Machine Learning Nanodegree

Capstone Project

Importing Libraries

```
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestRegressor
from matplotlib.pyplot import figure
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.dates as mdates
from sklearn import linear_model
from sklearn.model_selection import TimeSeriesSplit
from sklearn.svm import SVR
```

About Data

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.

Attributes:

Features

- Gold ETF :- Date, Open, High, Low, Close and Volume.
- S&P 500 Index :- 'SP_open', 'SP_high', 'SP_low', 'SP_close', 'SP_Ajclose', 'SP_volume'
- Dow Jones Index :- 'DJ_open','DJ_high', 'DJ_low', 'DJ_close', 'DJ_Ajclose', 'DJ_volume'
- Eldorado Gold Corporation (EGO) :- 'EG_open', 'EG_high', 'EG_low', 'EG_close', 'EG_Ajclose', 'EG_volume'

- EURO - USD Exchange Rate :- 'EU_Price','EU_open', 'EU_high', 'EU_low', 'EU_Trend'
- Brent Crude Oil Futures :- 'OF_Price', 'OF_Open', 'OF_High', 'OF_Low', 'OF_Volume', 'OF_Trend'
- Crude Oil WTI USD :- 'OS_Price', 'OS_Open', 'OS_High', 'OS_Low', 'OS_Trend'
- Silver Futures :- 'SF_Price', 'SF_Open', 'SF_High', 'SF_Low', 'SF_Volume', 'SF_Trend'
- US Bond Rate (10 years) :- 'USB_Price', 'USB_Open', 'USB_High','USB_Low', 'USB_Trend'
- Platinum Price :- 'PLT_Price', 'PLT_Open', 'PLT_High', 'PLT_Low','PLT_Trend'
- Palladium Price :- 'PLD_Price', 'PLD_Open', 'PLD_High', 'PLD_Low','PLD_Trend'
- Rhodium Prices :- 'RHO_PRICE'
- US Dollar Index : 'USDI_Price', 'USDI_Open', 'USDI_High','USDI_Low', 'USDI_Volume', 'USDI_Trend'
- Gold Miners ETF :- 'GDX_Open', 'GDX_High', 'GDX_Low', 'GDX_Close', 'GDX_Adj Close', 'GDX_Volume'
- Oil ETF USO :- 'USO_Open','USO_High', 'USO_Low', 'USO_Close', 'USO_Adj Close', 'USO_Volume'

Target Variable

- Gold ETF :- Adjusted Close

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

```
df_final = pd.read_csv("FINAL_USO.csv",na_values=['null'],index_col='Date',parse_da
```

```
df_final.head()
```

	Open	High	Low	Close	Adj Close	Volume	SP_open
Date							
2011-12-15	154.740005	154.949997	151.710007	152.330002	152.330002	21521900	123.029999

```
df_final.shape
```

```
(1718, 80)
```

12-19	155.479996	155.860001	154.360001	154.869995	154.869995	12547200	122.059998

So, we have 1718 records in the dataset and 80 columns including Adjusted Close which is our target variable.

2011

```
df_final.describe()
```

	Open	High	Low	Close	Adj Close	Volume	
count	1718.000000	1718.000000	1718.000000	1718.000000	1718.000000	1.718000e+03	1
mean	127.323434	127.854237	126.777695	127.319482	127.319482	8.446327e+06	
std	17.526993	17.631189	17.396513	17.536269	17.536269	4.920731e+06	
min	100.919998	100.989998	100.230003	100.500000	100.500000	1.501600e+06	
25%	116.220001	116.540001	115.739998	116.052502	116.052502	5.412925e+06	
50%	121.915001	122.325001	121.369999	121.795002	121.795002	7.483900e+06	
75%	128.427494	129.087497	127.840001	128.470001	128.470001	1.020795e+07	
max	173.199997	174.070007	172.919998	173.610001	173.610001	9.380420e+07	

Checking Missing Values

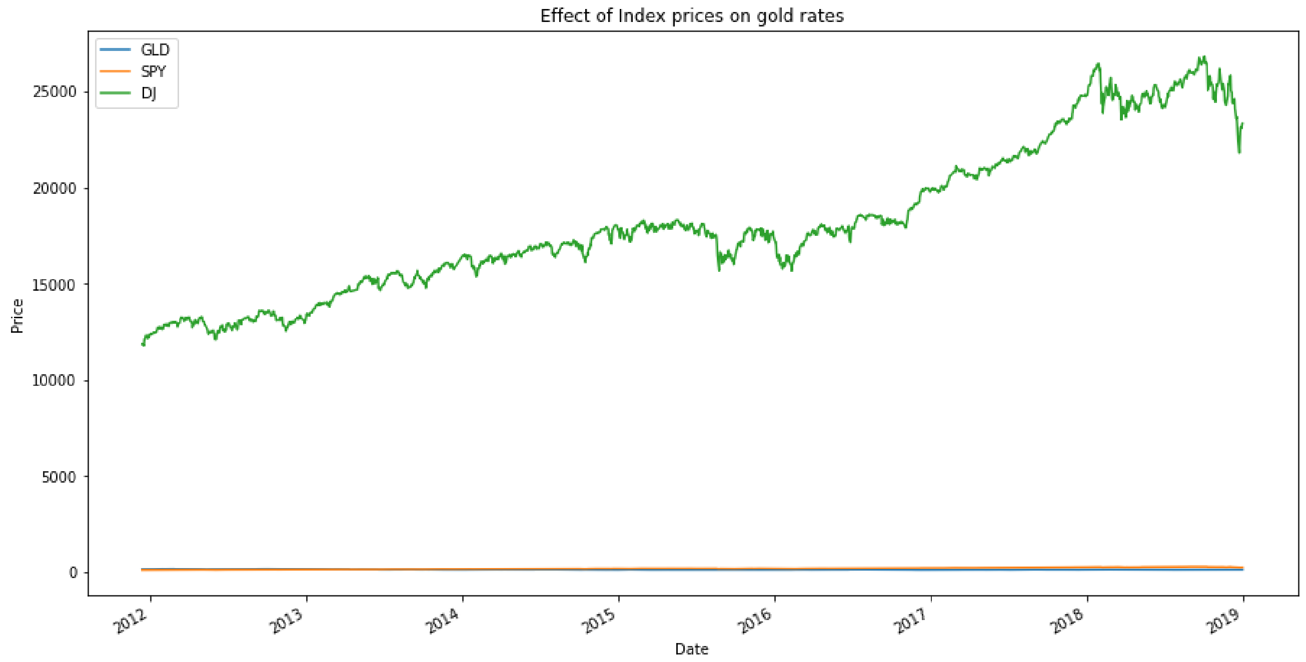
```
df_final.isnull().values.any()
```

```
False
```

That's great ! we dont have any missing values in our dataset

Effect of Index prices on gold rates

```
GLD_adj_close = df_final['Adj Close']
SPY_adj_close = df_final['SP_Ajclose']
DJ_adj_close  = df_final['DJ_Ajclose']

df_p = pd.DataFrame({'GLD':GLD_adj_close, 'SPY':SPY_adj_close, 'DJ':DJ_adj_close})

df_ax = df_p.plot(title='Effect of Index prices on gold rates',figsize=(15,8))

df_ax.set_ylabel('Price')
df_ax.legend(loc='upper left')
plt.show()
```

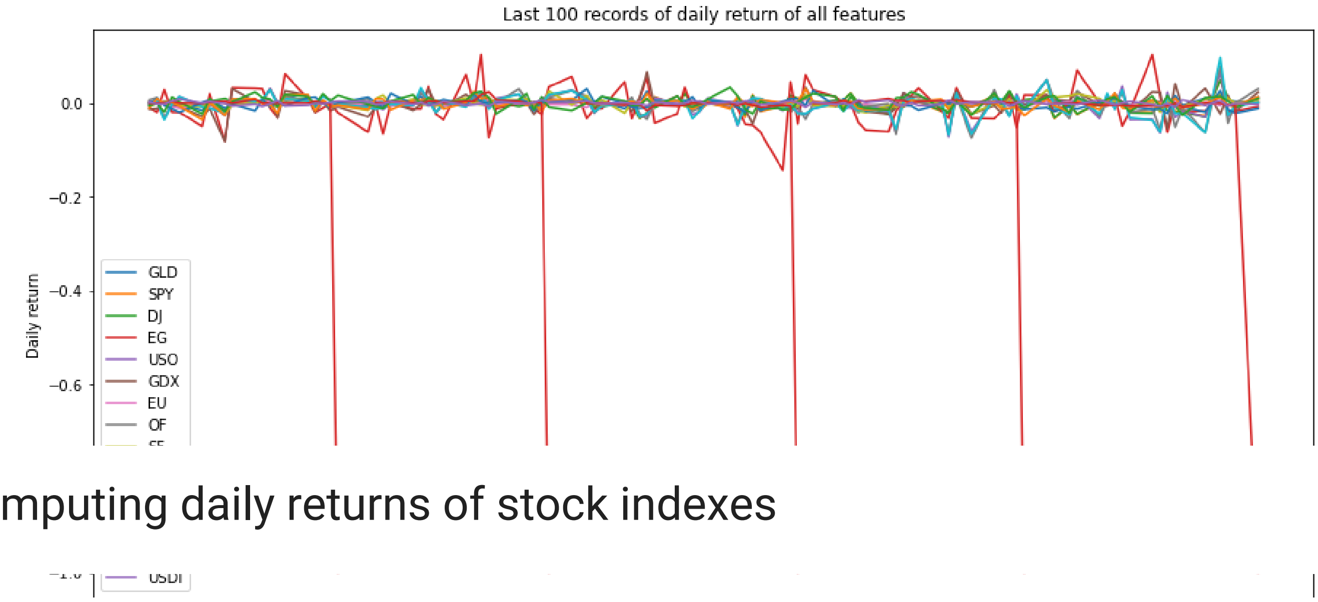


Computing Daily Returns of all Features

```
def compute_daily_returns(df):
    """Compute and return the daily return values."""
    # TODO: Your code here
    # Note: Returned DataFrame must have the same number of rows
    daily_return = (df / df.shift(1)) - 1
    daily_return[0] = 0
    return daily_return
```

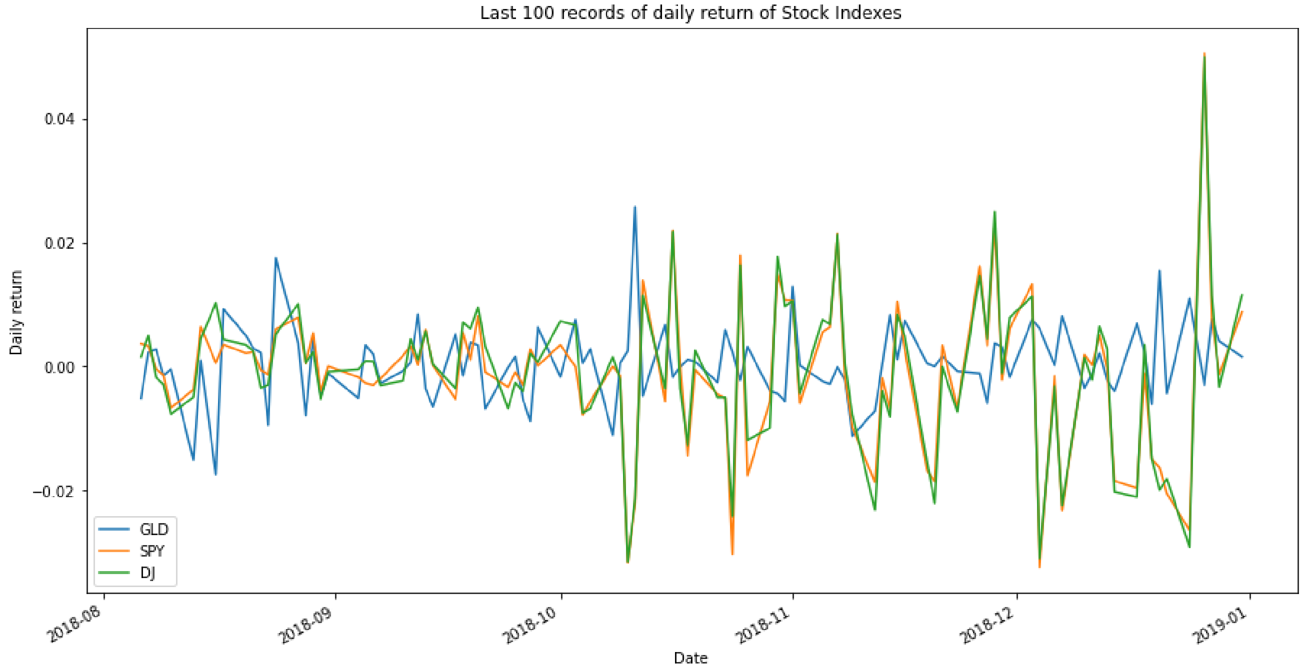
```
GLD_adj_close = df_final['Adj Close']
SPY_adj_close = df_final['SP_Ajclose']
DJ_adj_close  = df_final['DJ_Ajclose']
```

```
EG_adj_close =  df_final['EG_Ajclose']
USO_Adj_close = df_final['USO_Adj Close']
GDX_Adj_close = df_final['GDX_Adj Close']
EU_price      = df_final['EU_Price']
OF_price      = df_final['OF_Price']
OS_price      = df_final['OS_Price']
SF_price      = df_final['SF_Price']
USB_price      = df_final['USB_Price']
PLT_price      = df_final['PLT_Price']
PLD_price      = df_final['PLD_Price']
rho_price      = df_final['RHO_PRICE']
usdi_price      = df_final['USDI_Price']



GLD_daily_return = compute_daily_returns(GLD_adj_close)
SPY_daily_return = compute_daily_returns(SPY_adj_close)
DJ_adj_return    = compute_daily_returns(DJ_adj_close)
EG_adj_return     = compute_daily_returns(EG_adj_close)
USO_Adj_return    = compute_daily_returns(USO_Adj_close)
GDX_Adj_return   =compute_daily_returns(GDX_Adj_close)
EU_return        = compute_daily_returns(EU_price)
OF_price         =compute_daily_returns(OF_price)
OS_price         =compute_daily_returns(OS_price)
SF_price         =compute_daily_returns(SF_price)
USB_price         =compute_daily_returns(USB_price)
PLT_price         =compute_daily_returns(PLT_price)
PLD_price         =compute_daily_returns(PLD_price)
rho_price         =compute_daily_returns(rho_price)
USDI_price         =compute_daily_returns(usdi_price)

df_d = pd.DataFrame({'GLD':GLD_daily_return, 'SPY':SPY_daily_return, 'DJ':DJ_adj_re
                  'GDX':GDX_Adj_return,'EU':EU_return, 'OF':OF_price,'SF':SF_price,
                  'RHO':rho_price,'USDI':USDI_price})

daily_ax = df_d[-100:].plot(title='Last 100 records of daily return of all features

daily_ax.set_ylabel('Daily return')
daily_ax.legend(loc='lower left')
plt.show()
```



Computing daily returns of stock indexes

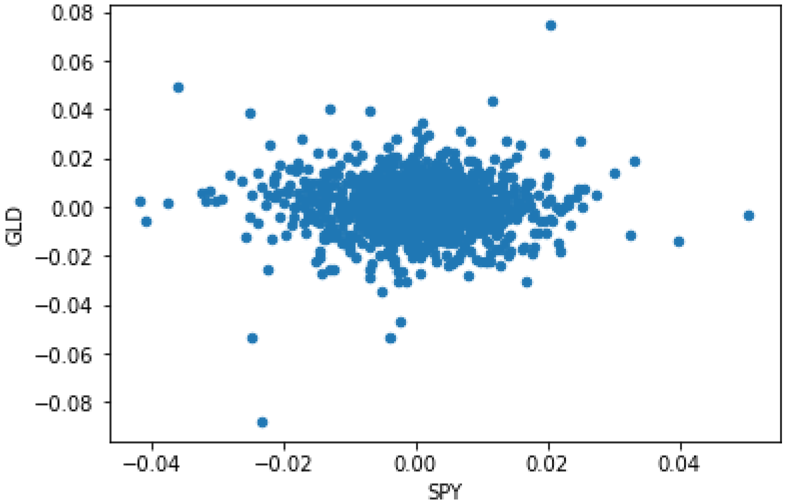
```
df_s = pd.DataFrame({'GLD':GLD_daily_return, 'SPY':SPY_daily_return, 'DJ':DJ_adj_re

daily_ax = df_s[-100:].plot(title='Last 100 records of daily return of Stock Indexe

daily_ax.set_ylabel('Daily return')
daily_ax.legend(loc='lower left')
plt.show()
```



Scatterplot

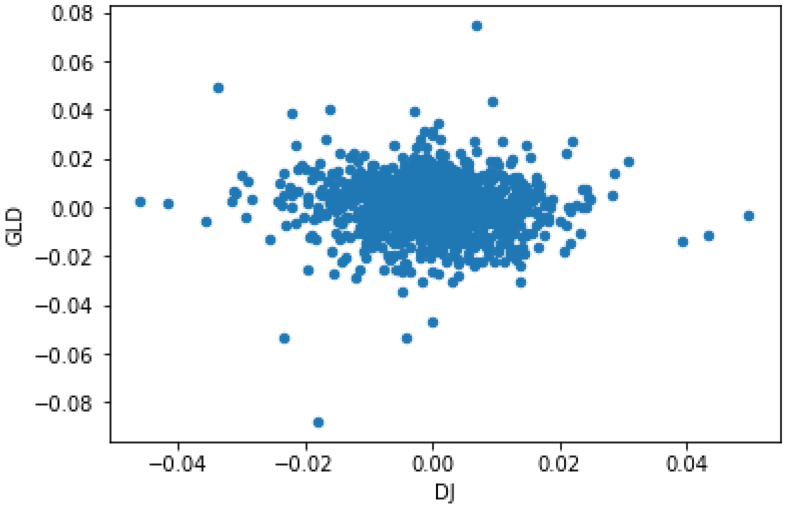
```
df_d.plot(kind='scatter', x='SPY', y='GLD')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f9cf7521fd0>



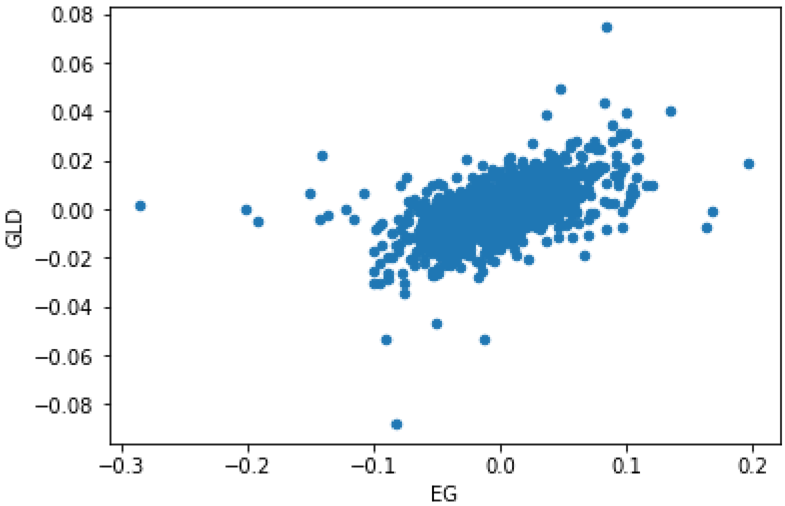

```
df_d.plot(kind='scatter', x='DJ', y='GLD')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f9cf7eff2d0>



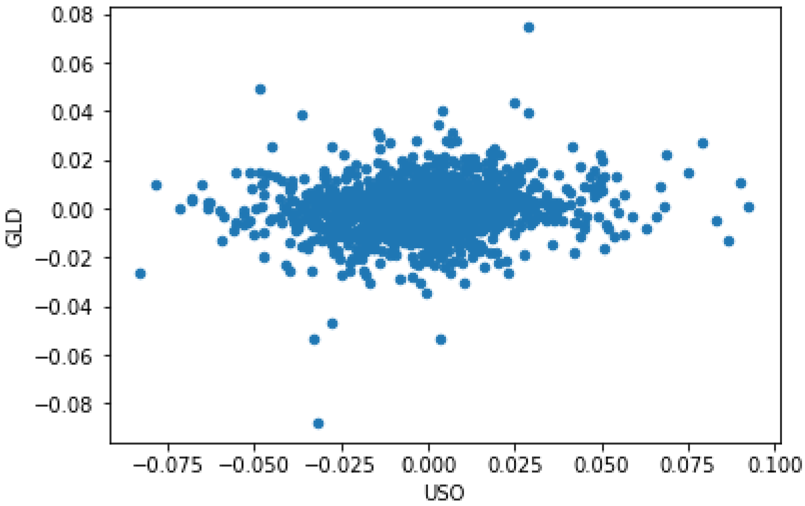

```
df_d.plot(kind='scatter', x='EG', y='GLD')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f9cf7181850>



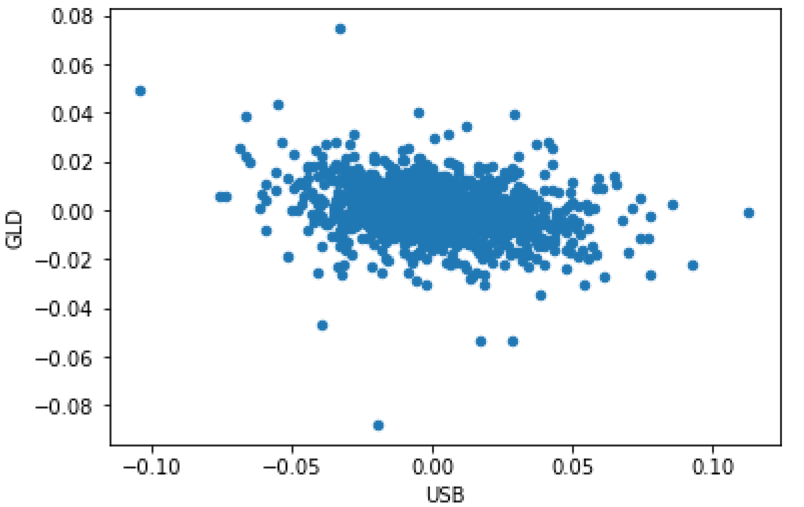
```
df_d.plot(kind='scatter', x='USO', y='GLD')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f9cf793db50>



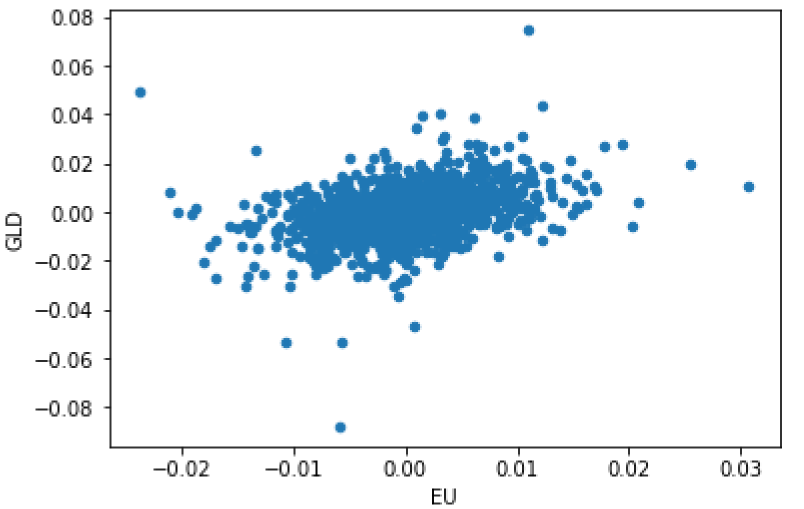

```
df_d.plot(kind='scatter', x='USB', y='GLD')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f9cf6e48ad0>



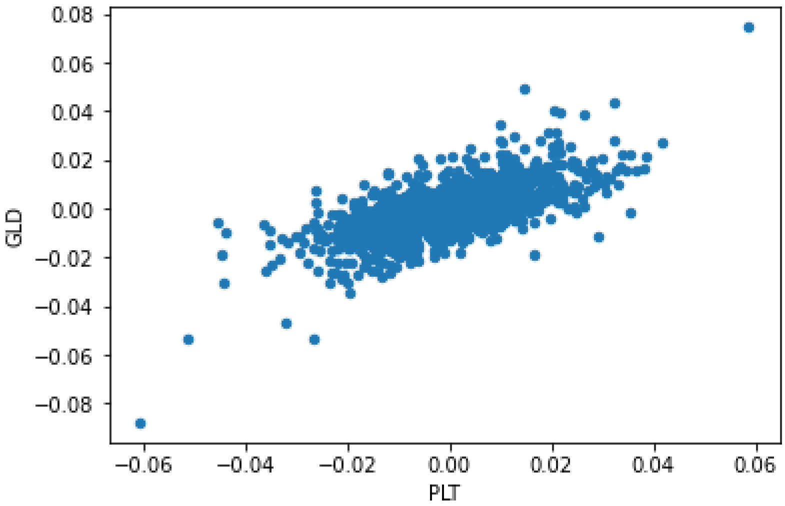

```
df_d.plot(kind='scatter', x='EU', y='GLD')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f9cf4525250>



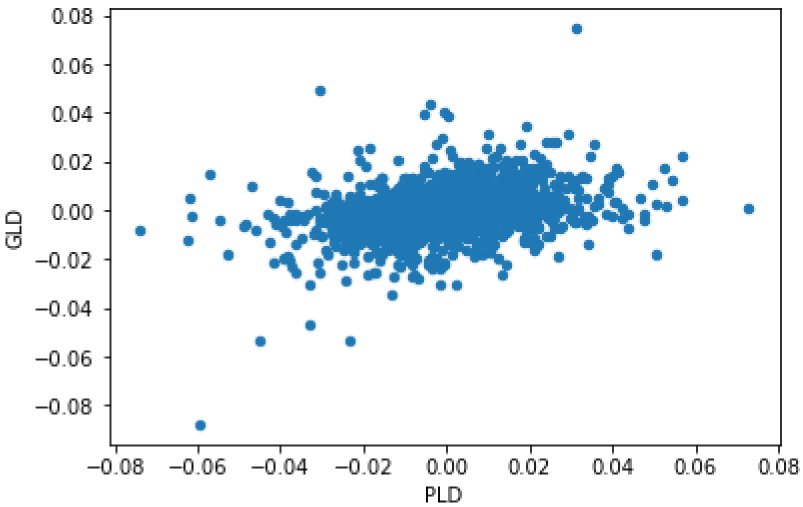
```
df_d.plot(kind='scatter', x='PLT', y='GLD')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9cf785dad0>
```




```
df_d.plot(kind='scatter', x='PLD', y='GLD')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9cf427d110>
```



Statistical Measures (Mean, Standard deviation, Kurtosis)

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.

For investors, high kurtosis of the return distribution implies that the investor will experience occasional extreme returns (either positive or negative), more extreme than the usual + or - three standard deviations from the mean that is predicted by the normal distribution of returns. This phenomenon is known as **kurtosis risk**.

Positive Kurtosis More weights in the tail

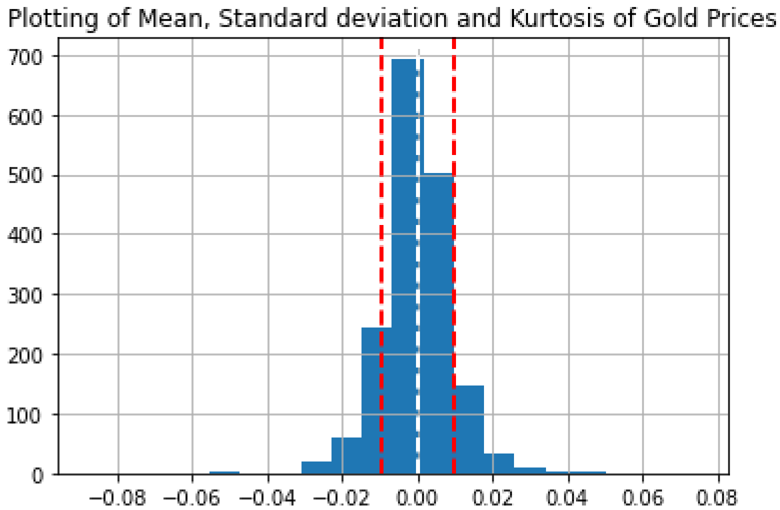
Positive

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

Negative

```
# computing mean,standard deviation and kurtosis of Gold ETF daily return

mean=df_d['GLD'].mean()
# computing standard deviation of Gold stock
std=df_d['GLD'].std()
kurt=df_d['GLD'].kurtosis()
print('Mean=',mean)
print('Standard Deviation=',std)
print('Kurtosis=',kurt)
#Plotting Histogram
df_d['GLD'].hist(bins=20)

plt.axvline(mean, color='w',linestyle='dashed',linewidth=2)
plt.axvline(std, color='r',linestyle='dashed',linewidth=2)
plt.axvline(-std, color='r',linestyle='dashed',linewidth=2)
plt.title("Plotting of Mean, Standard deviation and Kurtosis of Gold Prices")
plt.show()
```

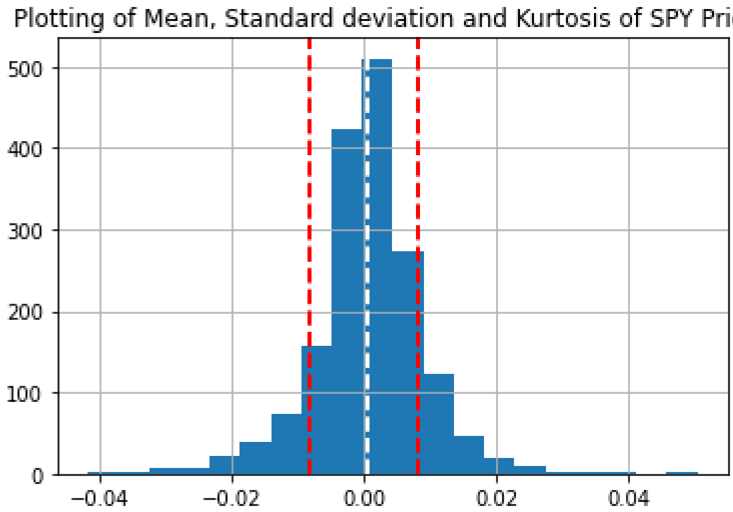
```
Mean= -8.656986121281953e-05
Standard Deviation= 0.009611536167006395
Kurtosis= 8.60658492491834
```




```
# computing mean,standard deviation and kurtosis of S&P 500 Index daily return

mean=df_d['SPY'].mean()
# computing standard deviation of Gold stock
std=df_d['SPY'].std()
kurt=df_d['SPY'].kurtosis()
print('Mean=',mean)
print('Standard Deviation=',std)
print('Kurtosis=',kurt)
#Plotting Histogram
df_d['SPY'].hist(bins=20)
```

```
plt.axvline(mean, color='w',linestyle='dashed',linewidth=2)
plt.axvline(std, color='r',linestyle='dashed',linewidth=2)
plt.axvline(-std, color='r',linestyle='dashed',linewidth=2)
plt.title("Plotting of Mean, Standard deviation and Kurtosis of SPY Prices")
plt.show()
```

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




```
# computing mean,standard deviation and kurtosis of Dow Jones Index daily return
mean=df_d['DJ'].mean()
# computing standard deviation of Gold stock
std=df_d['DJ'].std()
kurt=df_d['DJ'].kurtosis()
print('Mean=',mean)
print('Standard Deviation=',std)
print('Kurtosis=',kurt)
#Plotting Histogram
df_d['DJ'].hist(bins=20)

plt.axvline(mean, color='w',linestyle='dashed',linewidth=2)
plt.axvline(std, color='r',linestyle='dashed',linewidth=2)
plt.axvline(-std, color='r',linestyle='dashed',linewidth=2)
plt.title("Plotting of Mean, Standard deviation and Kurtosis of Dow jones Prices")
plt.show()
```

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

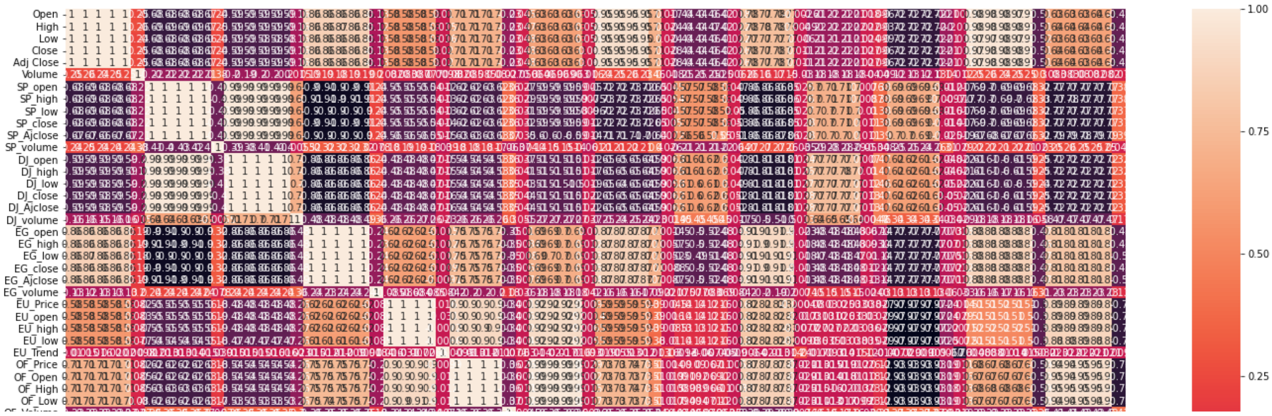
Plotting of Mean, Standard deviation and Kurtosis of Dow jones Prices

Correlation Analysis

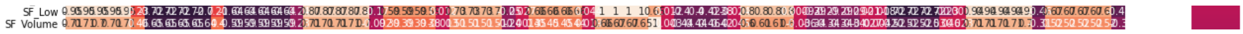
Plotting Correlation Matrix

```
plt.figure(figsize=(24,18))
sns.heatmap(df_final.corr(), annot=True)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f9cf771dfd0>

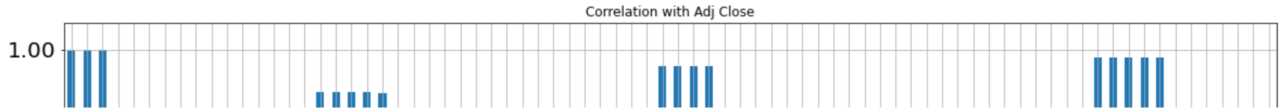



```
X=df_final.drop(['Adj Close'],axis=1)
X=X.drop(['Close'],axis=1)
```




```
X.corrwith(df_final['Adj Close']).plot.bar(
        figsize = (20, 10), title = "Correlation with Adj Close", fontsize = 20,
        rot = 90, grid = True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9cf1493a90>
```




```
corr_matrix=df_final.corr()
coef=corr_matrix["Adj Close"].sort_values(ascending=False)
```

Positively Correlated Variables

```
pos_corr=coef[coef>0]
pos_corr
```

```
Adj Close        1.000000
Close            1.000000
High             0.999535
Low              0.999532
Open             0.998976
GDX_Low          0.975561
GDX_Close        0.975459
GDX_High         0.975255
GDX_Adj Close    0.974980
GDX_Open         0.974824
SF_Low           0.947842
SF_Price         0.947420
SF_Open          0.945557
SF_High          0.945203
EG_low           0.863917
EG_open          0.862900
EG_close         0.862770
EG_high          0.861479
EG_Ajclose       0.859850
PLT_Price        0.775861
PLT_High         0.775481
PLT_Low          0.773993
PLT_Open         0.773760
OF_High          0.711334
OF_Price         0.710693
OF_Open          0.709096
OF_Low           0.708266
SF_Volume        0.706505
USO_Adj Close    0.635675
USO_Close        0.635675
USO_High         0.635311
USO_Open         0.635197
USO_Low          0.634732
OS_High          0.632001
OS_Price         0.630817
OS_Open          0.630046
OS_Low           0.629083
EU_high          0.582969
EU_Price         0.581036
EU_open          0.579036
EU_low           0.577000
Volume           0.246778
SP_volume        0.241949
RHO_PRICE        0.095782
```

```
OS_Trend         0.059510
OF_Trend         0.048205
SF_Trend         0.028100
PLD_Trend        0.026536
EU_Trend         0.019913
PLT_Trend        0.011355
Name: Adj Close, dtype: float64
```

Negatively Correlated Variables

[ ] ↳ 1 cell hidden

Technical Indicators

[ ] ↳ 2 cells hidden

Plotting Technical Indicators

[ ] ↳ 6 cells hidden

Normalizing the data

In this step I would perform feature scaling/normalization of feature variables using sklearn's MinMaxScaler function.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
feature_minmax_transform_data = scaler.fit_transform(test[feature_columns])
feature_minmax_transform = pd.DataFrame(columns=feature_columns, data=feature_minma
feature_minmax_transform.head()
```

	Open	High	Low	Volume	SP_open	SP_high	SP_low	SP_Ajclose
Date								
2012-	0.913669	0.912561	0.913193	0.079151	0.037098	0.034927	0.040627	0.028113

```
display(feature_minmax_transform.head())
print('Shape of features : ', feature_minmax_transform.shape)
print('Shape of target : ', target_adj_close.shape)

# Shift target array because we want to predict the n + 1 day value


target_adj_close = target_adj_close.shift(-1)
validation_y = target_adj_close[-90:-1]
target_adj_close = target_adj_close[:-90]

# Taking last 90 rows of data to be validation set
validation_X = feature_minmax_transform[-90:-1]
feature_minmax_transform = feature_minmax_transform[:-90]
display(validation_X.tail())
display(validation_y.tail())

print("\n -----After process------ \n")
print('Shape of features : ', feature_minmax_transform.shape)
print('Shape of target : ', target_adj_close.shape)
display(target_adj_close.tail())
```

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

5 rows × 84 columns

```
Shape of features :  (1685, 84)
Shape of target :  (1685, 1)
```

	Open	High	Low	Volume	SP_open	SP_high	SP_low	SP_Ajclose
Date								
2018-12-21	0.252767	0.249863	0.252304	0.131396	0.719499	0.732264	0.685249	0.721173
2018-12-24	0.258024	0.262042	0.266061	0.089215	0.672900	0.678571	0.650574	0.685607
2018-12-26	0.272551	0.273810	0.266061	0.138587	0.654321	0.710896	0.647477	0.751818
2018-12-27	0.271859	0.272441	0.273903	0.112378	0.694263	0.723668	0.679055	0.762387
2018-12-28	0.275042	0.274904	0.281882	0.058103	0.736686	0.742494	0.724540	0.760598

5 rows × 84 columns

	Adj Close
Date	

Train Test Split

In this step we would perform Train test split using sklearn's Timeseries split

```
ts_split= TimeSeriesSplit(n_splits=10)
for train_index, test_index in ts_split.split(feature_minmax_transform):
        X_train, X_test = feature_minmax_transform[:len(train_index)], feature_minm
```

```
        y_train, y_test = target_adj_close[:len(train_index)].values.ravel(), targe
```

```
X_train.shape
```

```
(1450, 84)
```

```
X_test.shape
```

```
(145, 84)
```

```
y_train.shape
```

```
(1450,)
```

```
y_test.shape
```

```
(145,)
```

```
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()
```

Model Building

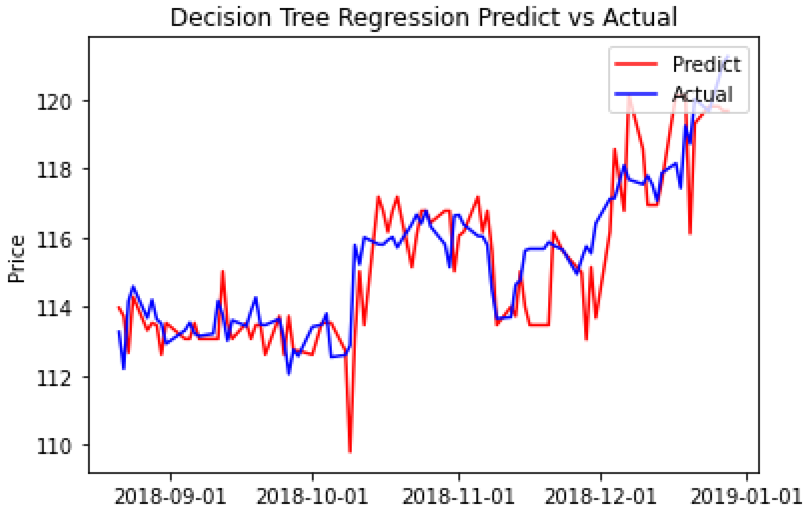
1. Benchmark Model :

```
    I will use Decision Tree Regressor with default parameter as my Benchmark model for the
```

```
from sklearn.tree import DecisionTreeRegressor

dt = DecisionTreeRegressor(random_state=0)

benchmark_dt=dt.fit(X_train, y_train)

validate_result(benchmark_dt, 'Decision Tree Regression')
```

```
validate_result(benchmark_dt, 'Decision Tree Regression')
```

```
RMSE:  1.209072477734157
R2 score:  0.6632906610684199
```

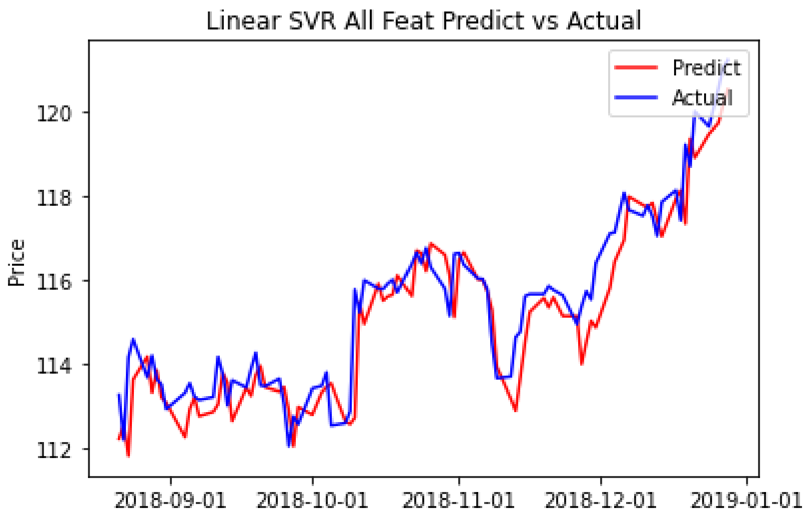


Solution Model

Support Vector Regressor (SVR)

```
# Save all soultion models
solution_models = {}
# SVR with  linear Kernel
svr_lin = SVR(kernel='linear')
linear_svr_clf_feat = svr_lin.fit(X_train,y_train)
validate_result(linear_svr_clf_feat,'Linear SVR All Feat')
```

```
RMSE:  0.8136995579159704
R2 score:  0.8474969071831893
```

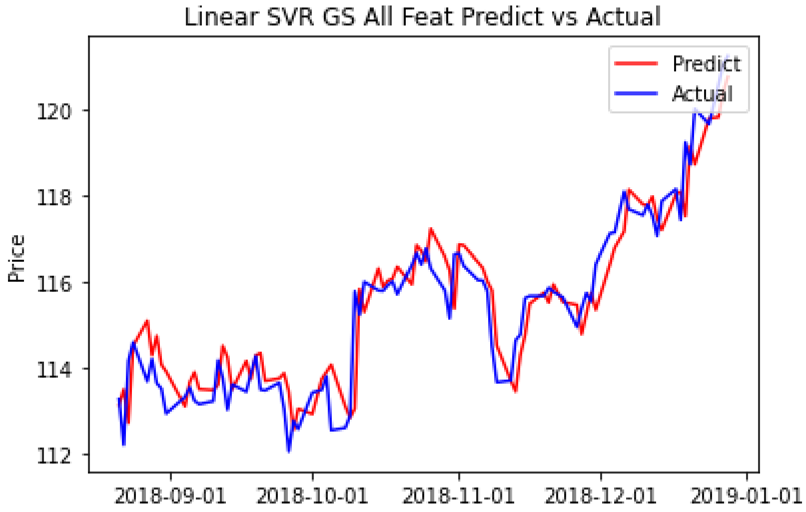


Hyperparameter Tuning

In this step I will tune two parameters of SVR C and epsilon to see if the model shows any

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

lsvr_grid_search_feat = GridSearchCV(estimator=linear_svr_clf_feat,
                                     param_grid=linear_svr_parameters,
                                     cv=ts_split,
)

lsvr_grid_search_feat.fit(X_train, y_train)

validate_result(lsvr_grid_search_feat,'Linear SVR GS All Feat')
```

```
RMSE:  0.7416902627051684
R2 score:  0.8732944477453806
```



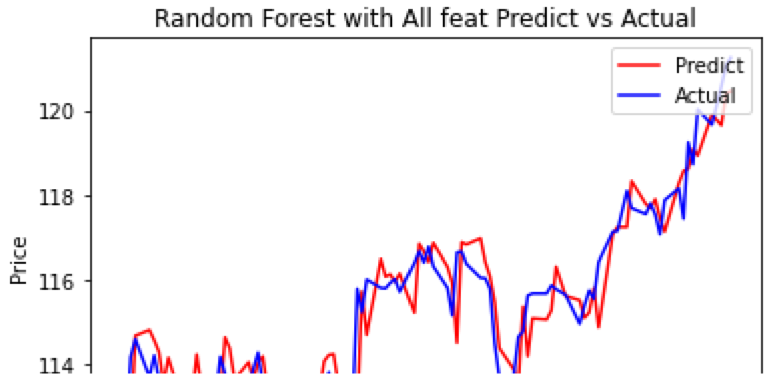
As we have seen using gridsearch on SVR we get significant improvement in R2 score and RMSE also came down so we will save this as our first solution model

```
solution_models['SVR All Feat'] = lsvr_grid_search_feat
```

Solution Model : Random Forest

```
rf_cl = RandomForestRegressor(n_estimators=50, random_state=0)
random_forest_clf_feat = rf_cl.fit(X_train,y_train)
validate_result(random_forest_clf_feat,'Random Forest with All feat')
```

```
RMSE:  0.8148324354418975
R2 score:  0.8470719650951224
```



Hyper parameter Tuning

In this I will tune 3 parameters of Random forest which are n_estimators,max_features,max_depth

```
random_forest_parameters = {
    'n_estimators':[10,15,20, 50, 100],
    'max_features':['auto','sqrt','log2'],
    'max_depth':[2, 3, 5, 7,10],
}

grid_search_RF_feat = GridSearchCV(estimator=random_forest_clf_feat,
                                   param_grid=random_forest_parameters,
                                   cv=ts_split,
)

grid_search_RF_feat.fit(X_train, y_train)
```

```
GridSearchCV(cv=TimeSeriesSplit(gap=0, max_train_size=None, n_splits=10, test_
             estimator=RandomForestRegressor(n_estimators=50, random_state=0),
             param_grid={'max_depth': [2, 3, 5, 7, 10],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'n_estimators': [10, 15, 20, 50, 100]})
```

```
print(grid_search_RF_feat.best_params_)
validate_result(grid_search_RF_feat,'RandomForest GS')
```

```
{'max_depth': 7, 'max_features': 'auto', 'n_estimators': 20}
RMSE:  0.8219610022630158
R2 score:  0.8443844767381965
```

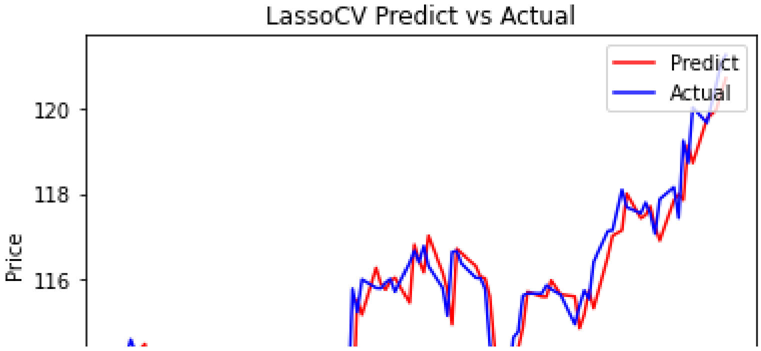
As we have seen, Random forest with default parameters performed better than tuned Random forest model.So, we will include random forest with default parameters as our second solution model.

```
solution_models['Random_Forest with Feat'] = random_forest_clf_feat
```

Solution Model : Lasso and Ridge

```
from sklearn.linear_model import LassoCV
from sklearn.linear_model import RidgeCV

lasso_clf = LassoCV(n_alphas=1000, max_iter=3000, random_state=0)
ridge_clf = RidgeCV(gcv_mode='auto')

lasso_clf_feat = lasso_clf.fit(X_train,y_train)
validate_result(lasso_clf_feat,'LassoCV')
solution_models['LassoCV All feat'] = lasso_clf_feat

ridge_clf_feat = ridge_clf.fit(X_train,y_train)
validate_result(ridge_clf_feat,'RidgeCV')
solution_models['RidgeCV All Feat'] = ridge_clf_feat
```

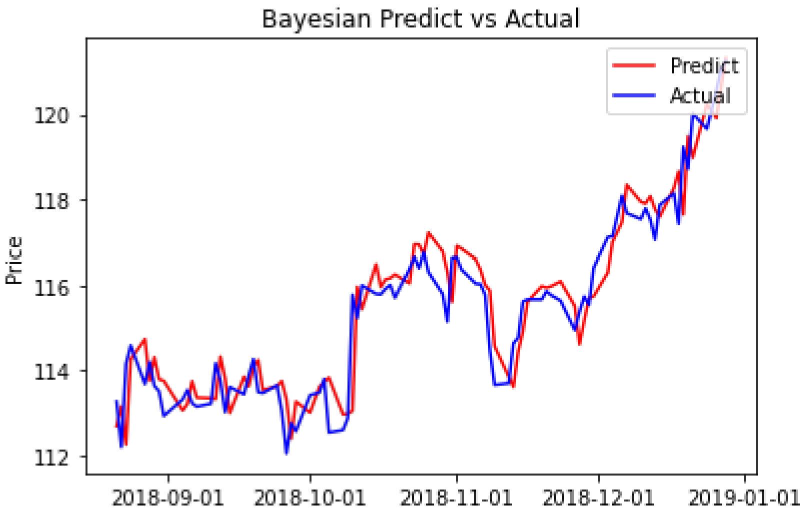
```
RMSE:  0.7117047240324225
R2 score:  0.8833324203076907
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descer
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
```



Solution Model : Bayesian Ridge


```
from sklearn import linear_model
bay = linear_model.BayesianRidge()
bay_feat = bay.fit(X_train,y_train)
validate_result(bay_feat,'Bayesian')
solution_models['Bay All Feat'] = bay_feat
```

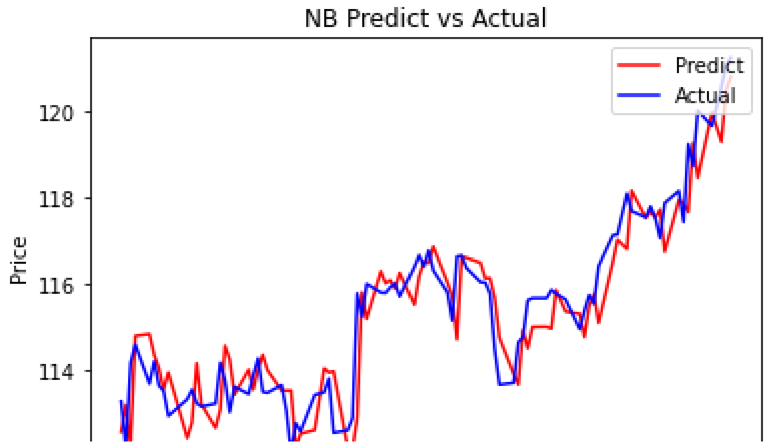
```
RMSE:  0.7195639601746158
R2 score:  0.8807415122691951
```



Solution Model : Gradient Boosting Regressor

```
from sklearn.ensemble import GradientBoostingRegressor
regr =GradientBoostingRegressor(n_estimators=70, learning_rate=0.1,max_depth=4, ran
GB_feat = regr.fit(X_train,y_train)
validate_result(GB_feat,'NB')
solution_models['GB All Feat'] = GB_feat
```

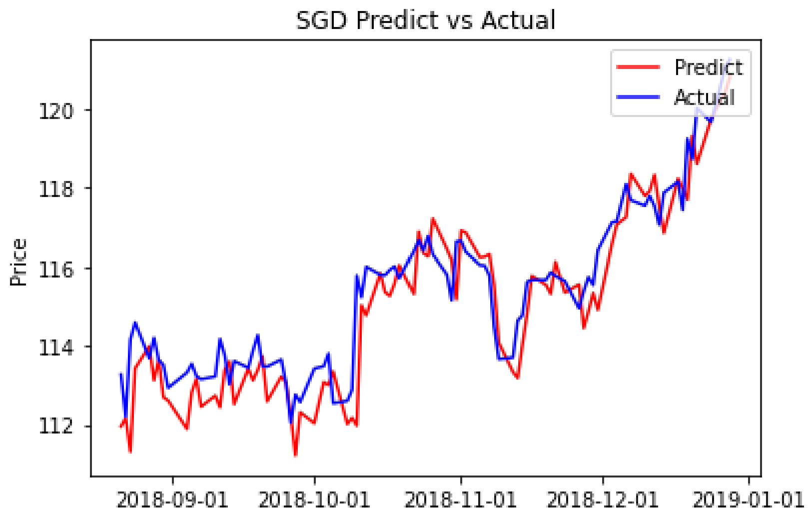
```
/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_gb.py:290: FutureWarn
  FutureWarning,
RMSE:  0.8094931831292773
R2 score:  0.8490695443986888
```



Solution Model : Stochastic Gradient Descent (SGD)

```
from sklearn.linear_model import SGDRegressor
sgd =SGDRegressor(max_iter=1000, tol=1e-3,loss='squared_epsilon_insensitive',penalt
sgd_feat = sgd.fit(X_train,y_train)
validate_result(sgd_feat,'SGD')
solution_models['SGD All Feat'] = sgd_feat
```

```
RMSE:  0.9020743795306159
R2 score:  0.8125716905896547
```

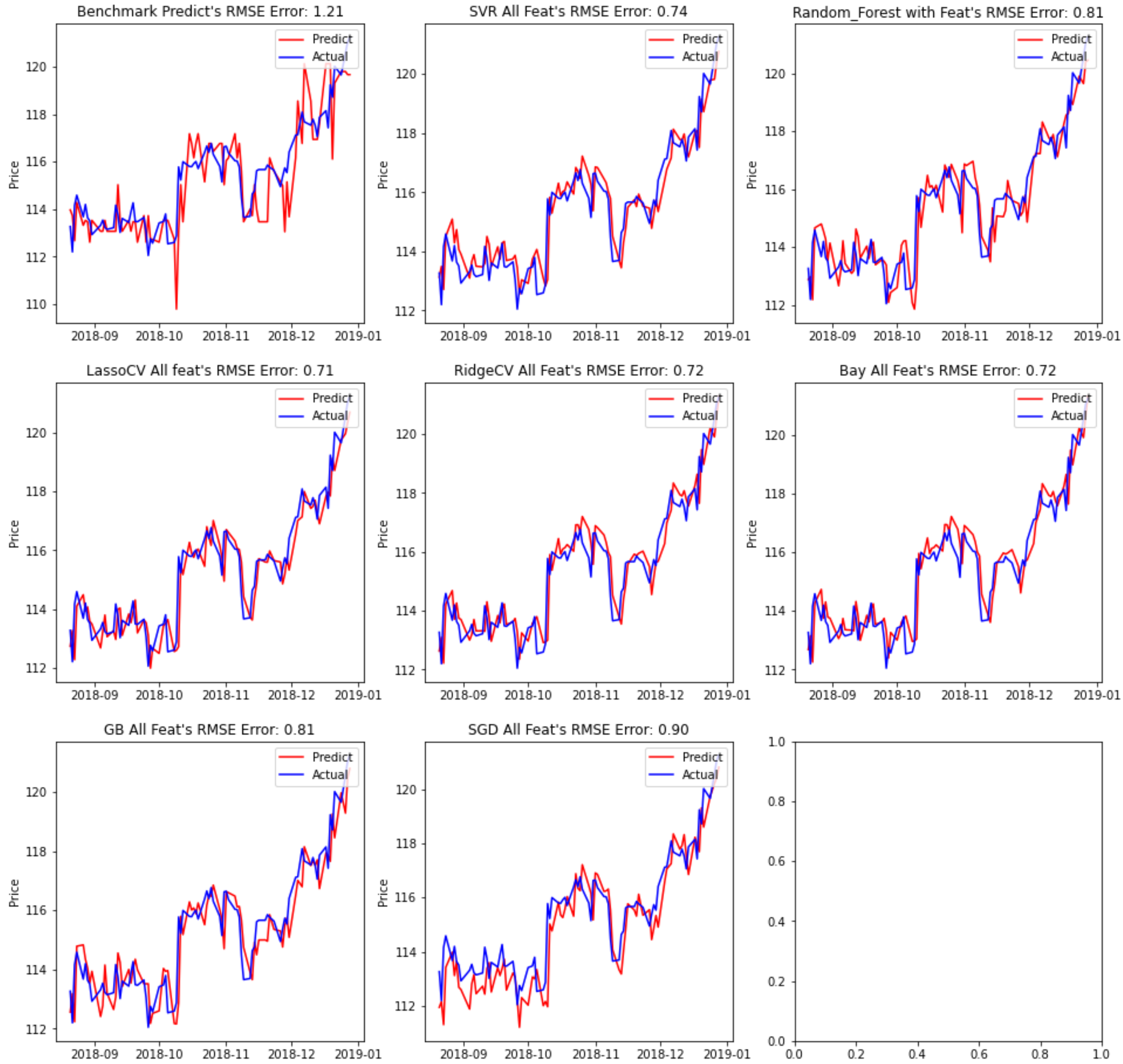


Model Review

In this step, we will review benchmark model and all the solution model based on evaluation metrics i.e, RMSE and R2 score

```
RMSE_scores = {}
def model_review(models):
    fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(16, 16))
```

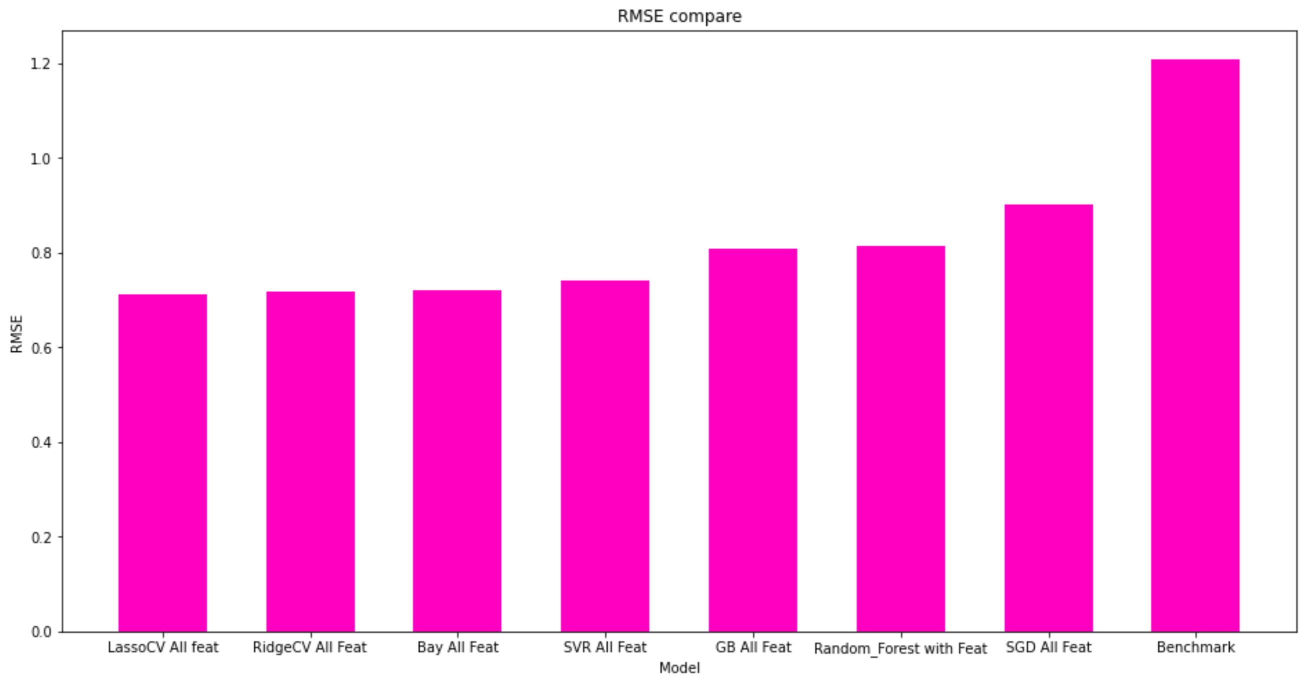
```
    #plot benchmark model
    benchmark_predicted = benchmark_dt.predict(validation_X)
    benchmark_RSME_score = np.sqrt(mean_squared_error(validation_y, benchmark_predi
    RMSE_scores['Benchmark'] = benchmark_RSME_score

    axes[0,0].plot(validation_y.index, benchmark_predicted,'r', label='Predict')
    axes[0,0].plot(validation_y.index, validation_y,'b', label='Actual')
    axes[0,0].xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))
    axes[0,0].xaxis.set_major_locator(mdates.MonthLocator())
    axes[0,0].set_ylabel('Price')
    axes[0,0].set_title("Benchmark Predict's RMSE Error: " +"{0:.2f}".format(benchm
    axes[0,0].legend(loc='upper right')

    #plot block
    ax_x = 0
    ax_y = 1
    #plot solution model
    for name, model in models.items():
        predicted = model.predict(validation_X)
        RSME_score = np.sqrt(mean_squared_error(validation_y, predicted))


        axes[ax_x][ax_y].plot(validation_y.index, predicted,'r', label='Predict')
        axes[ax_x][ax_y].plot(validation_y.index, validation_y,'b', label='Actual')
        axes[ax_x][ax_y].xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))
        axes[ax_x][ax_y].xaxis.set_major_locator(mdates.MonthLocator())
        axes[ax_x][ax_y].set_ylabel('Price')
        axes[ax_x][ax_y].set_title(name + "'s RMSE Error: " +"{0:.2f}".format(RSME_
        axes[ax_x][ax_y].legend(loc='upper right')
        RMSE_scores[name] = RSME_score
        if ax_x <=2:
            if ax_y < 2:
                ax_y += 1
            else:
                ax_x += 1
                ax_y = 0
    plt.show()

model_review(solution_models)
```



Comparison of RMSE of Benchmark and all Solution Models

```
model_names = []
model_values = []
for name, value in RMSE_scores.items():
    model_names.append(name)
    model_values.append(value)

model_values = np.array(model_values)
model_names = np.array(model_names)

indices = np.argsort(model_values)
columns = model_names[indices[:8]]
values = model_values[indices][:8]

fig = plt.figure(figsize = (16,8))
plt.bar(np.arange(8), values ,width = 0.6, align="center", color = '#ff00c1')
```

```
plt.xticks(np.arange(8), columns)
plt.xlabel('Model')
plt.ylabel('RMSE')
plt.title('RMSE compare')
plt.show()
```



Feature Selection

In this step we will select supporting features using sklearn's **SelectFromModel** library using Lasso regressor as it has lowest RMSE.

```
from sklearn.feature_selection import SelectFromModel

sfm = SelectFromModel(lasso_clf_feat)
sfm.fit(feature_minmax_transform, target_adj_close.values.ravel())
display(feature_minmax_transform.head())
sup = sfm.get_support()
zipped = zip(feature_minmax_transform,sup)
print(*zipped)
```

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

5 rows × 84 columns

```
# Selecting Features which supports Model building process

feature_selected = feature_minmax_transform[['Open','High','Low','OF_Trend','USB_Tr
feature_selected_validation_X = validation_X[['Open','High','Low','OF_Trend','USB_T
display(feature_selected.head())
display(feature_selected_validation_X.head())
```

	Open	High	Low	OF_Trend	USB_Trend	PLT_Trend	USDI_Price	GD
Date								
2012-02-06	0.913669	0.912561	0.913193	1.0	0.0	0.0	0.035457	
2012-02-07	0.919480	0.945539	0.920622	1.0	1.0	1.0	0.015007	
2012-02-08	0.945490	0.943760	0.925437	1.0	1.0	1.0	0.017368	
2012-02-09	0.955866	0.949370	0.927775	1.0	1.0	0.0	0.014087	
2012-02-10	0.907167	0.912014	0.909341	0.0	0.0	1.0	0.037058	

	Open	High	Low	OF_Trend	USB_Trend	PLT_Trend	USDI_Price	GD
Date								
2018-08-21	0.163946	0.165709	0.167974	1.0	1.0	0.0	0.674324	

Train Test Split

```
    for train_index, test_index in ts_split.split(feature_selected):
            X_train, X_test = feature_selected[:len(train_index)], feature_selected[len
            y_train, y_test = target_adj_close[:len(train_index)].values.ravel(), targe
```

validation Feature Selected Benchmark & Solution Model

[ ] ↳ 1 cell hidden

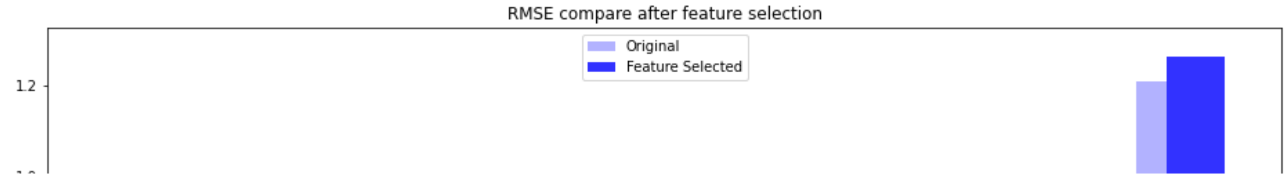
Model Review

```
FS_RMSE_scores = {}

def fs_model_review(models):
    fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(16, 16))

    #plot benchmark model
    benchmark_dt_predicted = benchmark_dt_fs.predict(feature_selected_validation_X)
    benchmark_RSME_score = np.sqrt(mean_squared_error(validation_y, benchmark_dt_pr
    FS_RMSE_scores['Benchmark'] = benchmark_RSME_score

    axes[0,0].plot(validation_y.index, benchmark_dt_predicted,'y', label='Predict')
    axes[0,0].plot(validation_y.index, validation_y,'b', label='Actual')
    axes[0,0].xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))
    axes[0,0].xaxis.set_major_locator(mdates.MonthLocator())
    axes[0,0].set_ylabel('Price')
    axes[0,0].set_title("Benchmark Predict's RMSE Error: " +"{0:.2f}".format(benchm
    axes[0,0].legend(loc='upper right')

    #plot block
    ax_x = 0
    ax_y = 1
    #plot solution model
    for name, model in models.items():
        predicted = model.predict(feature_selected_validation_X)
        RSME_score = np.sqrt(mean_squared_error(validation_y, predicted))

        R2_score = r2_score(validation_y, predicted)

        axes[ax_x][ax_y].plot(validation_y.index, predicted,'y', label='Predict')
        axes[ax_x][ax_y].plot(validation_y.index, validation_y,'b', label='Actual')
        axes[ax_x][ax_y].xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))
        axes[ax_x][ax_y].xaxis.set_major_locator(mdates.MonthLocator())
        axes[ax_x][ax_y].set_ylabel('Price')
        axes[ax_x][ax_y].set_title(name + "'s RMSE Error: " +"{0:.2f}".format(RSME_
        axes[ax_x][ax_y].legend(loc='upper right')
        FS_RMSE_scores[name] = RSME_score
        if ax_x <=2:
            if ax_y < 2:
                ax_y += 1
```

```
            else:
                ax_x += 1
                ax_y = 0
    plt.show()

fs_model_review(feature_selected_solution_models)
```

Comparison of RMSE of Feature selected models and Original Features model

```
fs_model_names = []
fs_model_values = []
for name, value in FS_RMSE_scores.items():
    fs_model_names.append(name)
    fs_model_values.append(value)

fs_model_values = np.array(fs_model_values)
fs_model_names = np.array(fs_model_names)

fs_indices = np.argsort(fs_model_values)
fs_columns = fs_model_names[fs_indices[:8]]
fs_values = fs_model_values[fs_indices][:8]
origin_values = model_values[fs_indices][:8]

fig = plt.figure(figsize = (16,8))
plt.bar(np.arange(8) - 0.2 , origin_values ,width = 0.4, align="center", color = '#
plt.bar(np.arange(8), fs_values ,width = 0.4, align="center", color = '#3232ff', la
plt.xticks(np.arange(8), fs_columns)
plt.xlabel('Model')
plt.ylabel('RMSE')
plt.title('RMSE compare after feature selection')
plt.legend(loc = 'upper center')
plt.show()
```



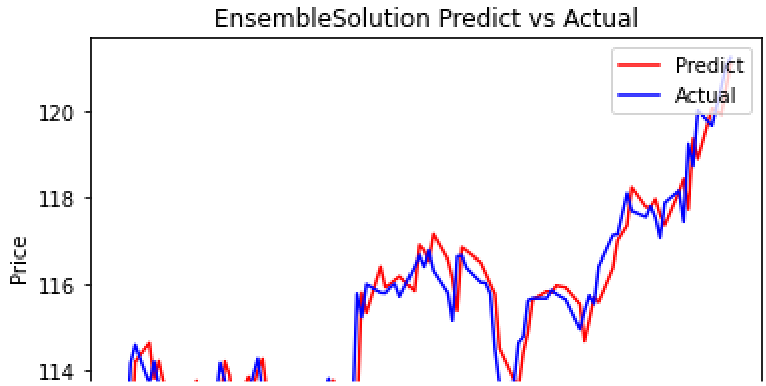
As we have seen from the above plot 3 Feature selected models performs better in RMSE error reduction and Feature selected Linear SVR is the best as it has RMSE of 0.716 in feature selected model and 0.741 with all features model. Also Lasso cv and Bayesian Ridge performs slightly better from original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

Ensemble Solution

So now we will ensemble top three performing models i.e, in case of all the features model Lasso,Bayesian ridge and Ridge are the best performing models so we will ensemble these three models while in case of feature selected models we will combine Lasso,Bayesian Ridge and Linear SVR and will compare all the feature ensemble models with feature selected ensemble models.

```
# Choosing the top three performing models to ensemble them
ensemble_solution_models = [lasso_clf_feat, bay_feat, ridge_clf_feat]
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


print("Ensemble Solution Model with Original features")
EnsembleModel = EnsembleSolution(ensemble_solution_models)
validate_result(EnsembleModel,'EnsembleSolution')
```

```
Ensemble Solution Model with Original features
RMSE:  0.7007271848704582
R2 score:  0.8869036929492634
```



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_model_fs = [lasso_clf_fs,bay_feat_fs,linear_svr_clf_fs]

print("Ensemble Solution Model with Selected features")
EnsembleModel_fs = EnsembleSolution(ensemble_solution_model_fs)
feature_selected_validate_result(EnsembleModel_fs,'EnsembleSolution with FS')
```

```
Ensemble Solution Model with Selected features
EnsembleSolution with FS

RMSE:  0.7106997685499894
R2 score:  0.883661666249319
----------------------
```

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)

Train Model Multiple Times

By the train_reg_multipletimes function.This function would train the model several times (I choosed 7 times), and use different parameters on TimeSeriesSplit in each time, average the R2 and RMSE. I will apply this function on Benchmark model and on top performing solution models with all features which are Linear SVR, Lasso, Ridge and Bayesian ridge and compare the same.

```
def train_reg_multipletimes(model, times):
    total_rmse = 0
    total_r2 = 0
    for i in range(times):
        reg = model
        for train_index, test_index in TimeSeriesSplit(n_splits=i+2).split(feature_
            X_train, X_test = feature_minmax_transform[:len(train_index)], feature_
            y_train, y_test = target_adj_close[:len(train_index)].values.ravel(), t
```

```
            reg.fit(X_train, y_train)
        predicted = reg.predict(validation_X)
        rmse, r2 = print_result(validation_y, predicted, [0,len(validation_y)])
        total_rmse += rmse
        total_r2 += r2
    return total_rmse / times, total_r2 / times

def print_result(actual, predict, index):
    RMSE_score = np.sqrt(mean_squared_error(actual, predict))
    print('From {} to {}'.format(index[0],index[-1]))
    print('RMSE: ', RMSE_score)
    R2_score = r2_score(actual, predict)
    print('R2 score: ', R2_score)
    print('--------------------')
    return RMSE_score, R2_score

print('Benchmark')
t_multiple_benchmark_RMSE,t_multiple_benchmark_R2 = train_reg_multipletimes(benchma
print('RMSE: {} //  R2: {}\n'.format(t_multiple_benchmark_RMSE, t_multiple_benchmar
```

```
Benchmark
From 0 to 89
RMSE:  2.5169023928812693
R2 score:  -0.45909388930653794
---------------------
From 0 to 89
RMSE:  1.9398836843424083
R2 score:  0.13323399036382322
---------------------
From 0 to 89
RMSE:  1.3807422861211505
R2 score:  0.5608875582297289
---------------------
From 0 to 89
RMSE:  3.168571954563515
R2 score:  -1.312478156510708
---------------------
From 0 to 89
RMSE:  3.156524043590225
R2 score:  -1.2949260452829572
---------------------
From 0 to 89
RMSE:  1.2151075466375603
R2 score:  0.6599209117630461
---------------------
From 0 to 89
RMSE:  1.2041319111018567
R2 score:  0.6660367927511187
---------------------
RMSE: 2.0831234027482837 //  R2: -0.14948840542749808
```

```
print('LSVR')
t_multiple_LSVR_RMSE,t_multiple_LSVR_R2 = train_reg_multipletimes(linear_svr_clf_fe
print(' RMSE: {} //  R2: {}'.format(t_multiple_LSVR_RMSE, t_multiple_LSVR_R2))
```

```
LSVR
From 0 to 89
RMSE:  7.448867721242006
R2 score:  -11.780004755021666
---------------------
From 0 to 89
RMSE:  6.193294220830053
R2 score:  -7.8347429042944565
---------------------
From 0 to 89
RMSE:  3.51963019211457
R2 score:  -1.8532811045906588
---------------------
From 0 to 89
RMSE:  1.31631673621737
R2 score:  0.6009095848161232
---------------------
From 0 to 89
RMSE:  1.1821451622443897
R2 score:  0.6781213954199379
---------------------
From 0 to 89
RMSE:  1.097280488203965
R2 score:  0.7226770542570906
---------------------
From 0 to 89
RMSE:  0.958971832231373
R2 score:  0.7881823267768645
---------------------
 RMSE: 3.102358050440532 //  R2: -2.668305486090966
```

```
print('Lasso')
t_multiple_lasso_RMSE,t_multiple_lasso_R2 = train_reg_multipletimes(lasso_clf_feat,
print(' RMSE: {} //  R2: {}'.format(t_multiple_lasso_RMSE, t_multiple_lasso_R2))
```

```
Lasso
From 0 to 89
RMSE:  1.141446143219328
R2 score:  0.699903215958863
---------------------
From 0 to 89
RMSE:  1.6184903121926937
R2 score:  0.3966480791468283
---------------------
From 0 to 89
RMSE:  1.2272053345149017
R2 score:  0.6531154475171917
---------------------
From 0 to 89
RMSE:  0.9458767879471476
R2 score:  0.7939276958254233
---------------------
From 0 to 89
RMSE:  0.8279814045285367
R2 score:  0.8420965359183785
---------------------
From 0 to 89
RMSE:  0.7928926592226434
R2 score:  0.8551964245471746
```

```
---------------------
From 0 to 89
RMSE:  0.7724600996807726
R2 score:  0.8625633366388292
---------------------
 RMSE: 1.046621820186575 //  R2: 0.7290643907932414
```

```
print('Ridge')
t_multiple_ridge_RMSE,t_multiple_ridge_R2 = train_reg_multipletimes(ridge_clf_feat,
print(' RMSE: {} //  R2: {}'.format(t_multiple_ridge_RMSE, t_multiple_ridge_R2))
```

```
Ridge
From 0 to 89
RMSE:  3.0358192307485194
R2 score:  -1.122766898394148
---------------------
From 0 to 89
RMSE:  3.361570100216821
R2 score:  -1.6027642671000146
---------------------
From 0 to 89
RMSE:  1.6567313507046708
R2 score:  0.3677997365458099
---------------------
From 0 to 89
RMSE:  0.8283219433964485
R2 score:  0.8419666215880584
---------------------
From 0 to 89
RMSE:  0.7493347104805284
R2 score:  0.8706691320990159
---------------------
From 0 to 89
RMSE:  0.717599661491697
R2 score:  0.8813917381210988
---------------------
From 0 to 89
RMSE:  0.7123183568826676
R2 score:  0.8831311516635544
---------------------
 RMSE: 1.580242193417336 //  R2: 0.15991817350333926
```

```
print('BayRidge')
t_multiple_bayridge_RMSE,t_multiple_bayridge_R2 = train_reg_multipletimes(bay_feat,
print(' RMSE: {} //  R2: {}'.format(t_multiple_bayridge_RMSE, t_multiple_bayridge_R
```

```
BayRidge
From 0 to 89
RMSE:  3.0049452619300716
R2 score:  -1.0798098080641862
---------------------
From 0 to 89
RMSE:  3.317186778085761
R2 score:  -1.5344886031368254
---------------------
From 0 to 89
RMSE:  1.597251002758483
R2 score:  0.4123796463141326
```

```
---------------------
From 0 to 89
RMSE:  0.7995748378668925
R2 score:  0.85274544802953
---------------------
From 0 to 89
RMSE:  0.7307474880240321
R2 score:  0.8770056522932022
---------------------
From 0 to 89
RMSE:  0.7098918230074388
R2 score:  0.8839260299485945
---------------------
From 0 to 89
RMSE:  0.7070366584924018
R2 score:  0.8848578446515624
---------------------
 RMSE: 1.5523762643092973 //  R2: 0.18523088714800137
```

```
print('Ensemble')
t_multiple_ensemble_RMSE,t_multiple_ensemble_R2 = train_reg_multipletimes(EnsembleS
print(' RMSE: {} //  R2: {}\n'.format(t_multiple_ensemble_RMSE, t_multiple_ensemble
```

```
Ensemble
From 0 to 89
RMSE:  2.3651010857514856
R2 score:  -0.28839759733247106
---------------------
From 0 to 89
RMSE:  2.7533763819291512
R2 score:  -0.7461504366394291
---------------------
From 0 to 89
RMSE:  1.4876420936359722
R2 score:  0.49026153330365463
---------------------
From 0 to 89
RMSE:  0.8472355619942524
R2 score:  0.8346672670920421
---------------------
From 0 to 89
RMSE:  0.7555732429228262
R2 score:  0.8685066980716548
---------------------
From 0 to 89
RMSE:  0.7204376102992527
R2 score:  0.8804517438842036
---------------------
From 0 to 89
RMSE:  0.7109945317122859
R2 score:  0.883565143447322
---------------------
 RMSE: 1.3771943583207467 //  R2: 0.41755776454671095
```

```
def cross_validate(model, ts_split):
    clf = model
    total_rmse = 0
```

```
    total_r2 = 0
    count = 0
    for train_index, test_index in ts_split.split(validation_X):
        X_test1, X_test2 = validation_X[:len(train_index)], validation_X[len(train_
        y_test1, y_test2 = validation_y[:len(train_index)].values.ravel(), validati
        predicted_test1 = clf.predict(X_test1)
        temp1_RMSE, temp1_R2 = print_result(y_test1, predicted_test1, train_index)

        predicted_test2 = clf.predict(X_test2)
        temp2_RMSE, temp2_R2 = print_result(y_test2, predicted_test2, test_index)

        total_rmse += temp1_RMSE + temp2_RMSE
        total_r2 += temp1_R2 + temp2_R2
        count += 2
    return total_rmse / count, total_r2 / count
```

Cross Validation

```
timeseries_cv = TimeSeriesSplit(n_splits=10)
test_bench__RMSE, test_bench_R2 = cross_validate(benchmark_dt,timeseries_cv)
```

```
R2 score:  -6.294727575190087
---------------------
From 0 to 32
RMSE:  1.1994671233542904
R2 score:  -3.2519324920753565
---------------------
From 33 to 40
RMSE:  1.2697238154428052
R2 score:  -0.6466118868276143

---------------------
From 0 to 40
RMSE:  1.213495194602714
R2 score:  -0.3166198016301527
---------------------
From 41 to 48
RMSE:  1.1078805204129623
R2 score:  -3.6618598597356753
---------------------
From 0 to 48
RMSE:  1.1968887222615123
R2 score:  0.17847775264847932
---------------------
From 49 to 56
RMSE:  1.3922409839914143
R2 score:  -0.9151193114538847
---------------------
From 0 to 56
RMSE:  1.2261855539186157
R2 score:  0.21927655942198443
---------------------
From 57 to 64
```

```
RMSE:  1.4807100619321951
R2 score:  -3.2511366949410334
---------------------
From 0 to 64
RMSE:  1.2602886483809603
R2 score:  0.13029582869961542
---------------------
From 65 to 72
RMSE:  0.9279775778599146
R2 score:  -0.4921185653311613
---------------------
From 0 to 72
RMSE:  1.2282654274551195
R2 score:  0.210149106063655
---------------------
From 73 to 80
RMSE:  1.0351315387505744
R2 score:  -9.030516178504952
---------------------
From 0 to 80
RMSE:  1.2105625848588892
R2 score:  0.42907949257186806
---------------------
From 81 to 88
RMSE:  1.1369747226758138
R2 score:  0.0946272844513888
---------------------
```

```
test_lsvr__RMSE, test_lsvr_R2 = cross_validate(lsvr_grid_search_feat,timeseries_cv)
```

```
From 0 to 8
RMSE:  0.9598378957440337
R2 score:  -0.9825809486429402
---------------------
From 9 to 16
RMSE:  0.6200304864546613
R2 score:  -2.041480944184528
---------------------
From 0 to 16
RMSE:  0.8177120702812725
R2 score:  -1.1496752713542273
---------------------
From 17 to 24
RMSE:  0.5010644286610816
R2 score:  -1.26557493785922
---------------------
From 0 to 24
RMSE:  0.7314540587454086
R2 score:  -1.1465603699564988
---------------------
From 25 to 32
RMSE:  0.8002035888164287
R2 score:  -1.0594620839132252
---------------------
From 0 to 32
RMSE:  0.7487005362791734
R2 score:  -0.6566316395414316
---------------------
From 33 to 40
RMSE:  1.0378437880607319
```

```
R2 score:  -0.10011147764143447
---------------------
From 0 to 40
RMSE:  0.8132318857068911
R2 score:  0.40869294932636857
---------------------
From 41 to 48
RMSE:  0.6622335530793207
R2 score:  -0.6656959266124016
---------------------
From 0 to 48
RMSE:  0.7905515554285993
R2 score:  0.641596556048452
---------------------
From 49 to 56
RMSE:  0.7595948611423622
R2 score:  0.4299262505172794
---------------------
From 0 to 56
RMSE:  0.7862802851928642
R2 score:  0.6789744480044105
---------------------
From 57 to 64
RMSE:  0.5678453654326652
R2 score:  0.3747910507718425
---------------------
From 0 to 64
RMSE:  0.762779109316979
```

```
test_ridge__RMSE, test_ridge_R2 = cross_validate(ridge_clf_feat,timeseries_cv)
```

```
R2 score:  -0.3709746968153498
---------------------
From 0 to 32
RMSE:  0.6659479329527943
R2 score:  -0.3106605077299107
---------------------
From 33 to 40
RMSE:  1.0970483764653298
R2 score:  -0.22920487336330164
---------------------
From 0 to 40
RMSE:  0.7692755628064281
R2 score:  0.47088736308971824
---------------------
From 41 to 48
RMSE:  0.6186638995251043
R2 score:  -0.45372718091052544
---------------------
From 0 to 48
RMSE:  0.7467636393182207
R2 score:  0.6802002599967387
---------------------
From 49 to 56
RMSE:  0.6859491739872478
R2 score:  0.5351092743894954
---------------------
From 0 to 56
RMSE:  0.738530438286739
R2 score:  0.7167814943286248
```

```
---------------------
From 57 to 64
RMSE:  0.5711062737255702
R2 score:  0.36758978426149236
---------------------
From 0 to 64
RMSE:  0.7200283101148157
R2 score:  0.7161227112086664
---------------------
From 65 to 72
RMSE:  0.6373339148930902
R2 score:  0.2961786649986501
---------------------
From 0 to 72
RMSE:  0.7114350347358946
R2 score:  0.7350089810638543
---------------------
From 73 to 80
RMSE:  0.4601503690763369
R2 score:  -0.9821282445412123
---------------------
From 0 to 80
RMSE:  0.6906975623069174
R2 score:  0.8141438246655778
---------------------
From 81 to 88
RMSE:  0.9025144991505628
R2 score:  0.429528229112353
---------------------
```

```
test_lasso__RMSE, test_lasso_R2 = cross_validate(lasso_clf_feat,timeseries_cv)
```

```
R2 score:  -0.6312186649885143
---------------------
From 0 to 32
RMSE:  0.716006116193007
R2 score:  -0.5151063813903662
---------------------
From 33 to 40
RMSE:  1.233009682994396
R2 score:  -0.5527648440223527
---------------------
From 0 to 40
RMSE:  0.842187049134352
R2 score:  0.36583630136625966
---------------------
From 41 to 48
RMSE:  0.5275986038528947
R2 score:  -0.057257164000477756
---------------------
From 0 to 48
RMSE:  0.7993278087883251
R2 score:  0.6335948037366042
---------------------
From 49 to 56
RMSE:  0.748542753626658
R2 score:  0.4463947139707469
---------------------
From 0 to 56
RMSE:  0.7923964582453901
```

```
R2 score:  0.6739607542324391
---------------------
From 57 to 64
RMSE:  0.6231312902373042
R2 score:  0.2471228192761571
---------------------
From 0 to 64
RMSE:  0.7735651056112085
R2 score:  0.6723386273524847
---------------------
From 65 to 72
RMSE:  0.685652594221525
R2 score:  0.18541461181541263
---------------------
From 0 to 72
RMSE:  0.7644242981948015
R2 score:  0.6940646757059235
---------------------
From 73 to 80
RMSE:  0.6137785876100886
R2 score:  -2.5265956514889374
---------------------
From 0 to 80
RMSE:  0.7508919987507857
R2 score:  0.7803374087005639
---------------------
From 81 to 88
RMSE:  0.9640330389483439
R2 score:  0.34910697208409225
---------------------
```

```
test_bay_RMSE, test_bay_R2 = cross_validate(bay_feat,timeseries_cv)
```

```
R2 score:  -0.4028224673902141
---------------------
From 0 to 32
RMSE:  0.6641758480635003

R2 score:  -0.30369446324235905
---------------------
From 33 to 40
RMSE:  1.0784663935690209
R2 score:  -0.1879165918243475
---------------------
From 0 to 40
RMSE:  0.7628891407243539
R2 score:  0.4796361397893819
---------------------
From 41 to 48
RMSE:  0.6374211888624806
R2 score:  -0.5432147063460999
---------------------
From 0 to 48
RMSE:  0.7438515275725354
R2 score:  0.6826896064935606
---------------------
From 49 to 56
RMSE:  0.7037555336934553
R2 score:  0.5106600729724657
---------------------
From 0 to 56
```

```
From 0 to 56
RMSE:  0.7383553852177266
R2 score:  0.7169157403672641
---------------------
From 57 to 64
RMSE:  0.536208764107043
R2 score:  0.4425154565138968
---------------------
From 0 to 64
RMSE:  0.7165598711128549
R2 score:  0.718851047213944
---------------------
From 65 to 72
RMSE:  0.6088460602782845
R2 score:  0.3576919444427197
---------------------
From 0 to 72
RMSE:  0.7055583757751654
R2 score:  0.7393687048722419
---------------------
From 73 to 80
RMSE:  0.45022162211623046
R2 score:  -0.8975136006739859
---------------------
From 0 to 80
RMSE:  0.6845915917844695
R2 score:  0.8174153468574218
---------------------
From 81 to 88
RMSE:  0.9034141487529068
R2 score:  0.4283903404688256
---------------------
```

```
test_ensemble_RMSE, test_ensemble_R2 = cross_validate(EnsembleSolution(ensemble_sol
```

```
From 0 to 8
RMSE:  0.8958126872467559
R2 score:  -0.7269094321656082
---------------------
From 9 to 16
RMSE:  0.5502091570144318
R2 score:  -1.395050306063352
---------------------
From 0 to 16
RMSE:  0.7531957119067643
R2 score:  -0.8238441368589391
---------------------
From 17 to 24
RMSE:  0.4676036274188001
R2 score:  -0.9730906095253504
---------------------
From 0 to 24
RMSE:  0.6750819057386244
R2 score:  -0.8284451395731807
---------------------
From 25 to 32
RMSE:  0.6485775205687428
R2 score:  -0.3529339513208274
---------------------
From 0 to 32
```

```
RMSE:  0.6687530659887596
R2 score:  -0.3217253980038164
---------------------
From 33 to 40
RMSE:  1.1249803273944534
R2 score:  -0.29259529646694116
---------------------
From 0 to 40
RMSE:  0.7790434689034405
R2 score:  0.45736519940503906
---------------------
From 41 to 48
RMSE:  0.5435743377581986
R2 score:  -0.12225423305509064
---------------------
From 0 to 48
RMSE:  0.7456957067942777
R2 score:  0.6811142850553823
---------------------
From 49 to 56
RMSE:  0.6943942605604283
R2 score:  0.5235917731604346
---------------------
From 0 to 56
RMSE:  0.7387104622629416
R2 score:  0.7166434029702649
---------------------
From 57 to 64
RMSE:  0.5666415405151063
R2 score:  0.37743911083079784
---------------------
From 0 to 64
RMSE:  0.7197560533930092
```

```python
print('Benchmark RMSE: {} // Benchmark R2: {}'.format(test_bench__RMSE, test_bench_
print('LSVR RMSE: {} // LSVR R2: {}'.format(test_lsvr__RMSE, test_lsvr_R2))
print('Lasso RMSE: {} // Lasso R2: {}'.format(test_lasso__RMSE, test_lasso_R2))
print('bayesian ridge RMSE: {} // Bayesian ridge R2: {}'.format(test_bay_RMSE, test

print('Ensemble RMSE: {} // Ensemble R2: {}'.format(test_ensemble_RMSE, test_ensemb
```

```
Benchmark RMSE: 1.1902274242440065 // Benchmark R2: -2.4165154102540303
LSVR RMSE: 0.7401323204333284 // LSVR R2: -0.23920364780548428
Lasso RMSE: 0.7566217129287226 // Lasso R2: -0.29827496844569257
bayesian ridge RMSE: 0.693527343546132 // Bayesian ridge R2: -0.0321707992618
Ensemble RMSE: 0.6978649336370706 // Ensemble R2: -0.048808435483526556
```

0s completed at 9:25 PM