FRONTIER TECHNOLOGY INSTITUTE

DATA SCIENCE CERTIFICATION

CAPSTONE PROJECT

Project Title: Predicting Crude Oil Prices Using Time Series Forecasting and Regression Analysis.

Project Manager: Usman Ali
Project Advisor: Dr.Tariq Mahmood
Team Members: Ali Ibad Brohi, Moiz Ali , M Shoaib, Khoulah Afzal Qamar

Acknowledgement

First of all, we would like to thank ALLAH Almighty the most gracious and the most magnificent for giving us the opportunity, strength and potentiality in doing our project. We would also like to give many thanks to our parents for making our career bright and worth full. Without them it would not be possible to achieve our goals.

Secondly, we would like to thank Frontier Technology institute, for providing students with an opportunity, helping us to explore and learn more about Data Science. We would also like to deeply express many thanks to Mr Usman Ali in helping us at every part of our project which was not possible without him. He is a very dedicated person with a pleasant personality and we feel very lucky to have him as our project manager. We especially thank him for his encouragement and his accurate comments which were of critical importance of this work.

Table of Contents

Abstract

Crude oil is the world's leading fuel, and its prices have a big impact on the global environment, economy as well as oil exploration and exploitation activities. Oil price forecasts are very useful to industries, governments and individuals. Although many methods have been developed for predicting oil prices, it remains one of the most challenging forecasting problems due to the high volatility of oil prices. In this project, we propose a novel approach for crude oil price prediction based on a machine learning approaches called time series forecasting and other related approaches .The main advantage of our these learning approach is that the model can predict the crude oil prices using datasets like Crude oil prices, Refinery, Oil companies and Pakistan GDP historical data the program predicting or forecast crude oil prices in order to develop appropriate economic plans in advance and then we makes conclusions regarding the impact of these forecasts on our economy.

Keywords: crude oil, economic geology, prediction model, machine learning, time series forecasting.

Introduction

It is a well-known fact that any minor or major fluctuation in the price of crude oil affects the economy at a global level, particularly countries involved in import and export of crude oil. Products related to crude oil (e.g., petroleum) are also important commodities globally. However, crude oil price has a high volatility level, primarily due to the impact of exogenous variables on this price, e.g., stock and trade market fluctuations, demand and supply KPIS, economic indicators, climate, foreign policy and citizen demographics. Most of these variables are out-of-control, and hence, a consequent practice for economists and financial analysts is to gather information about these variables and model it collectively in order to understand the inherent values and patterns.

This activity is complicated: any change in the values of the variables has a significant impact on the crude oil price. This becomes critical for the petroleum use case: price of crude oil contributes more than 50% to the average petroleum price. Hence, it becomes important for analysts to model the irregular and complex effect of the exogenous variables on crude oil price movements.

Objectives:

- Predict and Forecast crude oil prices.
- Develop appropriate economic plans.
- Analyse historical data of related companies.

Literature Review:

Heuristic approaches for oil price prediction include professional and survey forecasts, which are mainly based on professional knowledge, judgments, opinion and intuition. Another heuristic approach, the so-called no-change forecast, uses the current price of oil as the best prediction of future oil prices. Despite its simplicity, the no-change forecast appeared to be a good baseline approach for oil price prediction and was better than other heuristic judgmental approaches (Alquist et al., 2013).

Econometric models are the most widely used approaches for oil price prediction, which include autoregressive moving average (ARMA) models and vector autoregressive (VAR) models, with possibly different input variables (Pindyck, 1999; Frey et al., 2009). These econometric models provide more accurate prediction than the no-change model at least at some horizons (Alguist et al., 2013; Baumeister and Kilian, 2015). Recently, a forecast combination approach was proposed by Baumeister and Kilian (2015), which combines 6 different oil price prediction models including both econometric models (such as the VAR model) and the no-change model. It should be noted that most of the econometric models are linear models and are not be able to capture the nonlinearity of oil prices.

Several machine learning techniques were proposed for oil price prediction, such as artificial neural networks (ANN) (Yu et al., 2008; Kulkarni and Haidar, 2009), and support vector machine (SVM) (Xie et al., 2006). These are nonlinear models which may produce more accurate predictions if the oil price data are strongly nonlinear (Behmiri and Pires Manso, 2013). However, these machine learning techniques, like other traditional machine learning techniques, rely on a fixed set of training data to train a machine learning model and then apply the model to a test set. Such an approach works well if the training data and the test data are generated from a stationary process, but may not be effective for non-stationary time series data such as oil price data.

Methodology:

Datasets:

The following datasets used to train model and Predict Crude oil prices.

- Crude oil historical data.
- Attock Refinery Limited.
- Byco Petroleum Pakistan Limited.
- Pakistan Refinery Limited.
- Pakistan State Oil.
- Pakistan Oilfields Limited.
- Oil and Gas Development Company Limited.
- World Texas Intermediate Historical Data.
- Brent Crude Oil.
- KSE100 Historical Data.
- USD to PKR Data.
- Gold Price in PKR.

Models and Approaches:

Project are divided in two approaches.

1. Regression Analysis.
2. Time series.

1. Regression Analysis:

In Regression part work has been divided in two parts. In first we treated Change%_WTI as target variable but we don't get good result so after that we treated Price_WTI as target variable and we get good result as compare to Change%_WTI.

- **Prediction based on Change%**

1. Datasets.

```
ATOR =read_data('ATOR Historical Data.csv')
PKOL=read_data("PKOL Historical Data.csv")
PKRF=read_data("PKRF Historical Data.csv")
PSO=read_data("PSO Historical Data.csv")
WTI=read_data('Crude Oil WTI Futures Historical Data.csv')
OGDC=read_data('OGDC Historical Data.csv')
BRENT=read_data('Brent Oil Futures Historical Data.csv')
KSE100=read_data('Karachi 100 Historical Data.csv')
USDPKR=read_data('USD to PKR Data.csv')
GOLD_USD=read_data('Gold Price in PKR.csv')
```

❖ To make our work more easy I am making a function to import all different files and then calling them by sequence. I also making function for Removing columns Resampling and more.

2. Preview Datasets:

```
PKRF.head()
```

Date	Price	Open	High	Low	Vol.	Change %
2020-01-31	21.65	22.64	22.85	21.26	1.22M	-2.74%
2020-01-30	22.26	22.60	22.60	22.17	689.00K	-1.07%
2020-01-29	22.50	22.30	23.34	22.30	2.08M	0.67%
2020-01-28	22.35	22.30	22.60	22.06	326.00K	0.22%
2020-01-27	22.30	23.00	23.10	22.25	669.50K	-2.19%

```
PKOL.head(6)
```

Date	Price	Open	High	Low	Vol.	Change %
2020-01-31	424.34	434.93	436.00	423.0	186.80K	-1.11%
2020-01-30	429.11	433.00	434.90	429.0	138.30K	-1.24%
2020-01-29	434.50	437.00	441.00	433.1	56.00K	0.03%
2020-01-28	434.35	441.00	441.00	433.0	121.10K	-0.95%
2020-01-27	438.50	441.65	443.48	435.5	165.30K	-1.82%
2020-01-24	446.62	453.00	454.50	443.0	257.40K	-1.63%

```
PSO.head()
```

Date	Price	Open	High	Low	Vol.	Change %
2020-01-31	207.80	211.00	212.15	207.06	1.58M	-0.82%
2020-01-30	209.51	210.98	210.99	208.50	1.09M	-0.47%
2020-01-29	210.50	212.20	213.49	210.01	1.68M	-0.66%
2020-01-28	211.90	211.10	212.98	211.00	739.40K	-0.28%
2020-01-27	212.50	214.50	215.00	211.25	601.10K	-0.43%

```
WTI.head()
```

Date	Price	Open	High	Low	Vol.	Change %
2020-02-28	44.76	46.49	47.03	43.85	1.10M	-4.95%
2020-02-27	47.09	48.63	48.78	45.88	1.01M	-3.37%
2020-02-26	48.73	50.08	50.44	48.30	884.48K	-2.34%
2020-02-25	49.90	51.37	52.02	49.69	764.99K	-2.97%
2020-02-24	51.43	52.60	52.64	50.45	765.52K	-3.65%

3. Resampling:

Fill Missing value by Resampling

```
def AddMissingDays(ds):
    ds = ds.resample('D').bfill().reset_index()
    ds.set_index('Date', inplace=True)
    return ds
```

```
ATOR=AddMissingDays(ATOR)
PKOL=AddMissingDays(PKOL)
PKRF=AddMissingDays(PKRF)
PSO=AddMissingDays(PSO)
WTI=AddMissingDays(WTI)
OGDC=AddMissingDays(OGDC)
BRENT=AddMissingDays(BRENT)
KSE100=AddMissingDays(KSE100)
USDPKR=AddMissingDays(USDPKR)
#GOLD_USD=AddMissingDays(GOLD_USD)
```

- We have All data with missing values of Saturday and Sunday so I use the technique of Resampling that fill Data with Daily basis. For e.g: If there is a missing value of sat and sun of first week so It will automatically fill that value with that first week Friday.

4. Removing extra columns:

Remove All Useless Columns

```
def Removecols(ds):
    ds=ds.drop(ds.columns[[0,1,2,3,4]],axis=1)
    return ds
```

```
ATOR=Removecols(ATOR)
PKOL=Removecols(PKOL)
PKRF=Removecols(PKRF)
PSO=Removecols(PSO)
WTI=Removecols(WTI)
OGDC=Removecols(OGDC)
BRENT=Removecols(BRENT)
KSE100=Removecols(KSE100)
USDPKR=Removecols(USDPKR)
GOLD_USD=Removecols(GOLD_USD)
```

❖ There are some data which do not occur same index as other data so I add some fake columns to match the column indexes and drop all the of the data useless columns with index technique.

5. Merging all datasets:

Merged All Datasets

```
data_frames=[ATOR,PSO,PKRF,PKOL,WTI,OGDC,BRENT,KSE100,USDPKR,GOLD_USD]
ds_merged = reduce(lambda  left,right: pd.merge(left,right,on=['Date'],
                                            how='outer'), data_frames)
```

```
ds_merged.head()
```

Date	Change%_ATOR	Change%_PSO	Change%_PKRF	Change%_PKOL	Change%_WTI	Change%_OGDC	Change%_BRENT	Change%_KSE100	C
2004-02-06	1.41	-0.40	1.29	0.45	-1.81	0.19	-1.50	0.97	
2004-02-07	-4.75	-1.39	-2.48	0.94	1.08	-1.49	0.97	-0.82	
2004-02-08	-4.75	-1.39	-2.48	0.94	1.08	-1.49	0.97	-0.82	
2004-02-09	-4.75	-1.39	-2.48	0.94	1.08	-1.49	0.97	-0.82	
2004-02-10	7.50	1.45	0.58	0.20	3.17	1.32	3.19	1.23	

6. Construct more features and Label encoding:

- For more features I extract some features through ‘Data’ column for example: Year, Month and WeekDay. After Extracting features there is one column ‘WeekDay’ which is in object like Monday,Tuesday etc so I label encoding that column.

Construct features like Year, Month, Week, and Weekday to have some more features.

Also Label Encoding of WeekDay

```
def extractfeatures(ds_merged):
    ds_merged['Year']=ds_merged.index.year
    ds_merged['Month']=ds_merged.index.month
    ds_merged['WeekDay']=ds_merged.index.weekday_name
    labelencoder=LabelEncoder()
    ds_merged.WeekDay=labelencoder.fit_transform(ds_merged.WeekDay)
    return ds_merged
```

```
ds_merged=extractfeatures(ds_merged)
```

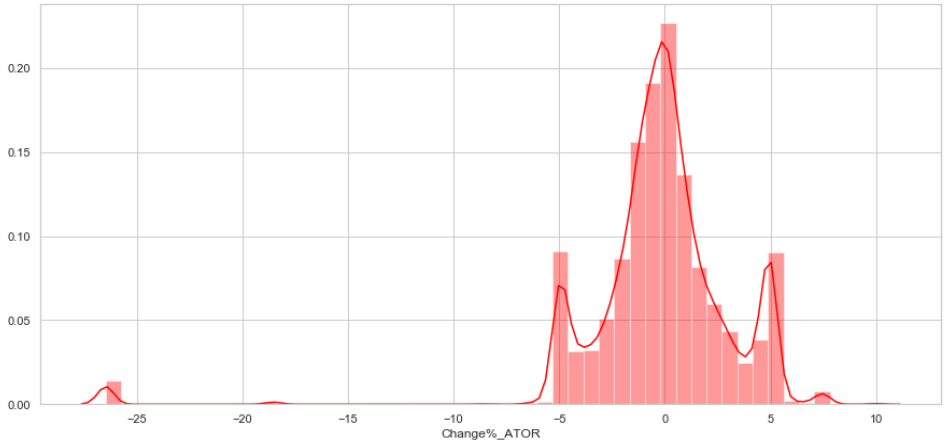
```
ds_merged.head()
```

e%_PKRF	Change%_PKOL	Change%_WTI	Change%_OGDC	Change%_BRENT	Change%_KSE100	Change%_USDPKR	Change%_GOLD	Year	Month	WeekDay
1.29	0.45	-1.81	0.19	-1.50	0.97	0.28	0.01	2004	2	0
-2.48	0.94	1.08	-1.49	0.97	-0.82	0.11	NaN	2004	2	2
-2.48	0.94	1.08	-1.49	0.97	-0.82	0.11	NaN	2004	2	3
-2.48	0.94	1.08	-1.49	0.97	-0.82	0.11	0.00	2004	2	1
0.58	0.20	3.17	1.32	3.19	1.23	-0.08	0.01	2004	2	5

7. Handling Outliers:

Checking Variations for outliers

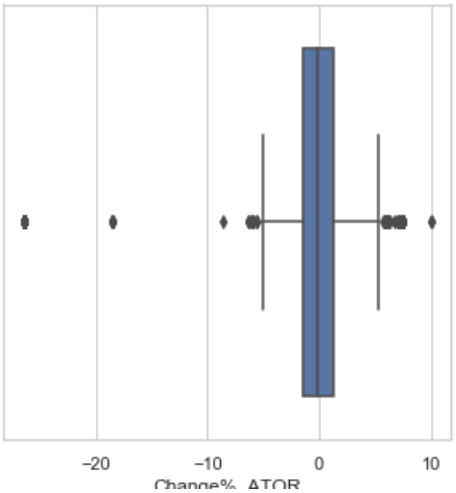
```
for col in ds_merged:
  plt.rcParams['figure.figsize'] = (15, 7)
  sns.distplot(ds_merged[col], color = 'red')
  plt.show()
  print('\n')
```

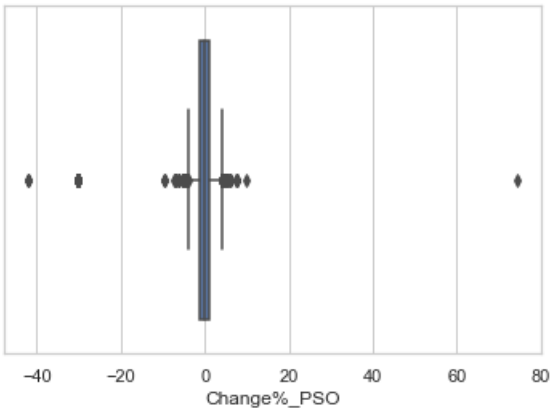


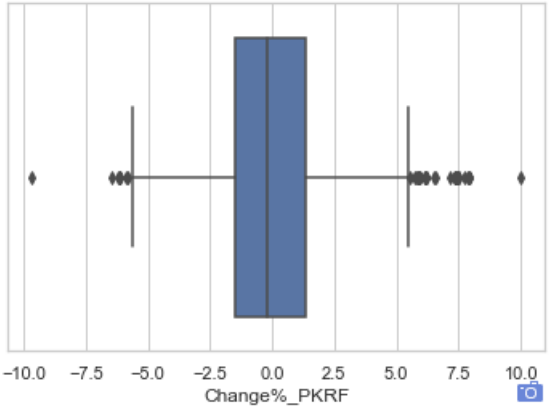
- Plotting Box Plot:

Plotting Box-Plots to checking Outliers

```
sns.set(style="whitegrid")
fig = plt.figure(figsize = (5,5))
for num in ds_merged:
    sns.boxplot(data = ds_merged, x = num)
    plt.show()
```







- Quartile technique to detect outliers:

Using Quartiles Technique to detect Outliers

```
Q1 = ds_merged.quantile(0.25)
Q3 = ds_merged.quantile(0.75)
```

```
IQR = Q3 - Q1
IQR
```

```
Change%_ATOR      2.76
Change%_PSO       2.02
Change%_PKRF      2.82
Change%_PKOL      1.82
Change%_WTI       2.36
Change%_OGDC      1.74
Change%_BRENT     2.12
Change%_KSE100    1.23
Change%_USDPKR    0.13
Change%_GOLD      0.00
Year              8.00
Month             7.00
WeekDay           4.00
dtype: float64
```

Total number of Outliers

In [95]:
```
outliers.count().sum()
```

Out[95]: 4791

Replacing Outliers with Median ¶

```
for col in ds_merged:
  mean = np.mean(ds_merged[col])
  std = np.std(ds_merged[col])
  f1 = mean - 2 * std
  f2 = mean + 2 * std

  ds_merged[col] = np.where(ds_merged[col].between(f1, f2), ds_merged[col], ds_merged[col].median())
```

Checking Variations after removing Outliers

```
for col in ds_merged:
  plt.rcParams['figure.figsize'] = (15, 7)
  sns.distplot(ds_merged[col], color = 'red')
  plt.show()
  print('\n')
```

Checking BoxPlots after Removing Outliers

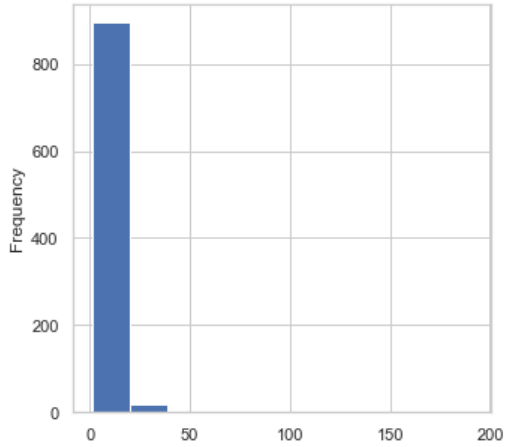
```
sns.set(style="whitegrid")
fig = plt.figure(figsize = (15,7))
for num in ds_merged:
    sns.boxplot(data = ds_merged, x = num)
    plt.show()
```

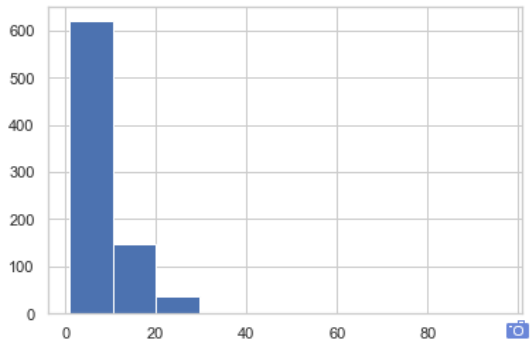
- After Detecting Outliers I was using technique to Replacing Outliers with median, but some times outliers plays an important role. When I apply models the result was not good by using replace technique but when I ignore this code we get good result.

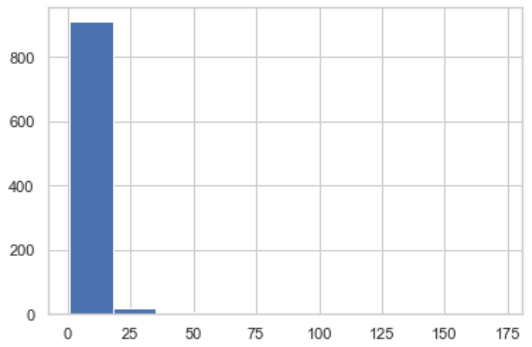
7. Histograms:

Histograms

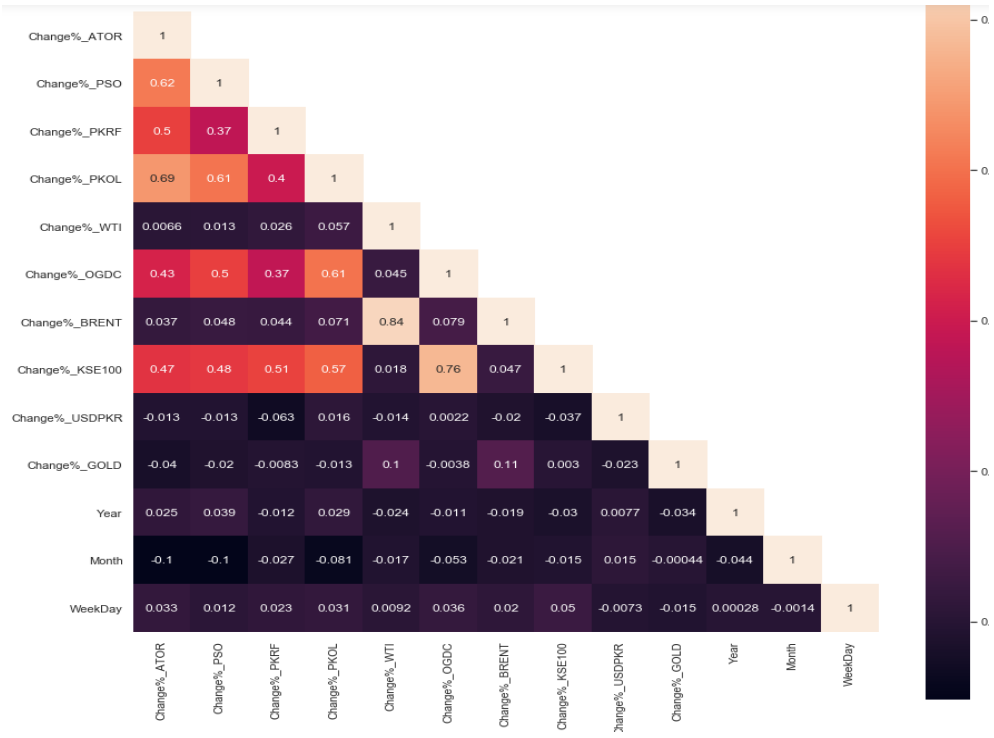
```
fig = plt.figure(figsize=(5,5))
for col in ds_merged:
    ds_merged[col].value_counts().sort_index().plot.hist()
    plt.show()
    print("\n")
```







8. Heat Map for Selecting Best Features:



9. Splitting Data:

Seperating TARGET

```
target = ds_merged['Change%_WTI'].values
target
```

```
array([-1.81,  1.08,  1.08, ..., -2.52, -2.93, -0.06])
```

Seperating FEATURES

```
features = (ds_merged.drop(['Change%_WTI'], axis=1)).values
features
```

```
array([[ 1.410e+00, -4.000e-01,  1.290e+00, ...,  2.004e+03,  2.000e+00,
         0.000e+00],
       [-4.750e+00, -1.390e+00, -2.480e+00, ...,  2.004e+03,  2.000e+00,
         2.000e+00],
       [-4.750e+00, -1.390e+00, -2.480e+00, ...,  2.004e+03,  2.000e+00,
         3.000e+00],
       ...,
       [-1.100e-01, -8.000e-02, -2.000e-01, ...,  2.004e+03,  2.000e+00,
         5.000e+00],
       [-1.100e-01, -8.000e-02, -2.000e-01, ...,  2.004e+03,  2.000e+00,
         6.000e+00],
       [-1.100e-01, -8.000e-02, -2.000e-01, ...,  2.004e+03,  2.000e+00,
         4.000e+00]])
```

Spilitting Data Into 70:30

```
X_train, X_test, y_train, y_test = train_test_split(features,target, test_size=0.3, random_state=0)
```

❖ Making ‘Change%_WTI’ as target variable and all other as features. I also selecting best feature but this technique made our result more down so I select all features.

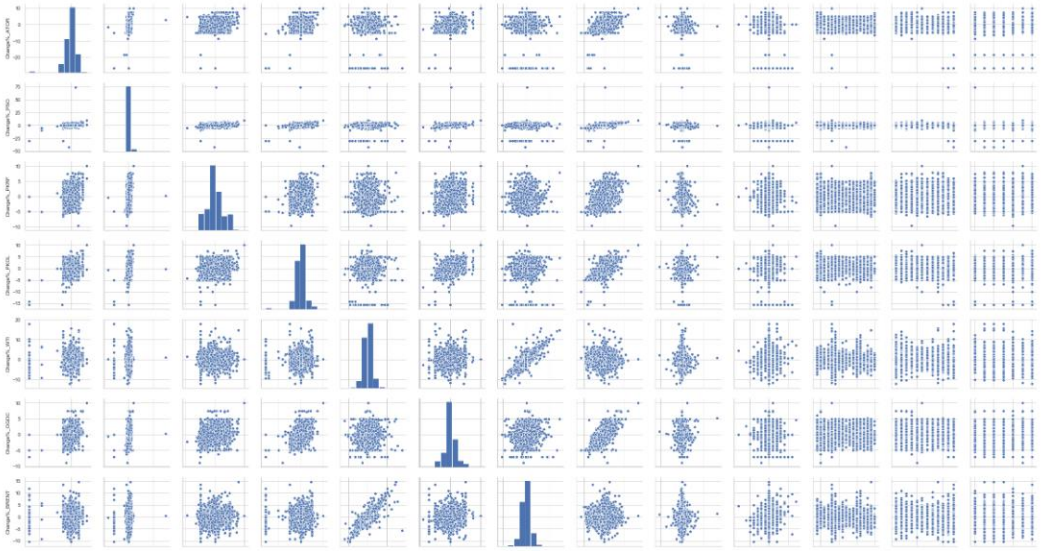
10.Verifying Assumption to Plot Linear Model:

Verifying Assumption

1) Linear Relationship b/w all features and target

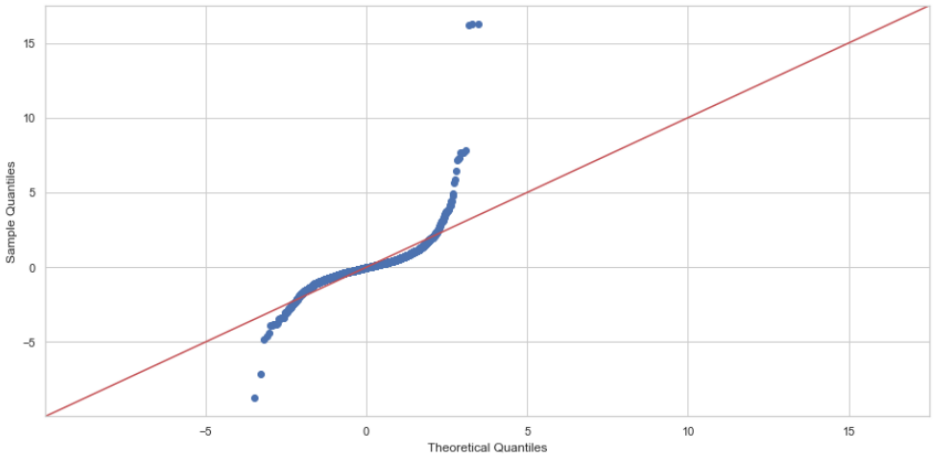
```
In [102]: sns.pairplot(ds_merged)
```

```
Out[102]: <seaborn.axisgrid.PairGrid at 0x142a81aafd0>
```



4) Normal Distribution of error terms

In [104]:
```
import statsmodels.api as sm
mod_fit=sm.OLS(y_train,X_train).fit()
res=mod_fit.resid
fig=sm.qqplot(res,fit=True,line='45')
plt.show()
```



5) Little or No autocorrelation in the residuals

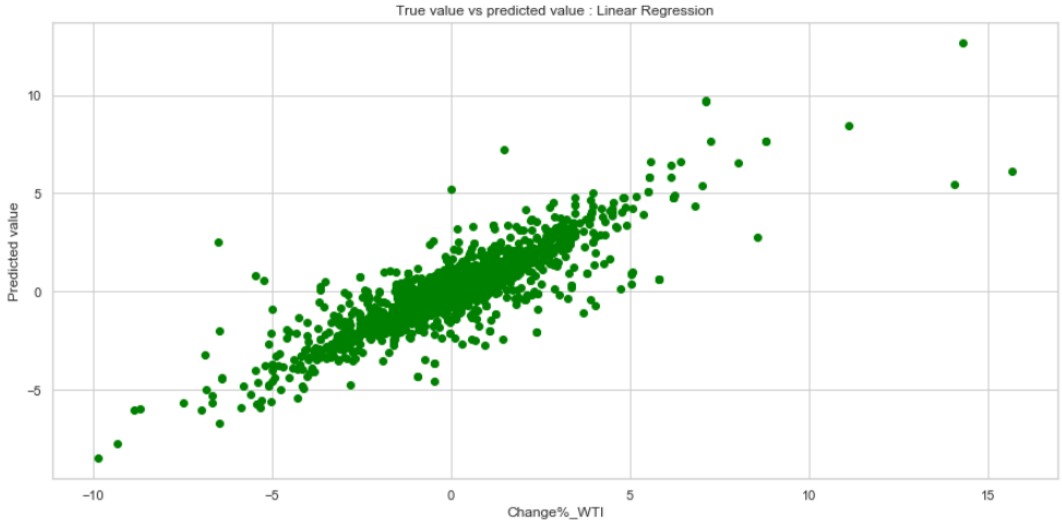
In [105]:
```
model=sm.OLS(y_train,X_train)
results=model.fit()
print(results.summary())
```

```
                            OLS Regression Results
```

11. Multiple Linear Regression Model:

```
y_predLR = LR.predict(X_test)
```

```
plt.scatter(y_test, y_predLR, c = 'green')
plt.xlabel("Change%_WTI")
plt.ylabel("Predicted value")
plt.title("True value vs predicted value : Linear Regression")
plt.show()
```

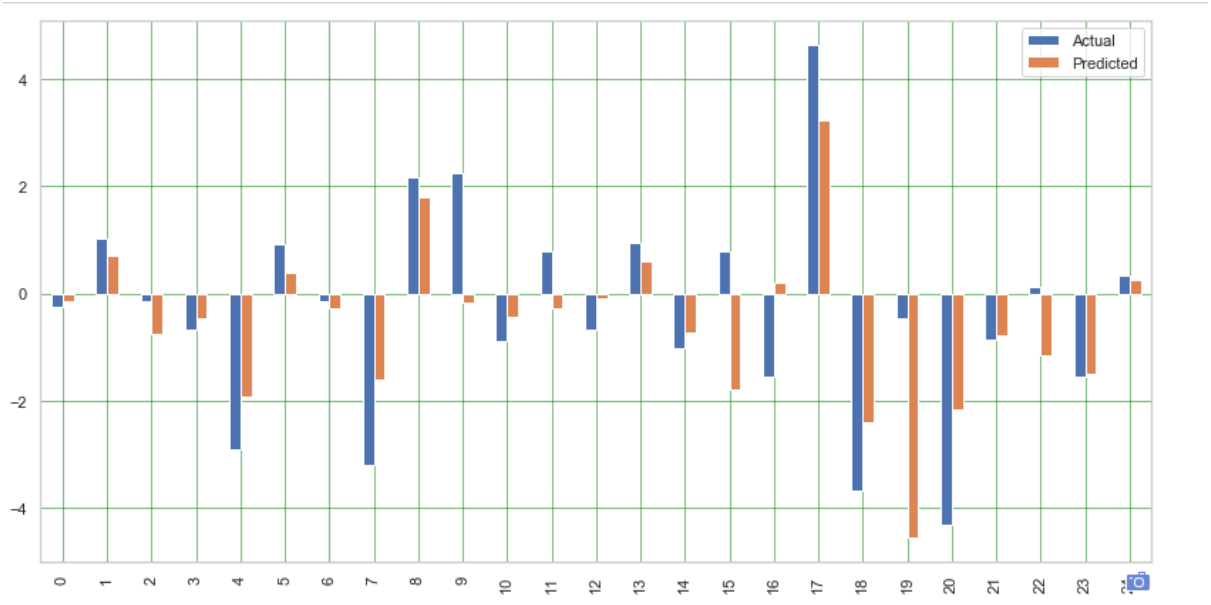


R-Square Score of Linear Regression

```
print('R-Square Score of training data:', LR.score(X_train, y_train))
print('R-Square Score on test data:', LR.score(X_test, y_test))
```

```
R-Square Score of training data: 0.7008110765169387
R-Square Score on test data: 0.7484250921166387
```

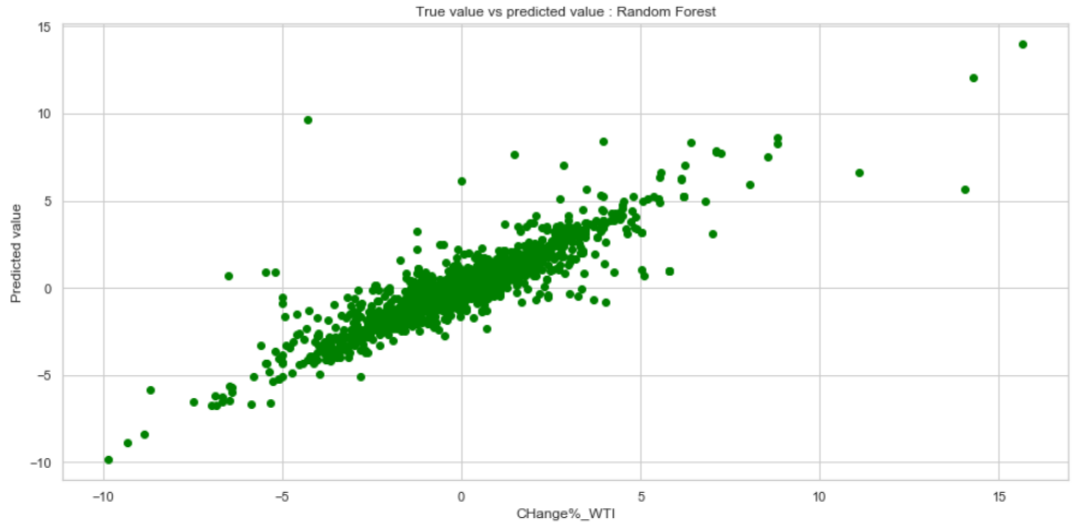
- Actual vs Predicted Bar Chart:



12.Random Forest Regressor:

```
y_predRF= RF.predict(X_test)
```

```
plt.scatter(y_test, y_predRF, c = 'green')
plt.xlabel("CHange%_WTI")
plt.ylabel("Predicted value")
plt.title("True value vs predicted value : Random Forest")
plt.show()
```

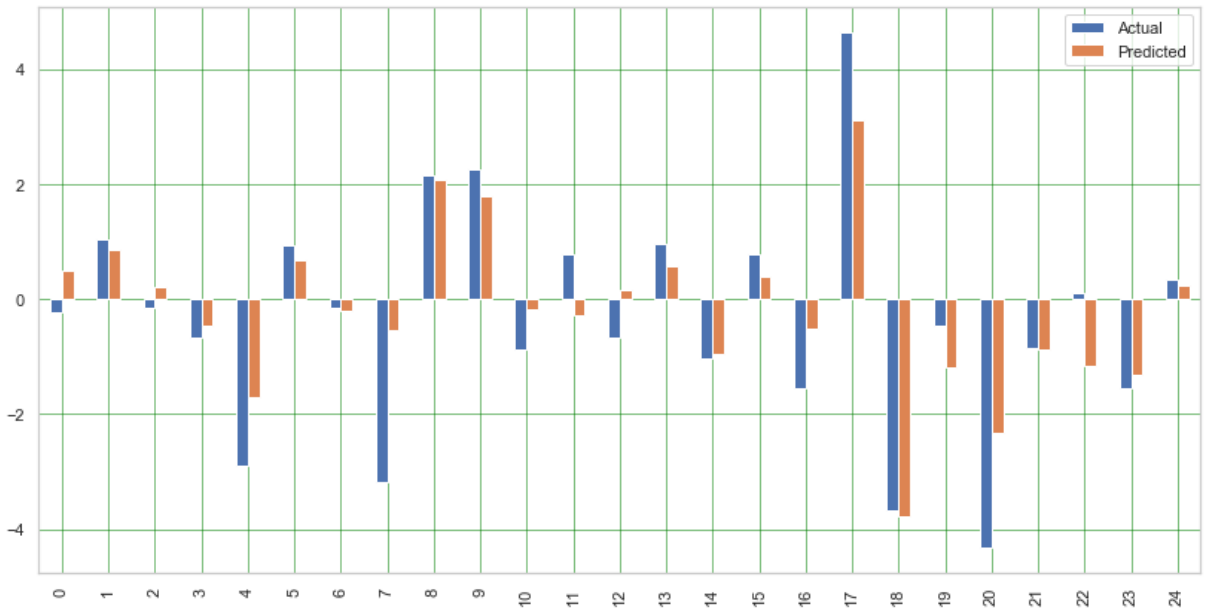


R-Square Score of Random Forest

```
print('R-Square Score on training data:', RF.score(X_train, y_train))
print('R-Square Score on test data:', RF.score(X_test, y_test))
```

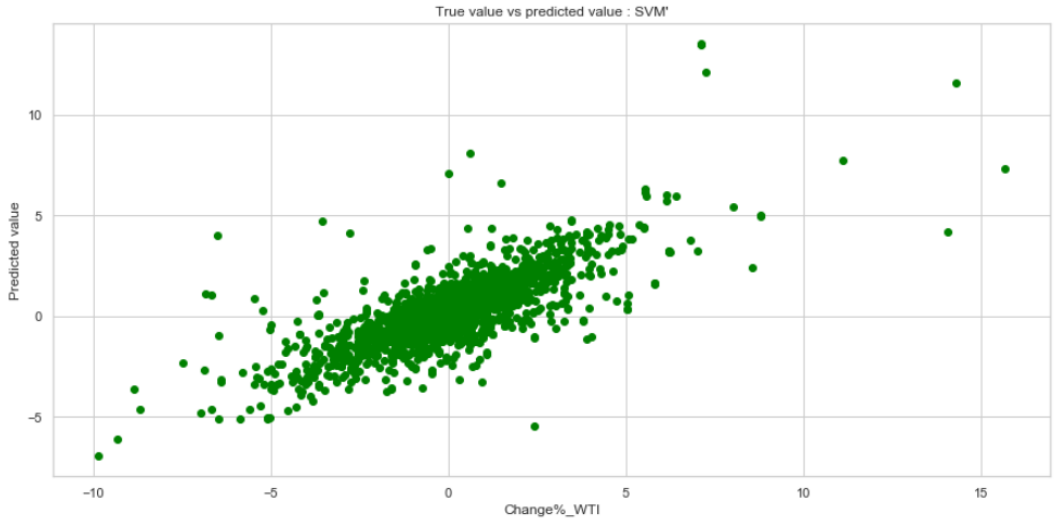
```
R-Square Score on training data: 0.97678812596066
R-Square Score on test data: 0.8037547650324194
```

- Actual vs Predicted Bar Chart:



13.Support Vector Regression:

```
plt.scatter(y_test, y_predSVM, c = 'green')
plt.xlabel("Change%_WTI")
plt.ylabel("Predicted value")
plt.title("True value vs predicted value : SVM'")
plt.show()
```

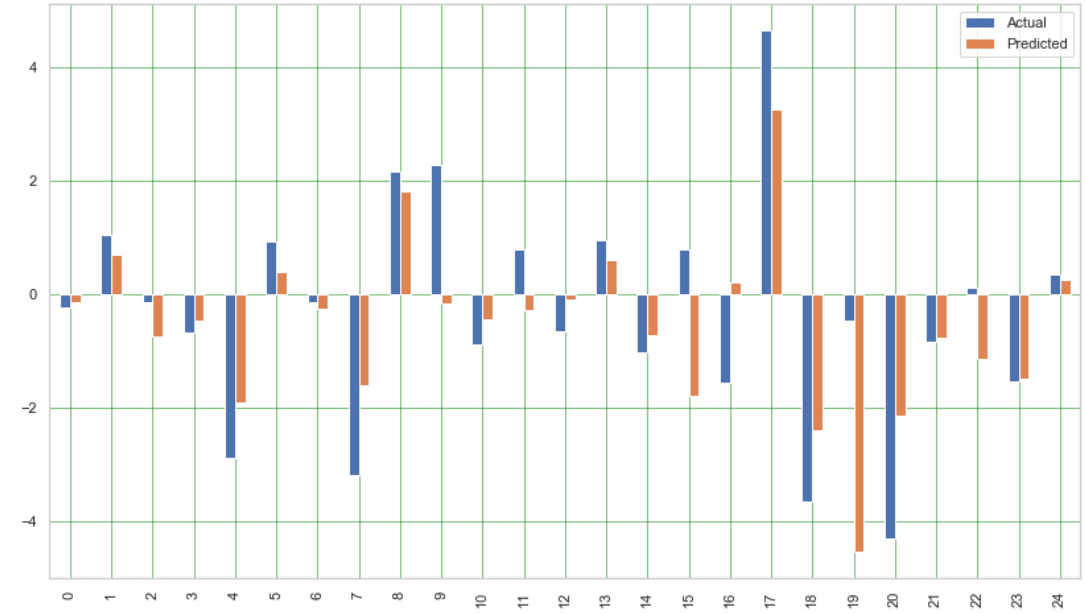


R-Square Score of Support Vector Machine

```
print('R-Square Score of training data:', SVM.score(X_train, y_train))
print('R-Square Score of test data:', SVM.score(X_test, y_test))
```

```
R-Square Score of training data: 0.5742306201054124
R-Square Score of test data: 0.5780790151043597
```

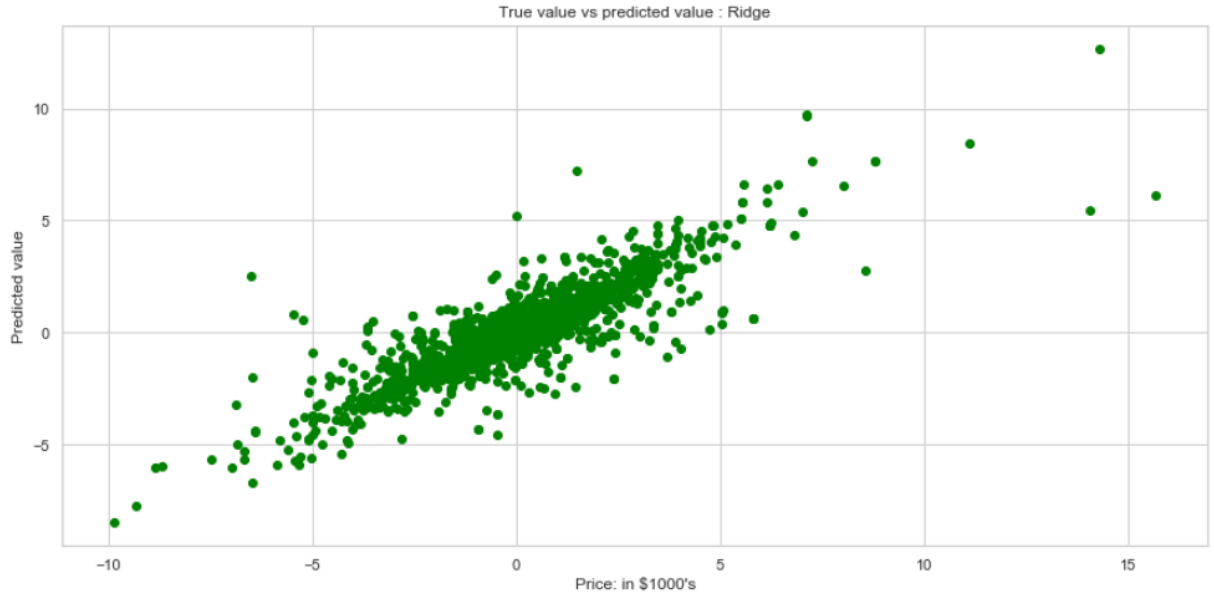
- Actual vs Predicted Bar Chart:



14. Ridge Regression:

```
y_predRI= RI.predict(X_test)
```

```
plt.scatter(y_test, y_predRI, c = 'green')
plt.xlabel("Price: in $1000's")
plt.ylabel("Predicted value")
plt.title("True value vs predicted value : Ridge")
plt.show()
```

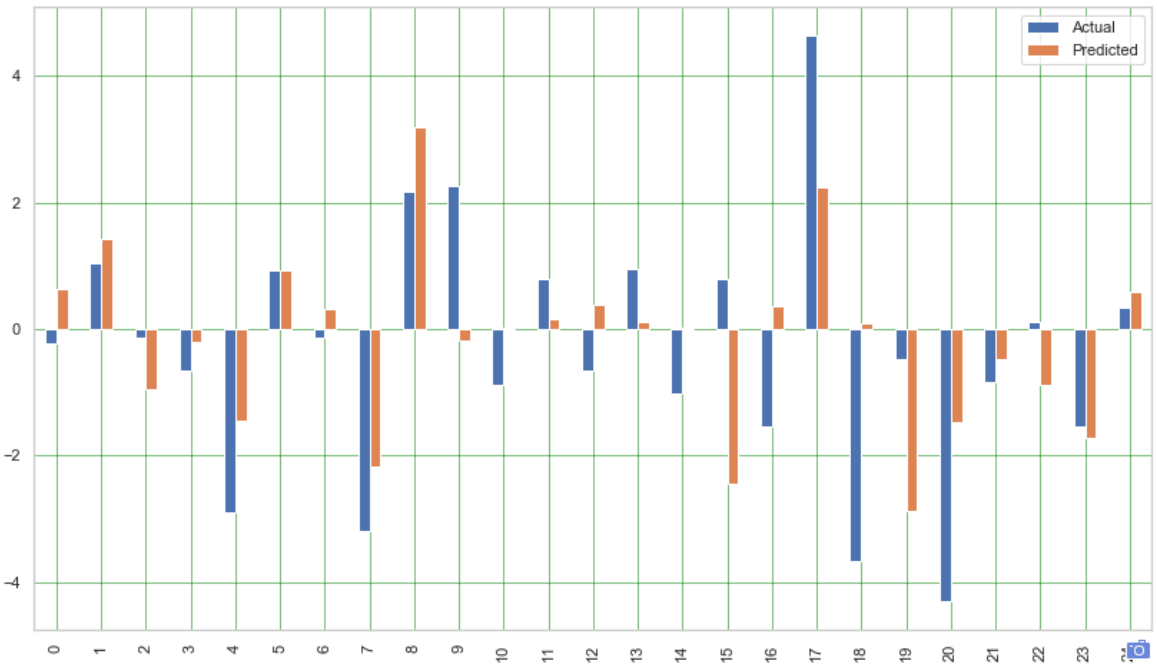


R-Square Score of Ridge

```
print('R-Square Score of training data:', RI.score(X_train, y_train))
print('R-Square Score of  test data:', RI.score(X_test, y_test))
```

```
R-Square Score of training data: 0.7008110235714311
R-Square Score of  test data: 0.7484212580249654
```

- Actual vs Predicted Bar Chart:



15.Tuning of Ridge Regression:

Tuning of Ridge

```
from sklearn.model_selection import RandomizedSearchCV as rs
from scipy.stats import uniform as sp_rand

# SVM with random search
param_grid = {'alpha': sp_rand()}
model = Ridge()
clf_random = rs(estimator=model, param_distributions = param_grid, n_iter = 100, random_state = 0)
clf_random.fit(X_train, y_train)
predictions = clf_random.predict(X_test)
print('Test Score of Ridge (%):', clf_random.score(X_test, y_test) * 100)
print('Train Score of Ridge (%):', clf_random.score(X_train, y_train) * 100)
# examine the best model
# Single best score achieved across all params (min_samples_split)
print('Best Score (%):', clf_random.best_score_ * 100)
# Dictionary  containing the parameters (min_samples_split) used to generate
print('Best Parameters:', clf_random.best_params_)
# Actual model object fit with those best parameters
# Shows default parameters that we didn't specify
print('Best Estimator:', clf_random.best_estimator_)
```

```
Test Score of Ridge (%): 74.8346676414835
Train Score of Ridge (%): 70.07956462845786
Best Score (%): 69.15597728452444
Best Parameters: {'alpha': 0.9883738380592262}
Best Estimator: Ridge(alpha=0.9883738380592262, copy_X=True, fit_intercept=True,
   max_iter=None, normalize=False, random_state=None, solver='auto',
   tol=0.001)
```

Ridge After Tuning

```
from sklearn.linear_model import Ridge
model = Ridge(alpha=0.9883738380592262, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
model.fit(X_train, y_train)
print('R-Square Score on training data:', model.score(X_train, y_train))
print('R-Square Score on test data:', model.score(X_test, y_test))
```

```
R-Square Score on training data: 0.7007956462845786
R-Square Score on test data: 0.7483466764148351
```

16. Lasso Regression:

```
y_predLS= LS.predict(X_test)
```

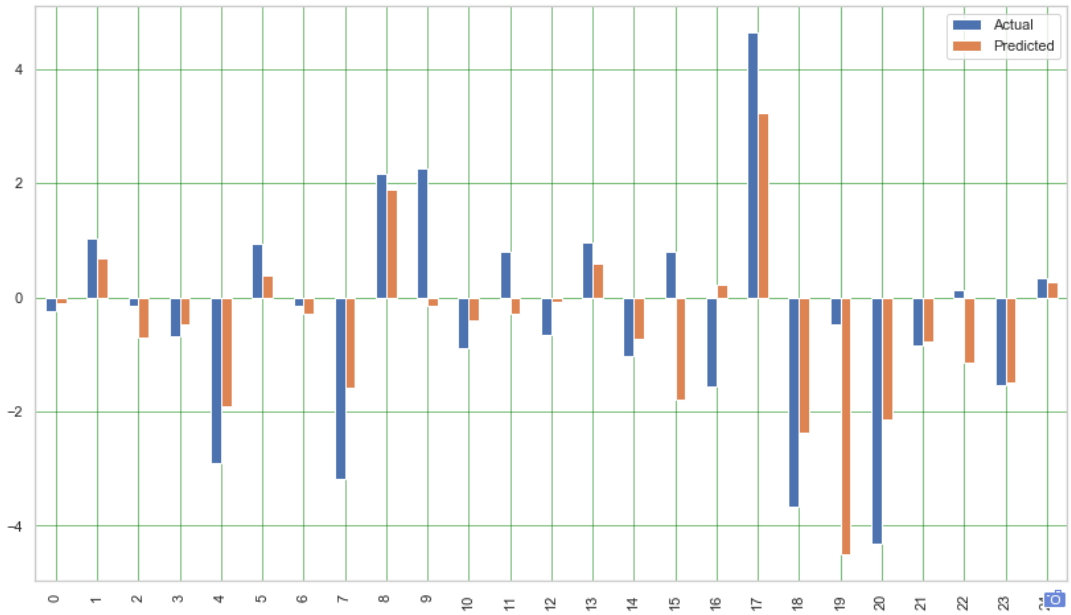
```
plt.scatter(y_test, y_predLS, c = 'green')
plt.xlabel("Price: in $1000's")
plt.ylabel("Predicted value")
plt.title("True value vs predicted value : Lasso")
plt.show()
```

R-Square Score of Lasso

```
print('R-Square Score on training data:', LS.score(X_train, y_train))
print('R-Square Score on test data:', LS.score(X_test, y_test))
```

```
R-Square Score on training data: 0.7007157759906198
R-Square Score on test data: 0.749118372663687
```

- Actual vs Predicted Bar Chart:



17.Tuning of Lasso Regression:

Tuning of Lasso

```
from sklearn.model_selection import RandomizedSearchCV as rs
from scipy.stats import uniform as sp_rand

# SVM with random search
param_grid = {'alpha': sp_rand()}
model = Lasso()
clf_random = rs(estimator=model, param_distributions = param_grid, n_iter = 100, random_state = 0)
clf_random.fit(X_train, y_train)
predictions = clf_random.predict(X_test)
print('Test Score of Lasso (%):', clf_random.score(X_test, y_test) * 100)
print('Train Score of Lasso (%):', clf_random.score(X_train, y_train) * 100)
# examine the best model
# Single best score achieved across all params (min_samples_split)
print('Best Score (%):', clf_random.best_score_ * 100)
# Dictionary  containing the parameters (min_samples_split) used to generate
print('Best Parameters:', clf_random.best_params_)
# Actual model object fit with those best parameters
# Shows default parameters that we didn't specify
print('Best Estimator:', clf_random.best_estimator_)
```

```
Test Score of Lasso (%): 74.97762070658999
Train Score of Lasso (%): 70.06002409152241
Best Score (%): 69.26812867828708
Best Parameters: {'alpha': 0.018789800436355142}
Best Estimator: Lasso(alpha=0.018789800436355142, copy_X=True, fit_intercept=True,
   max_iter=1000, normalize=False, positive=False, precompute=False,
   random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

After Tuning

```
from sklearn.linear_model import Ridge
model = Lasso(alpha=0.09710127579306127, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
model.fit(X_train, y_train)
print('R-Square Score of training data:', model.score(X_train, y_train))
print('R-Square Score of test data:', model.score(X_test, y_test))
```

```
R-Square Score of training data: 0.6981692175457375
R-Square Score of test data: 0.7529729257383562
```

18. Comparison between all the Models Predicted Values:

Actual Value with All Predicted values

```
dsorg = pd.DataFrame({'Actual': y_test.flatten(), 'Predicted_Linear': y_predLR.flatten(),
                      'Predicted_Random_forest': y_predRF.flatten(),'Predicted_SVC': y_predSVM.flatten(),
                     'Predicted_Ridge': y_predRI.flatten(),'Predicted_Lasso': y_predLS.flatten()})
```

```
dsorg.head()
```

	Actual	Predicted_Linear	Predicted_Random_forest	Predicted_SVC	Predicted_Ridge	Predicted_Lasso
0	-0.24	-0.137635	0.49973	0.626821	-0.137617	-0.092137
1	1.03	0.698077	0.86725	1.428331	0.697347	0.687695
2	-0.14	-0.747215	0.20885	-0.959754	-0.747217	-0.708145
3	-0.67	-0.460166	-0.45419	-0.204919	-0.460171	-0.461456
4	-2.89	-1.908549	-1.69350	-1.442053	-1.908647	-1.906353

19. MAE, MSE, MAPE and RMSE of All Models:

- Error Calculation of Linear and Random:

Error Calculation of Multiple Linear Regression ¶

```
print('Multiple Linear Regression:\n')
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_predLR))
print('Mean Absolute Percentage Error:', (metrics.mean_absolute_error(y_test, y_predLR))*100)
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_predLR))
print('Root Mean Squared Error:', math.sqrt(metrics.mean_squared_error(y_test, y_predLR)))
```

```
Multiple Linear Regression:

Mean Absolute Error: 0.7148537693119633
Mean Absolute Percentage Error: 71.48537693119633
Mean Squared Error: 1.2660401220784396
Root Mean Squared Error: 1.1251844835752223
```

Error Calculation of Random Forest Regression

```
print('Random Forest Regression:\n')
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_predRF))
print('Mean Absolute Percentage Error:', (metrics.mean_absolute_error(y_test, y_predRF))*100)
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_predRF))
print('Root Mean Squared Error:', math.sqrt(metrics.mean_squared_error(y_test, y_predRF)))
```

```
Random Forest Regression:

Mean Absolute Error: 0.5647867648725208
Mean Absolute Percentage Error: 56.47867648725208
Mean Squared Error: 0.9875958748273088
Root Mean Squared Error: 0.9937785844076682
```

- Error Calculation of SVR and Ridge Regression:

Error Calculation of Simple Vector Regression

```
print('SVM Regression:\n')
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_predSVM))
print('Mean Absolute Percentage Error:', (metrics.mean_absolute_error(y_test, y_predSVM))*100)
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_predSVM))
print('Root Mean Squared Error:', math.sqrt(metrics.mean_squared_error(y_test, y_predSVM)))
```

```
SVM Regression:

Mean Absolute Error: 1.0058667894936668
Mean Absolute Percentage Error: 100.58667894936669
Mean Squared Error: 2.1232995759354143
Root Mean Squared Error: 1.4571546163449554
```

Error Calculation of Ridge Regression

```
print('Ridge Regression:\n')
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_predRI))
print('Mean Absolute Percentage Error:', (metrics.mean_absolute_error(y_test, y_predRI))*100)
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_predRI))
print('Root Mean Squared Error:', math.sqrt(metrics.mean_squared_error(y_test, y_predRI)))
```

```
Ridge Regression:

Mean Absolute Error: 0.7148505874339722
Mean Absolute Percentage Error: 71.48505874339722
Mean Squared Error: 1.2660594169832096
Root Mean Squared Error: 1.125193057649757
```

- Error Calculation of Lasso Regression:

Error Calculation of Lasso Regression

```
print('Lasso Regression:\n')
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_predLS))
print('Mean Absolute Percentage Error:', (metrics.mean_absolute_error(y_test, y_predLS))*100)
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_predLS))
print('Root Mean Squared Error:', math.sqrt(metrics.mean_squared_error(y_test, y_predLS)))
```

```
Lasso Regression:

Mean Absolute Error: 0.7131413371595879
Mean Absolute Percentage Error: 71.31413371595879
Mean Squared Error: 1.2625512169415785
Root Mean Squared Error: 1.1236330437209376
```

- **Prediction based on Price values:**

1. Datasets:

```
ATOR =read_data('ATOR Historical Data.csv')
PKOL=read_data("PKOL Historical Data.csv")
PKRF=read_data("PKRF Historical Data.csv")
PSO=read_data("PSO Historical Data.csv")
WTI=read_data('Crude Oil WTI Futures Historical Data.csv')
OGDC=read_data('OGDC Historical Data.csv')
BRENT=read_data('Brent Oil Futures Historical Data.csv')
KSE100=read_data('Karachi 100 Historical Data.csv')
USDPKR=read_data('USD to PKR Data.csv')
GOLD_USD=read_data('Gold Price in PKR.csv')
```

❖ All the below work is same mentioned above except I make a 'Price_WTI' as Target variable and the Models Results and its Error Calculations.

❖ One more thing I want to mention I am not including the SVR Model because it was totally overfitted and I also change its different parameters but there is no effect on result.

2. Preview Datasets:

```
PKRF.head()
```

Date	Price	Open	High	Low	Vol.	Change %
2020-01-31	21.65	22.64	22.85	21.26	1.22M	-2.74%
2020-01-30	22.26	22.60	22.60	22.17	689.00K	-1.07%
2020-01-29	22.50	22.30	23.34	22.30	2.08M	0.67%
2020-01-28	22.35	22.30	22.60	22.06	326.00K	0.22%
2020-01-27	22.30	23.00	23.10	22.25	669.50K	-2.19%

```
PKOL.head(6)
```

Date	Price	Open	High	Low	Vol.	Change %
2020-01-31	424.34	434.93	436.00	423.0	186.80K	-1.11%
2020-01-30	429.11	433.00	434.90	429.0	138.30K	-1.24%
2020-01-29	434.50	437.00	441.00	433.1	56.00K	0.03%
2020-01-28	434.35	441.00	441.00	433.0	121.10K	-0.95%
2020-01-27	438.50	441.65	443.48	435.5	165.30K	-1.82%
2020-01-24	446.62	453.00	454.50	443.0	257.40K	-1.63%

```
PSO.head()
```

Date	Price	Open	High	Low	Vol.	Change %
2020-01-31	207.80	211.00	212.15	207.06	1.58M	-0.82%
2020-01-30	209.51	210.98	210.99	208.50	1.09M	-0.47%
2020-01-29	210.50	212.20	213.49	210.01	1.68M	-0.66%
2020-01-28	211.90	211.10	212.98	211.00	739.40K	-0.28%
2020-01-27	212.50	214.50	215.00	211.25	601.10K	-0.43%

```
WTI.head()
```

Date	Price	Open	High	Low	Vol.	Change %
2020-02-28	44.76	46.49	47.03	43.85	1.10M	-4.95%
2020-02-27	47.09	48.63	48.78	45.88	1.01M	-3.37%
2020-02-26	48.73	50.08	50.44	48.30	884.48K	-2.34%
2020-02-25	49.90	51.37	52.02	49.69	764.99K	-2.97%
2020-02-24	51.43	52.60	52.64	50.45	765.52K	-3.65%

3. Resampling:

Fill Missing value by Resampling

```
def AddMissingDays(ds):
    ds = ds.resample('D').bfill().reset_index()
    ds.set_index('Date', inplace=True)
    return ds
```

```
ATOR=AddMissingDays(ATOR)
PKOL=AddMissingDays(PKOL)
PKRF=AddMissingDays(PKRF)
PSO=AddMissingDays(PSO)
WTI=AddMissingDays(WTI)
OGDC=AddMissingDays(OGDC)
BRENT=AddMissingDays(BRENT)
KSE100=AddMissingDays(KSE100)
USDPKR=AddMissingDays(USDPKR)
#GOLD_USD=AddMissingDays(GOLD_USD)
```

4. Removing extra columns:

Remove All Useless Columns

```
def Removecols(ds):
    ds=ds.drop(ds.columns[[1,2,3,4,5]],axis=1)
    return ds
```

```
ATOR=Removecols(ATOR)
PKOL=Removecols(PKOL)
PKRF=Removecols(PKRF)
PSO=Removecols(PSO)
WTI=Removecols(WTI)
OGDC=Removecols(OGDC)
BRENT=Removecols(BRENT)
KSE100=Removecols(KSE100)
USDPKR=Removecols(USDPKR)
GOLD_PKR=Removecols(GOLD_PKR)
```

5. Merging datasets:

Merged All Datasets

```
data_frames=[ATOR,PSO,PKRF,PKOL,WTI,OGDC,BRENT,KSE100,USDPKR,GOLD_PKR]
ds_merged = reduce(lambda  left,right: pd.merge(left,right,on=['Date'],
                                    how='outer'), data_frames)
```

```
ds_merged.head()
```

Date	Price_ATOR	Price_PSO	Price_PKRF	Price_PKOL	Price_WTI	Price_OGDC	Price_BRENT	Price_KSE100	Price_USDPKR	Price_GOLD_PKR
2004-02-06	22.97	104.67	14.10	104.84	32.48	53.85	28.83	4,888.45	57.160	23,106.90
2004-02-07	21.88	103.21	13.75	105.83	32.83	53.05	29.11	4,848.31	57.225	23,230.50
2004-02-08	21.88	103.21	13.75	105.83	32.83	53.05	29.11	4,848.31	57.225	23,230.50
2004-02-09	21.88	103.21	13.75	105.83	32.83	53.05	29.11	4,848.31	57.225	23,230.50
2004-02-10	23.52	104.71	13.83	106.04	33.87	53.75	30.04	4,907.93	57.180	23,360.90

6. Construct more features and label encoding:

Construct features like Year, Month, Week, and Weekday to have some more features.

Also Label Encoding of WeekDay

```
def extractfeatures(ds_merged):
    ds_merged['Year']=ds_merged.index.year
    ds_merged['Month']=ds_merged.index.month
    ds_merged['WeekDay']=ds_merged.index.weekday_name
    labelencoder=LabelEncoder()
    ds_merged.WeekDay=labelencoder.fit_transform(ds_merged.WeekDay)
    return ds_merged
```

```
ds_merged=extractfeatures(ds_merged)
```

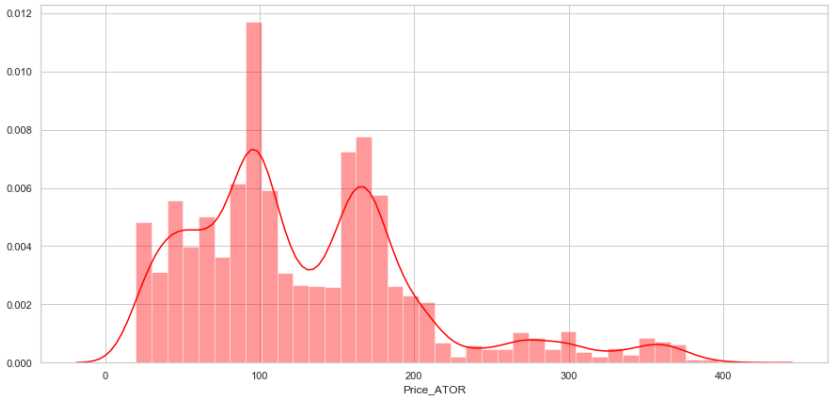
```
ds_merged.head()
```

e_ATOR	Price_PSO	Price_PKRF	Price_PKOL	Price_WTI	Price_OGDC	Price_BRENT	Price_KSE100	Price_USDPKR	Price_GOLD_PKR	Year	Month	WeekDay
22.97	104.67	14.10	104.84	32.48	53.85	28.83	4,888.45	57.160	23,106.90	2004	2	0
21.88	103.21	13.75	105.83	32.83	53.05	29.11	4,848.31	57.225	23,230.50	2004	2	2
21.88	103.21	13.75	105.83	32.83	53.05	29.11	4,848.31	57.225	23,230.50	2004	2	3
21.88	103.21	13.75	105.83	32.83	53.05	29.11	4,848.31	57.225	23,230.50	2004	2	1
23.52	104.71	13.83	106.04	33.87	53.75	30.04	4,907.93	57.180	23,360.90	2004	2	5

7. Handling outliers:

Checking Variations for outliers

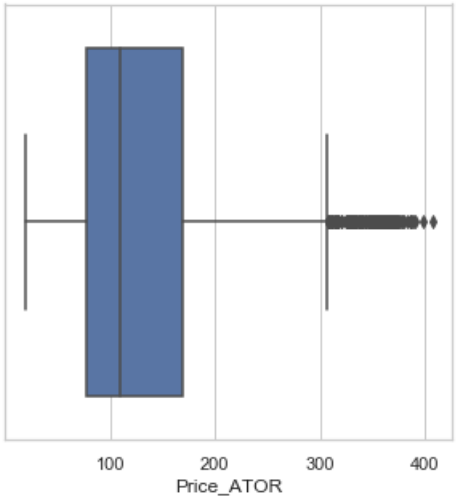
```
for col in ds_merged:
  plt.rcParams['figure.figsize'] = (15, 7)
  sns.distplot(ds_merged[col], color = 'red')
  plt.show()
  print('\n')
```

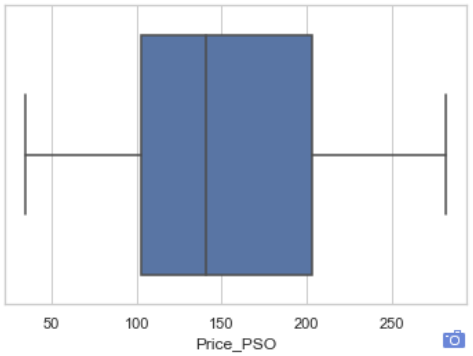


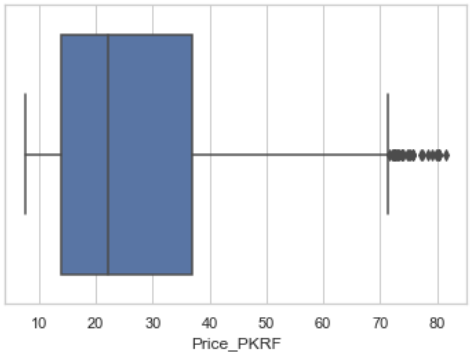
- Plotting Box-Plots:

Plotting Box-Plots to checking Outliers

```
sns.set(style="whitegrid")
fig = plt.figure(figsize = (5,5))
for num in ds_merged:
    sns.boxplot(data = ds_merged, x = num)
    plt.show()
```







- Quartiles technique:

Using Quartiles Technique to detect Outliers

```
Q1 = ds_merged.quantile(0.25)
Q3 = ds_merged.quantile(0.75)
```

```
IQR = Q3 - Q1
IQR
```

```
Price_ATOR           92.0900
Price_PSO           100.2950
Price_PKRF           22.9700
Price_PKOL          195.9700
Price_WTI            37.4300
Price_OGDC           46.5650
Price_BRENT          44.6950
Price_KSE100      23755.8100
Price_USDPKR         42.0825
Price_GOLD_PKR    84435.3500
Year                  8.0000
Month                 7.0000
WeekDay               4.0000
dtype: float64
```

8. Splitting Data:

Seperating Target Variable

```
target = ds_merged['Price_WTI'].values
target
```

```
array([32.48, 32.83, 32.83, ..., 34.1 , 33.1 , 33.08])
```

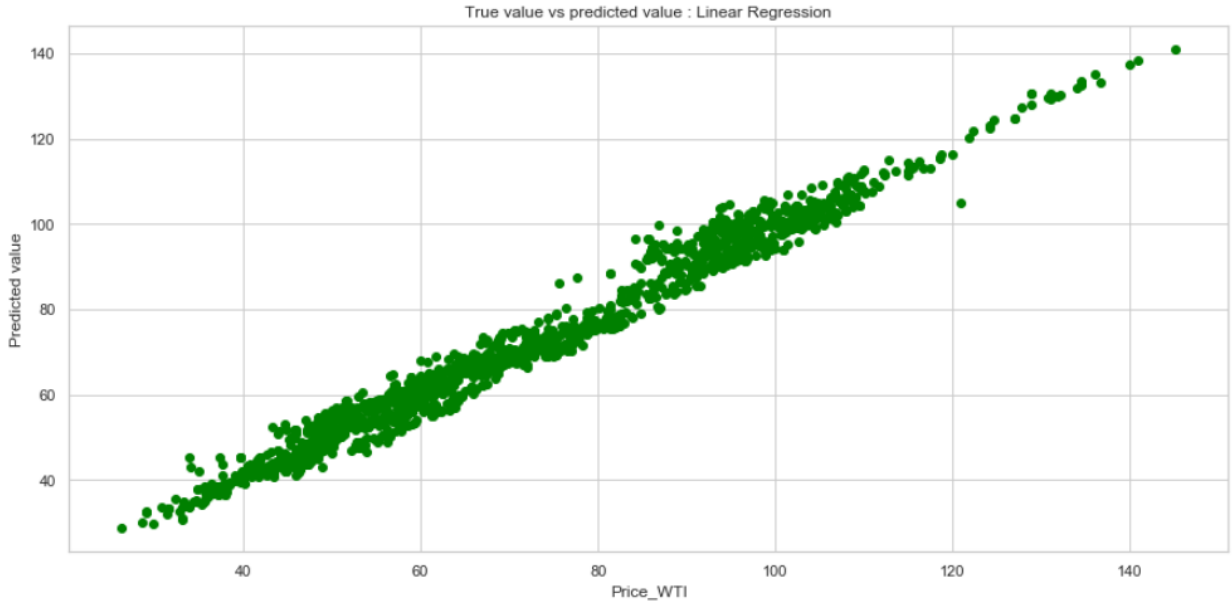
Seperating Features

```
features = (ds_merged.drop(['Price_WTI'], axis=1)).values
features
```

```
array([[2.2970e+01, 1.0467e+02, 1.4100e+01, ..., 2.0040e+03, 2.0000e+00,
        0.0000e+00],
       [2.1880e+01, 1.0321e+02, 1.3750e+01, ..., 2.0040e+03, 2.0000e+00,
        2.0000e+00],
       [2.1880e+01, 1.0321e+02, 1.3750e+01, ..., 2.0040e+03, 2.0000e+00,
        3.0000e+00],
       ...,
       [1.0975e+02, 1.4038e+02, 2.2190e+01, ..., 2.0040e+03, 2.0000e+00,
        5.0000e+00],
       [1.0975e+02, 1.4038e+02, 2.2190e+01, ..., 2.0040e+03, 2.0000e+00,
        6.0000e+00],
       [1.0975e+02, 1.4038e+02, 2.2190e+01, ..., 2.0040e+03, 2.0000e+00,
        4.0000e+00]])
```

9. Linear Regression Model:

```
plt.scatter(y_test, y_predLR, c = 'green')
plt.xlabel("Price_WTI")
plt.ylabel("Predicted value")
plt.title("True value vs predicted value : Linear Regression")
plt.show()
```



R-Square of Linear Regression

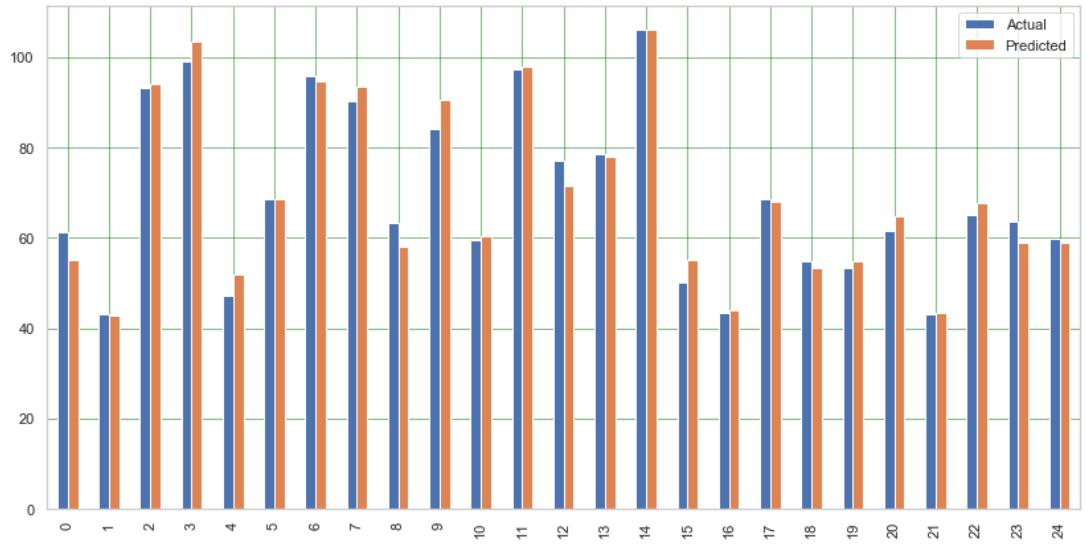
```
print('R-Square Score of training data:', LR.score(X_train, y_train))
print('R-Square Score of test data:', LR.score(X_test, y_test))
```

```
R-Square Score of training data: 0.977624326112349
R-Square Score of test data: 0.9793207142413378
```

- Actual vs Predicted Bar Chart of Linear:

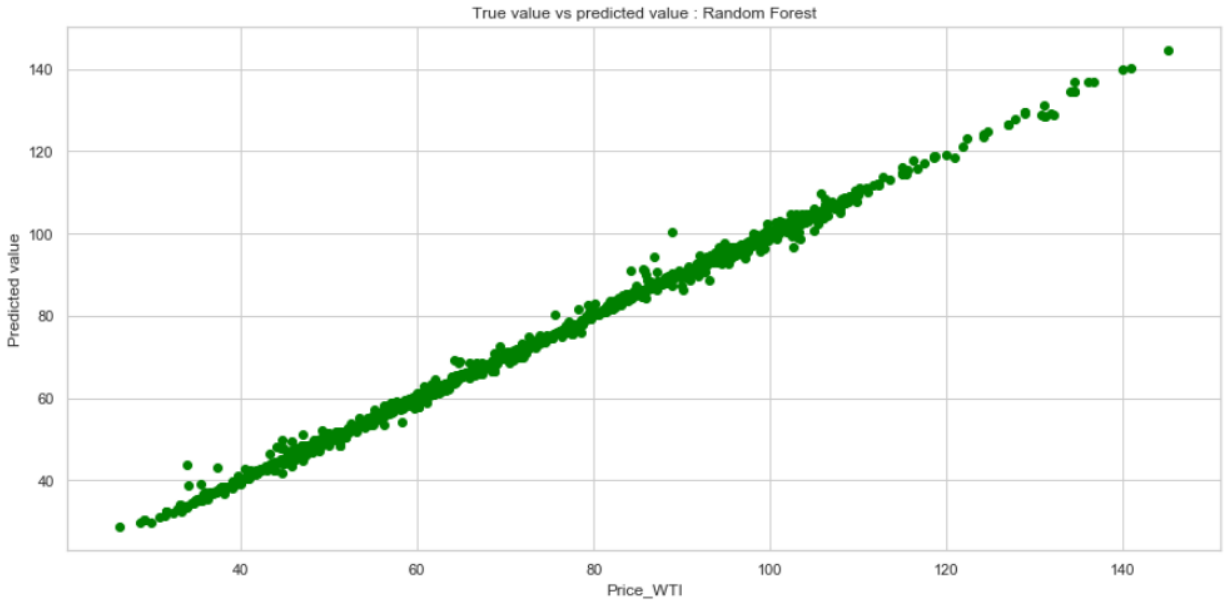
Bar chart of Actual vs Predicted

```
ds = dsLR.head(25)
ds.plot(kind='bar',figsize=(14,7))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



10. Random Forest Model:

```
plt.scatter(y_test, y_predRF, c = 'green')
plt.xlabel("Price_WTI")
plt.ylabel("Predicted value")
plt.title("True value vs predicted value : Random Forest")
plt.show()
```



R-Square Score of Random Forest

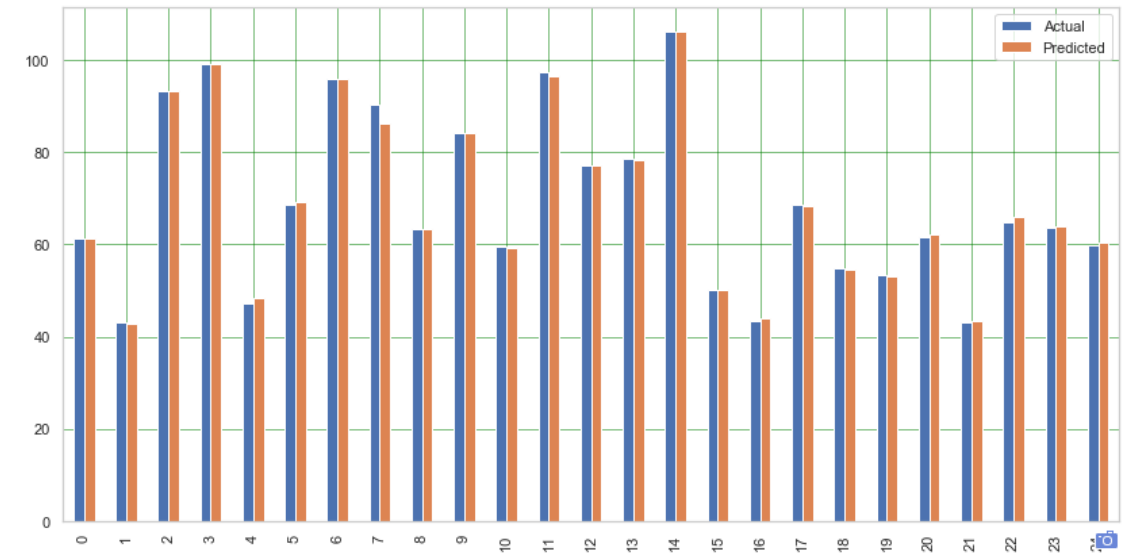
```
print('R-Square Score of training data:', RF.score(X_train, y_train))
print('R-Square Score of test data:', RF.score(X_test, y_test))
```

```
R-Square Score of training data: 0.9997538252737664
R-Square Score of test data: 0.997907141417991
```

Actual vs Predicted Bar Chart of Random:

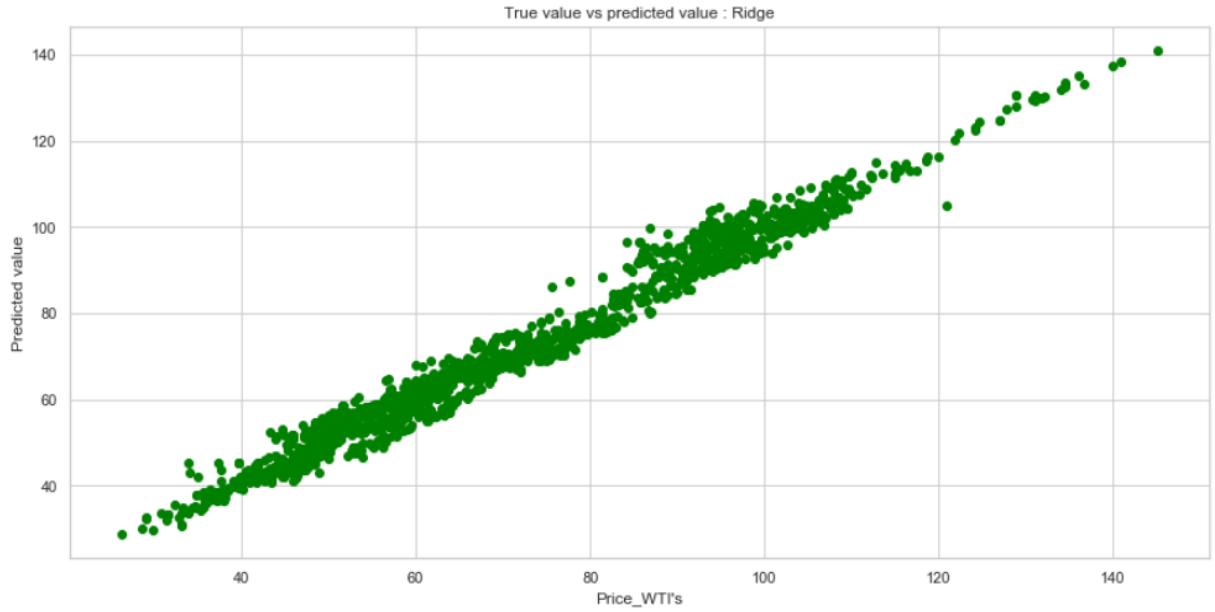
Bar Chart of Actual vs Predicted

```
ds= dsRF.head(25)
ds.plot(kind='bar',figsize=(14,7))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



10.Ridge Regression:

```
plt.scatter(y_test, y_predRI, c = 'green')
plt.xlabel("Price_WTI's")
plt.ylabel("Predicted value")
plt.title("True value vs predicted value : Ridge")
plt.show()
```



R-Square Score of Ridge

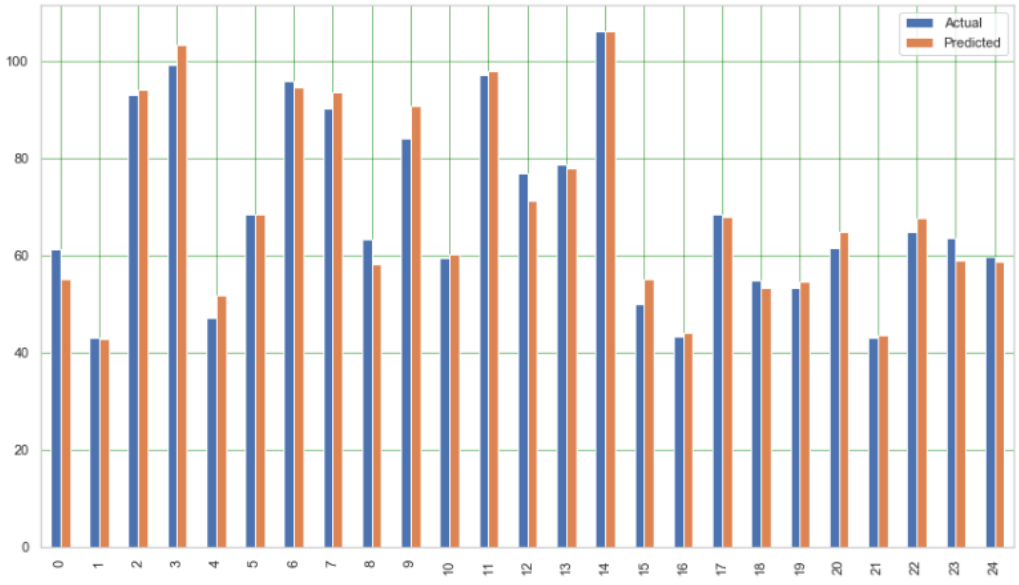
```
print('R-Square Score of training data:', RI.score(X_train, y_train))
print('R-Square Score of test data:', RI.score(X_test, y_test))
```

```
R-Square Score of training data: 0.9776243261123481
R-Square Score of test data: 0.9793207138608213
```

- Actual vs Predicted Value of Ridge:

Bar Chart of Actual vs Predicted

```
ds = dsLS.head(25)
ds.plot(kind='bar',figsize=(14,8))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



11. Tuning of Ridge:

Tuning of Ridge:

```
from sklearn.model_selection import RandomizedSearchCV as rs
from scipy.stats import uniform as sp_rand

# SVM with random search
param_grid = {'alpha': sp_rand()}
model = Ridge()
clf_random = rs(estimator=model, param_distributions = param_grid, n_iter = 100, random_state = 0)
clf_random.fit(X_train, y_train)
predictions = clf_random.predict(X_test)
print('Test Score of Ridge (%):', clf_random.score(X_test, y_test) * 100)
print('Train Score of Ridge (%):', clf_random.score(X_train, y_train) * 100)
# examine the best model
# Single best score achieved across all params (min_samples_split)
print('Best Score (%):', clf_random.best_score_ * 100)
# Dictionary  containing the parameters (min_samples_split) used to generate
print('Best Parameters:', clf_random.best_params_)
# Actual model object fit with those best parameters
# Shows default parameters that we didn't specify
print('Best Estimator:', clf_random.best_estimator_)
```

```
Test Score of Ridge (%): 97.93206766413454
Train Score of Ridge (%): 97.76243261041962
Best Score (%): 97.74481920292799
Best Parameters: {'alpha': 0.9883738380592262}
Best Estimator: Ridge(alpha=0.9883738380592262, copy_X=True, fit_intercept=True,
   max_iter=None, normalize=False, random_state=None, solver='auto',
   tol=0.001)
```

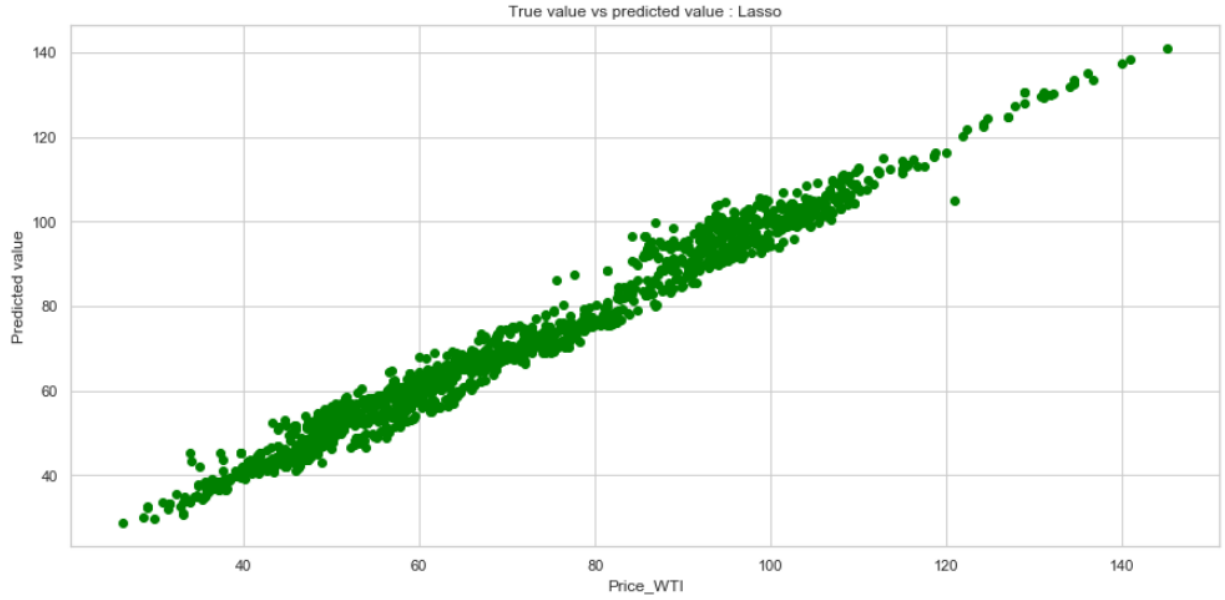
Ridge after Tuning:

```
from sklearn.linear_model import Ridge
model = Ridge(alpha=0.9883738380592262, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
model.fit(X_train, y_train)
print('R-Square Score of training data:', model.score(X_train, y_train))
print('R-Square Score of test data:', model.score(X_test, y_test))
```

```
R-Square Score of training data: 0.9776243261041963
R-Square Score of test data: 0.9793206766413454
```

12. Lasso Regression:

```
plt.scatter(y_test, y_predLS, c = 'green')
plt.xlabel("Price_WTI")
plt.ylabel("Predicted value")
plt.title("True value vs predicted value : Lasso")
plt.show()
```



R-Square Score of Lasso

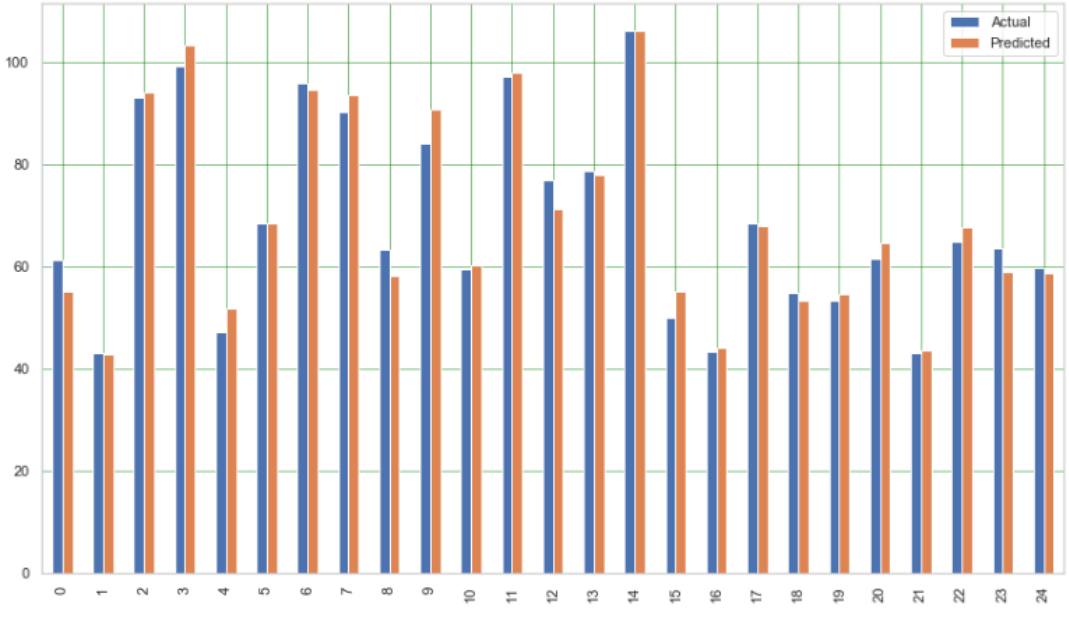
```
print('Accuracy on training data:', LS.score(X_train, y_train))
print('Accuracy on test data:', LS.score(X_test, y_test))
```

```
Accuracy on training data: 0.9776238533916661
Accuracy on test data: 0.9793132197177501
```

Actual vs Predicted Bar Chart of Lasso:

Bar Chart of Actual vs Predicted

```
ds = dsRI.head(25)
ds.plot(kind='bar',figsize=(14,8))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



13. Tuning of Lasso:

Tuning of Lasso:

```
from sklearn.model_selection import RandomizedSearchCV as rs
from scipy.stats import uniform as sp_rand

# SVM with random search
param_grid = {'alpha': sp_rand()}
model = Lasso()
clf_random = rs(estimator=model, param_distributions = param_grid, n_iter = 100, random_state = 0)
clf_random.fit(X_train, y_train)
predictions = clf_random.predict(X_test)
print('Test Score of Lasso (%):', clf_random.score(X_test, y_test) * 100)
print('Train Score of Lasso (%):', clf_random.score(X_train, y_train) * 100)
# examine the best model
# Single best score achieved across all params (min_samples_split)
print('Best Score (%):', clf_random.best_score_ * 100)
# Dictionary  containing the parameters (min_samples_split) used to generate
print('Best Parameters:', clf_random.best_params_)
# Actual model object fit with those best parameters
# Shows default parameters that we didn't specify
print('Best Estimator:', clf_random.best_estimator_)
```

```
Test Score of Lasso (%): 97.92426014796688
Train Score of Lasso (%): 97.75958547198611
Best Score (%): 97.74698592987339
Best Parameters: {'alpha': 0.13179786240439217}
Best Estimator: Lasso(alpha=0.13179786240439217, copy_X=True, fit_intercept=True,
   max_iter=1000, normalize=False, positive=False, precompute=False,
   random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

Lasso after Tuning:

```
from sklearn.linear_model import Ridge
model = Lasso(alpha=0.09710127579306127, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
model.fit(X_train, y_train)
print('R-Square Score on training data:', model.score(X_train, y_train))
print('R-Square Score on test data:', model.score(X_test, y_test))
```

```
R-Square Score on training data: 0.9775983739401907
R-Square Score on test data: 0.9792425672794509
```

14. MAE, MSE, MAPE and RMSE of All Model:

- Error Calculation of Linear and Random:

Error Calculation of Multiple Linear Regression

```
print('Multiple Linear Regression:\n')
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_predLR))
print('Mean Absolute Percentage Error:', (metrics.mean_absolute_error(y_test, y_predLR))*100)
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_predLR))
print('Root Mean Squared Error:', math.sqrt(metrics.mean_squared_error(y_test, y_predLR)))
```

```
Multiple Linear Regression:

Mean Absolute Error: 2.5897406509128413
Mean Absolute Percentage Error: 258.97406509128416
Mean Squared Error: 10.569454811725004
Root Mean Squared Error: 3.2510697949636524
```

Error Calculation of Random Forest Regression

```
print('Random Forest Regression:\n')
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_predRF))
print('Mean Absolute Percentage Error:', (metrics.mean_absolute_error(y_test, y_predRF))*100)
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_predRF))
print('Root Mean Squared Error:', math.sqrt(metrics.mean_squared_error(y_test, y_predRF)))
```

```
Random Forest Regression:

Mean Absolute Error: 0.5926263909348732
Mean Absolute Percentage Error: 59.26263909348732
Mean Squared Error: 1.069687535054712
Root Mean Squared Error: 1.0342569966186894
```

- Error Calculation of Lasso and Ridge:

Error Calculation of Ridge Regression

```
print('Ridge Regression:\n')
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_predRI))
print('Mean Absolute Percentage Error:', (metrics.mean_absolute_error(y_test, y_predRI))*100)
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_predRI))
print('Root Mean Squared Error:', math.sqrt(metrics.mean_squared_error(y_test, y_predRI)))
```

```
Ridge Regression:

Mean Absolute Error: 2.589740679677737
Mean Absolute Percentage Error: 258.9740679677737
Mean Squared Error: 10.569455006211985
Root Mean Squared Error: 3.25106982487488
```

Error Calculation of Lasso Regression

```
print('Lasso Regression:\n')
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_predLS))
print('Mean Absolute Percentage Error:', (metrics.mean_absolute_error(y_test, y_predLS))*100)
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_predLS))
print('Root Mean Squared Error:', math.sqrt(metrics.mean_squared_error(y_test, y_predLS)))
```

```
Lasso Regression:

Mean Absolute Error: 2.5903075423217894
Mean Absolute Percentage Error: 259.03075423217894
Mean Squared Error: 10.573285361257545
Root Mean Squared Error: 3.2516588629894043
```

Datasets Links:

- Crude oil historical data
  (https://www.investing.com/commodities/crude-oil)
- Attock Refinery Limited.
  (https://www.investing.com/equities/attock-refiner)
- Byco Petroleum Pakistan Limited.
  (https://www.investing.com/equities/byco-petroleum-chart)
- Pakistan Refinery Limited.
  (https://www.investing.com/equities/pak-refinery-l)
- Pakistan State Oil.
  (https://www.investing.com/equities/pak-state-oil-chart)
- Pakistan Oilfields Limited.
  (https://www.investing.com/equities/pak-oilfields-historical-data)
- Oil and Gas Development Company Limited.
  (https://www.investing.com/equities/oil---gas-dev-chart)
- World Texas Intermediate Historical Data.
  (https://www.investing.com/commodities/crude-oil)
- Brent Crude Oil.
  (https://www.investing.com/commodities/brent-oil)
- KSE100 Historical Data.
  (https://www.investing.com/indices/karachi-100)
- USD to PKR Data.
  (https://www.investing.com/)
- Gold Price in PKR.
  (https://data.worldbank.org/)

Charts of Score:

- **When Change%_WTI treated as Target:**

Models	Training Score	Testing Score
Multiple Linear	0.70	0.74
Random Forest	0.97	0.80
Simple Vector	0.54	0.57
Ridge	0.70	0.74
Lasso	0.70	0.74
Ridge with Tuning	0.70	0.74
Lasso with Tuning	0.69	0.75

- **When Price_WTI treated as Target:**

Models	Training Score	Testing Score
Multiple Linear	0.97	0.97
Random Forest	0.99	0.99
Ridge	0.977	0.979
Lasso	0.977	0.979
Ridge with Tuning	0.979	0.977
Lasso with Tuning	0.979	0.977