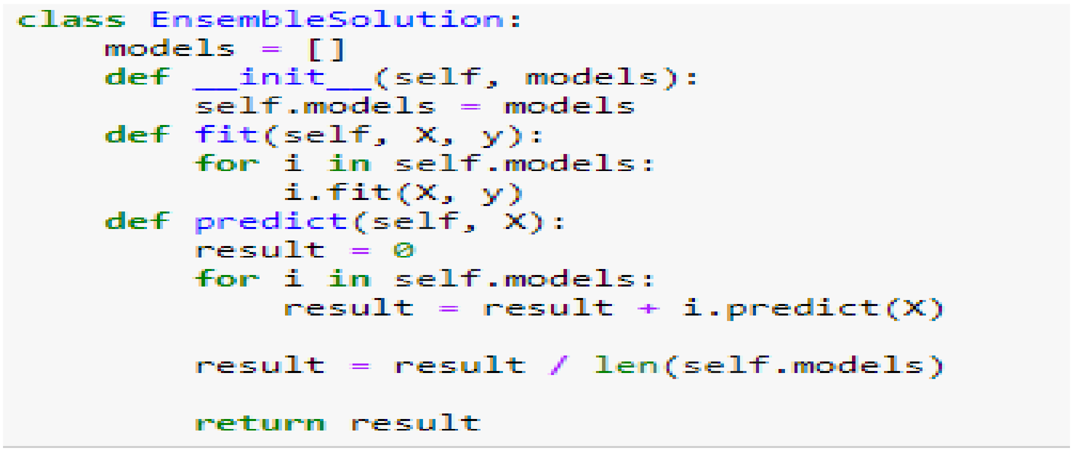
IV Results

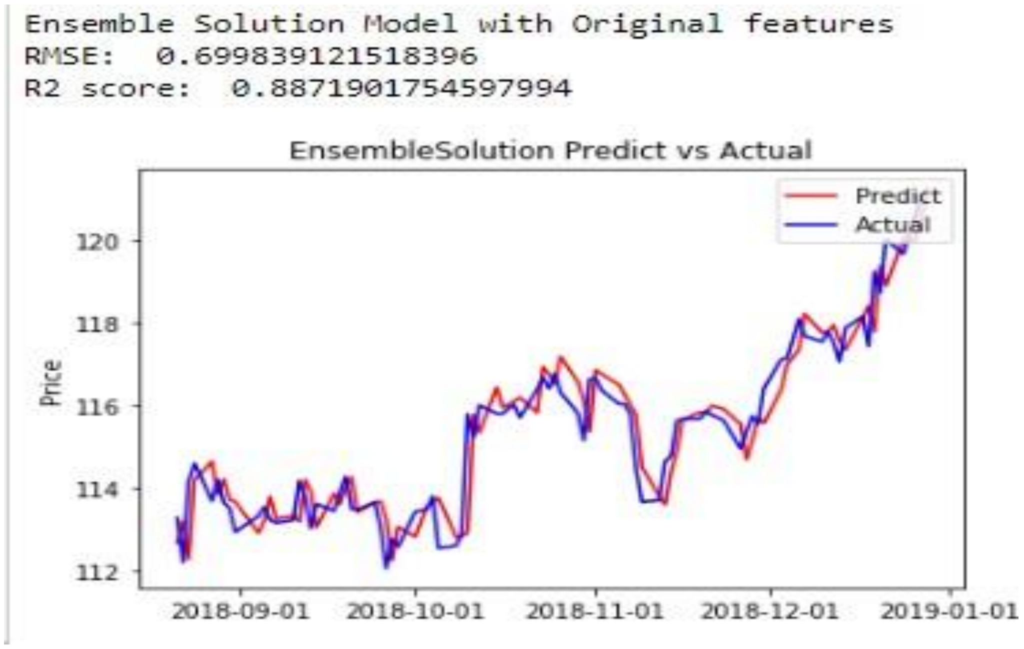
Model Evaluation and Validation

Finally we picked up three top performing models for ensemble solution which are Lasso CV, Bayesian ridge and Ridge CV in case of original features dataset and Lasso CV, Bayesian Ridge and SVR with linear kernel in case of selected feature dataset and at last compare this ensemble model with Benchmark model, Linear SVR, Lasso CV and Bayesian ridge model.

We create a class that ensemble these three models, it would take input, and average the result predicted by model.

Below is the snapshot of the ensemble solution code.


```
class EnsembleSolution:
    models = []
    def __init__(self, models):
        self.models = models
    def fit(self, X, y):
        for i in self.models:
            i.fit(X, y)
    def predict(self, X):
        result = 0
        for i in self.models:
            result = result + i.predict(X)

        result = result / len(self.models)

        return result
```


```
Ensemble Solution Model with Original features
RMSE:  0.699839121518396
R2 score:  0.8871901754597994
```

Ensemble solution with all features shows best result (with RMSE 0.699 and R2 score of 0.887) in comparison with other solution models.

Ensemble solution with feature selection has better solution (RMSE 0.711 and R2score 0.884) but Lasso has best performance (RMSE - 0.709 and R2 score 0.884)

Validation

Now comes the validation part in this step we will see whether the ensemble and other solution models are robust or not. We check this through training them on random time series split and also test on random split and take average of **RMSE** and **R2**.

First, we train **Benchmark**, **LinearSVR, Lasso, Ridge, Bayesian Ridge** and the **Ensemble** model on different data split. By the **train_reg_multipletimes** function.This function would train the pass in model several times (7 times I choose), and use different parameters on **TimeSeriesSplit** in each time and average the **R2** and **RMSE**.

Models	**RMSE**	**R2 Score**
Benchmark model	1.53	0.41
LSVR	3.102	- 2.668
Lasso	1.05	0.72
Ridge	1.58	0.159
Bayesian Ridge	1.55	0.185
Ensemble	1.38	0.40

Above is the result of training models multiple times on different cross validation, the Lasso CV performs the best, and ensemble model is the second best and benchmark model is the third best.

Finally, we again try above models on different split of validation set. You can see the actual implementation in Jupyter Notebook. Below is the performance result of all the models

Models	**RMSE**	**R2 Score**
Benchmark model	1.224	-3.111
LSVR	0.74	-0.239
Lasso	0.72	-0.197
Bayesian Ridge	0.693	-0.0321
Ensemble	0.692	-0.035

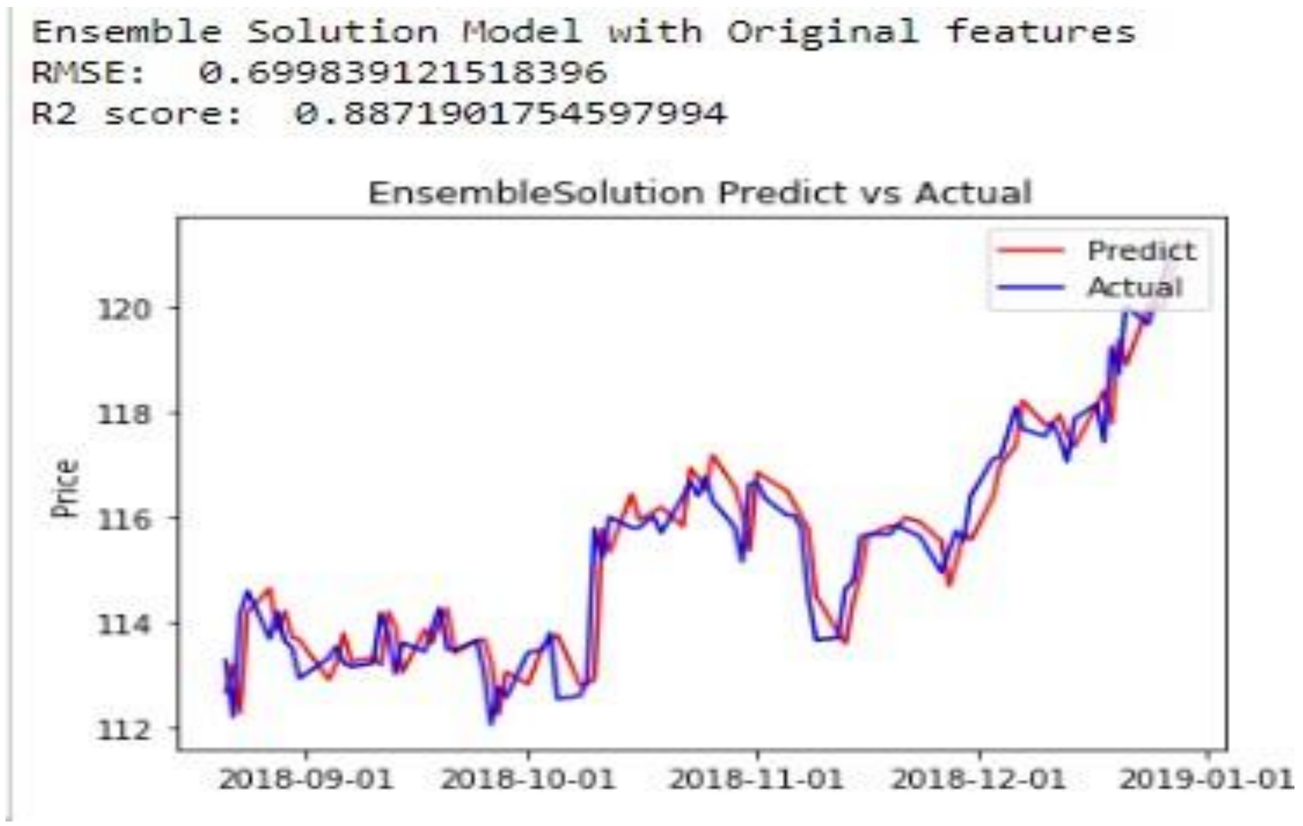
Justification

After validation on comparing the Benchmark model – Decision Tree with solution models ensemble model, Bayesian ridge and Lasso CV has RMSE of 0.692,0.693 and 0.72 respectively has very lesser RMSE in comparison to benchmark model's RMSE of 1.224. So, it is evident that our solution model has surpassed the performance of benchmark model.

V Conclusion

Free-Form Visualization

I have already discussed all the important features of the datasets and their visualization in above sections. But in order to conclude my report I would choose ensemble model with original feature visualization. As I was very satisfied by seeing how close I have predicted the price with actual price and it has lowest Root Mean Square error of 0.699



```
Ensemble Solution Model with Original features
RMSE:  0.699839121518396
R2 score:  0.8871901754597994
```

Reflection

The process undertaken this project are as follows :

- Set Up Infrastructure
  - Python Notebook
  - Incorporate required libraries Sklearn, Numpy, Pandas, Matplotlib and Seaborn
  - Git Project Organization

- Dataset Preparation
  - Incorporate data from different sources
  - Process the data into pandas dataframe
  - Normalize the data using Sklearn's MinMax Function
  - Timeseries Split with n_splits=10

- Develop Benchmark Model
  - Set up basic decision tree regressor with default parameters as benchmark model
- Develop Solution Model & improve using GridsearchCV
- Ensemble top three performing models

- Evaluate all the solution models and benchmark model & Plot the result
- Refine the models using Feature selection
- Develop Benchmark, solution and ensemble models
- Evaluate and compare the results with Original feature models
- Plot, analyze and describe the results for report.

It is almost practically impossible to get a model that can 100% predict the price without any error, there are too many factors on which gold prices depends I tried to capture as much factors as possible but still there are other factors such as CPI, political factors, disasters, financial breakdown that can affect the market. But this solution model performed really well in this project, which will help the trader to make better decision. The general trend of predicted price is in line with the actual data, so the trader could have an indicator to reference, and makes trading decision by himself.

Improvement

There are still many technical indicators and feature variables which we have not included in our project, may be there are some other indicators which we haven't explored would perform better.

There are lots of Machine Learning algorithms which we haven't tried and may be neural network or LSTM would perform better than our solution.

And last but not the least we have used data of around 14 years if we would increase the data, I think the performance of our solution models may be improved.