In [ ]:

```python
# Table
import pandas as pd
import numpy as np
import datetime
import random

# Graphic
import seaborn as sns
import missingno as msno
import matplotlib.pyplot as plt

# Regression Algorithm:
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression, Ridge, SGDRegressor, Logis
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR, LinearSVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor

# Model Helper
from sklearn.model_selection import train_test_split

# Preprocess
from sklearn.preprocessing import  StandardScaler

# Regression
import math
from sklearn.metrics import mean_squared_error, r2_score,mean_absolute_error

# Ignore warnings :
import warnings
warnings.filterwarnings('ignore')

import scipy.stats as sci
```

In [ ]:

```python
#Step-1 Load data
dataset = pd.read_csv("diamonds.csv")# load dataset
print(dataset.head(10)) #visualise
print('Dataset instance and features:', dataset.shape)
```

```
   Unnamed: 0  carat        cut color clarity  depth  table     x     y     z
\
0           1   0.23      Ideal     E     SI2   61.5   55.0  3.95  3.98  2.43
1           2   0.21    Premium     E     SI1   59.8   61.0  3.89  3.84  2.31
2           3   0.23       Good     E     VS1   56.9   65.0  4.05  4.07  2.31
3           4   0.29    Premium     I     VS2   62.4   58.0  4.20  4.23  2.63
4           5   0.31       Good     J     SI2   63.3   58.0  4.34  4.35  2.75
5           6   0.24  Very Good     J    VVS2   62.8   57.0  3.94  3.96  2.48
6           7   0.24  Very Good     I    VVS1   62.3   57.0  3.95  3.98  2.47
7           8   0.26  Very Good     H     SI1   61.9   55.0  4.07  4.11  2.53
8           9   0.22       Fair     E     VS2   65.1   61.0  3.87  3.78  2.49
9          10   0.23  Very Good     H     VS1   59.4   61.0  4.00  4.05  2.39

   price
0    326
1    326
2    327
3    334
4    335
5    336
6    336
7    337
```

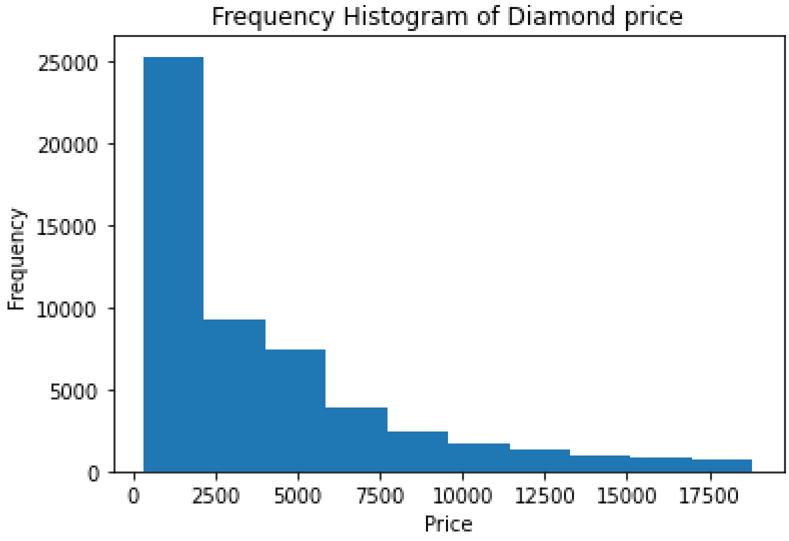
```
8    337
9    338
Dataset instance and features: (53940, 11)
```

In [ ]:

```
#Step-2 distribution
plt.hist(dataset['price'])
plt.gca().set(title='Frequency Histogram of Diamond price', ylabel='Frequency
```

Out[ ]:

```
[Text(0.5, 1.0, 'Frequency Histogram of Diamond price'),
 Text(0, 0.5, 'Frequency'),
 Text(0.5, 0, 'Price')]
```



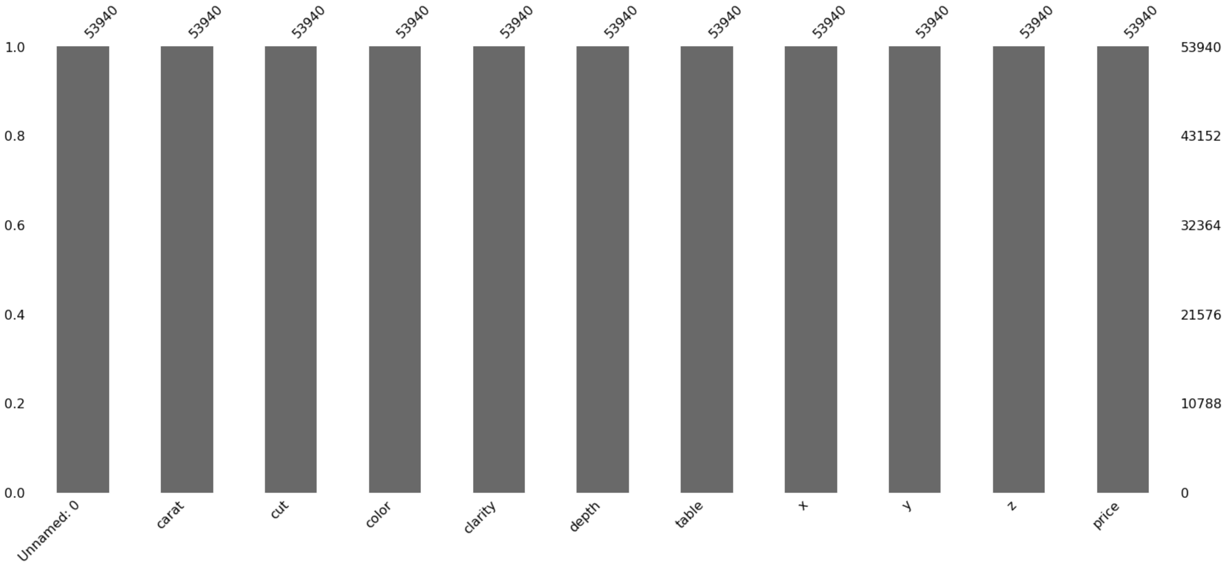
In [ ]:

```
#Step-2 Analyse (missing values)
print(dataset.isnull().sum()) # Missing value Non-graphic
msno.bar(dataset,labels=True) # Missing value Graphic
```

```
Unnamed: 0    0
carat         0
cut           0
color         0
clarity       0
depth         0
table         0
x             0
y             0
z             0
price         0
dtype: int64
```

Out[ ]:

```
<AxesSubplot:>
```



In [ ]:

```
#Step-2 Analyse (data features)
print(dataset.describe()) # Non-graphic
# Lowest value for X,Y,Z (length, width, depth) is 0 this is impossible - ind
print(dataset.loc[(dataset['x']==0) | (dataset['y']==0) | (dataset['z']==0)])
print("Number of instances with 0 value for X,Y,Z: ")
print(len(dataset.loc[(dataset['x']==0) | (dataset['y']==0) | (dataset['z']==
```

```
         Unnamed: 0         carat         depth         table             x  \
count  53940.000000  53940.000000  53940.000000  53940.000000  53940.000000
mean   26970.500000      0.797940     61.749405     57.457184      5.731157
std    15571.281097      0.474011      1.432621      2.234491      1.121761
min        1.000000      0.200000     43.000000     43.000000      0.000000
25%    13485.750000      0.400000     61.000000     56.000000      4.710000
50%    26970.500000      0.700000     61.800000     57.000000      5.700000
75%    40455.250000      1.040000     62.500000     59.000000      6.540000
max    53940.000000      5.010000     79.000000     95.000000     10.740000

                  y             z         price
count  53940.000000  53940.000000  53940.000000
mean       5.734526      3.538734   3932.799722
std        1.142135      0.705699   3989.439738
min        0.000000      0.000000    326.000000
25%        4.720000      2.910000    950.000000
50%        5.710000      3.530000   2401.000000
75%        6.540000      4.040000   5324.250000
max       58.900000     31.800000  18823.000000
       Unnamed: 0  carat        cut color clarity  depth  table     x     y  \
2207         2208   1.00    Premium     G     SI2   59.1   59.0  6.55  6.48
2314         2315   1.01    Premium     H      I1   58.1   59.0  6.66  6.60
4791         4792   1.10    Premium     G     SI2   63.0   59.0  6.50  6.47
5471         5472   1.01    Premium     F     SI2   59.2   58.0  6.50  6.47
10167       10168   1.50       Good     G      I1   64.0   61.0  7.15  7.04
11182       11183   1.07      Ideal     F     SI2   61.6   56.0  0.00  6.62
11963       11964   1.00  Very Good     H     VS2   63.3   53.0  0.00  0.00
13601       13602   1.15      Ideal     G     VS2   59.2   56.0  6.88  6.83
15951       15952   1.14       Fair     G     VS1   57.5   67.0  0.00  0.00
24394       24395   2.18    Premium     H     SI2   59.4   61.0  8.49  8.45
24520       24521   1.56      Ideal     G     VS2   62.2   54.0  0.00  0.00
26123       26124   2.25    Premium     I     SI1   61.3   58.0  8.52  8.42
26243       26244   1.20    Premium     D    VVS1   62.1   59.0  0.00  0.00
27112       27113   2.20    Premium     H     SI1   61.2   59.0  8.42  8.37
27429       27430   2.25    Premium     H     SI2   62.8   59.0  0.00  0.00
27503       27504   2.02    Premium     H     VS2   62.7   53.0  8.02  7.95
27739       27740   2.80       Good     G     SI2   63.8   58.0  8.90  8.85
49556       49557   0.71       Good     F     SI2   64.1   60.0  0.00  0.00
```

```
49557       49558   0.71      Good     F     SI2    64.1    60.0   0.00   0.00
51506       51507   1.12   Premium     G      I1    60.4    59.0   6.71   6.67

         z  price
2207   0.0   3142
2314   0.0   3167
4791   0.0   3696
5471   0.0   3837
10167  0.0   4731
11182  0.0   4954
11963  0.0   5139
13601  0.0   5564
15951  0.0   6381
24394  0.0  12631
24520  0.0  12800
26123  0.0  15397
26243  0.0  15686
27112  0.0  17265
27429  0.0  18034
27503  0.0  18207
27739  0.0  18788
49556  0.0   2130
49557  0.0   2130
51506  0.0   2383
Number of instances with 0 value for X,Y,Z:
20
```

In [ ]:

```
#Step-3 Preprocess (remove incorrect instance)
dataset = dataset[(dataset[['x', 'y', 'z']] != 0).all(axis=1)] #Remove instan
#Check for successful removal
print("Number of instances with 0 value for X,Y,Z: ")
print(len(dataset.loc[(dataset['x']==0) | (dataset['y']==0) | (dataset['z']==
```

```
Number of instances with 0 value for X,Y,Z:
0
```

In [ ]:

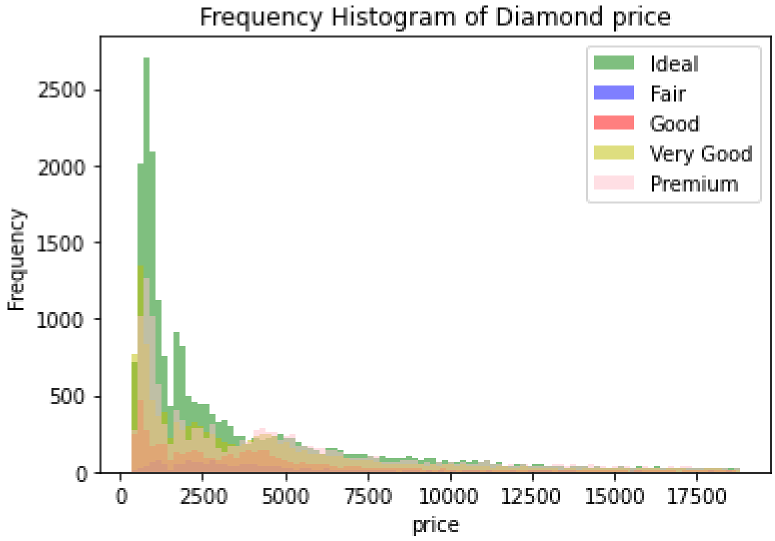
```
#Step-3 Preprocess (remove irrelevant features)
dataset.drop(['Unnamed: 0'] , axis=1 , inplace=True)# remove the first column
print(dataset.head(5))# check
```

```
   carat      cut color clarity  depth  table     x     y     z  price
0   0.23    Ideal     E     SI2   61.5   55.0  3.95  3.98  2.43    326
1   0.21  Premium     E     SI1   59.8   61.0  3.89  3.84  2.31    326
2   0.23     Good     E     VS1   56.9   65.0  4.05  4.07  2.31    327
3   0.29  Premium     I     VS2   62.4   58.0  4.20  4.23  2.63    334
4   0.31     Good     J     SI2   63.3   58.0  4.34  4.35  2.75    335
```

In [ ]:

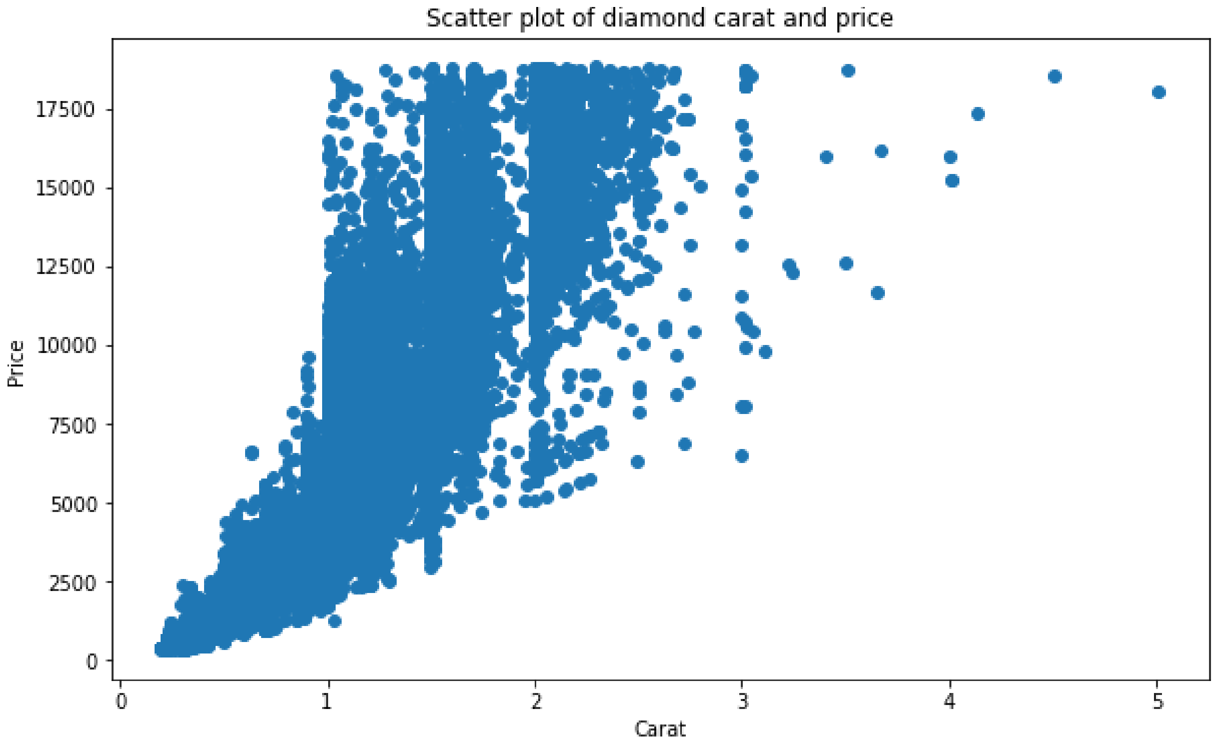
```
#Step-4 Exploratory Data Analysis: Histogram
#https://www.machinelearningplus.com/plots/matplotlib-histogram-python-exampl

x1 = dataset.loc[dataset.cut=='Ideal', 'price']
x2 = dataset.loc[dataset.cut=='Fair', 'price']
x3 = dataset.loc[dataset.cut=='Good', 'price']
x4 = dataset.loc[dataset.cut=='Very Good', 'price']
x5 = dataset.loc[dataset.cut=='Premium', 'price']

kwargs = dict(alpha=0.5, bins=100)

plt.hist(x1, **kwargs, color='g', label='Ideal')
plt.hist(x2, **kwargs, color='b', label='Fair')
plt.hist(x3, **kwargs, color='r', label='Good')
plt.hist(x4, **kwargs, color='y', label='Very Good')
plt.hist(x5, **kwargs, color='pink', label='Premium')
```

```
plt.gca().set(title='Frequency Histogram of Diamond price', ylabel='Frequency
plt.legend();
```



In [ ]:

```
#Step-4 Exploratory Data Analysis: Scatter-plot
fig, ax = plt.subplots(figsize=(10, 6))
ax.scatter(x = dataset['carat'], y = dataset['price'])
plt.gca().set(title='Scatter plot of diamond carat and price', ylabel="Price"
plt.show()
```



In [ ]:

```
#Step-4 Exploratory Data Analysis - making categorical variables numerical (p
mappingCut = {'Ideal': 60, 'Fair': 70,'Good': 80,'Very Good': 90, 'Premium':
mappingCla = {'I1': 30, 'SI1': 40, 'SI2': 50, 'VS1': 60, 'VS2': 70, 'VVS1': 8
mappingCol = {'J': 40, 'I': 50 , 'H': 60, 'G': 70,'F': 80 ,'E': 90, 'D': 100}
dataset = dataset.replace({'cut': mappingCut, 'clarity': mappingCla, 'color':m
print(dataset.head(5))# check
```

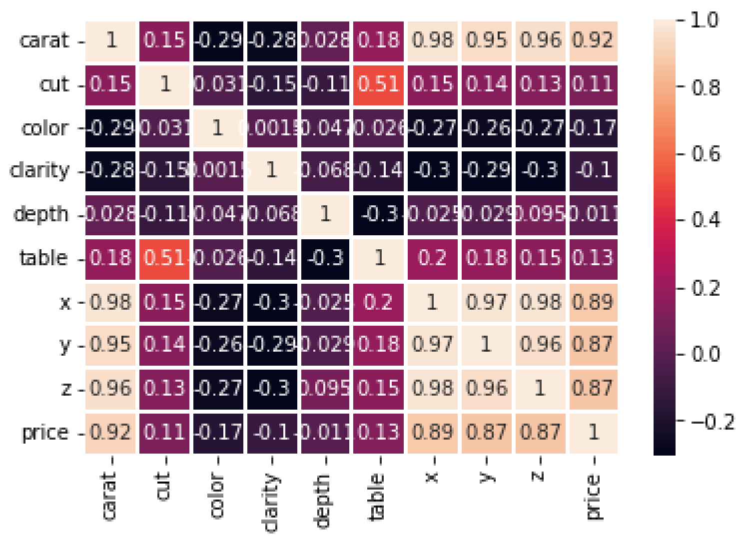
```
   carat  cut  color  clarity  depth  table     x     y     z  price
0   0.23   60     90       50   61.5   55.0  3.95  3.98  2.43    326
1   0.21  100     90       40   59.8   61.0  3.89  3.84  2.31    326
2   0.23   80     90       60   56.9   65.0  4.05  4.07  2.31    327
```

```
3    0.29   100      50        70    62.4    58.0   4.20   4.23   2.63     334
4    0.31    80      40        50    63.3    58.0   4.34   4.35   2.75     335
```

In [ ]:

```python
#Step-4 Exploratory Data Analysis - Correlation
corr = dataset.corr()
sns.heatmap(data = corr, annot = True, cbar = True, linewidths = 0.3)

#x, y, z is highly correlated with each other and price and carat
#price and carat are also highly correlated
```

Out[ ]: <AxesSubplot:>



In [ ]:

```python
# Dimensionality reduction
dataset = dataset.drop("x", axis=1)
dataset = dataset.drop("y", axis=1)
dataset = dataset.drop("z", axis=1)

print(dataset.head(5))# check
```

```
   carat  cut  color  clarity  depth  table  price
0   0.23   60     90       50   61.5   55.0    326
1   0.21  100     90       40   59.8   61.0    326
2   0.23   80     90       60   56.9   65.0    327
3   0.29  100     50       70   62.4   58.0    334
4   0.31   80     40       50   63.3   58.0    335
```

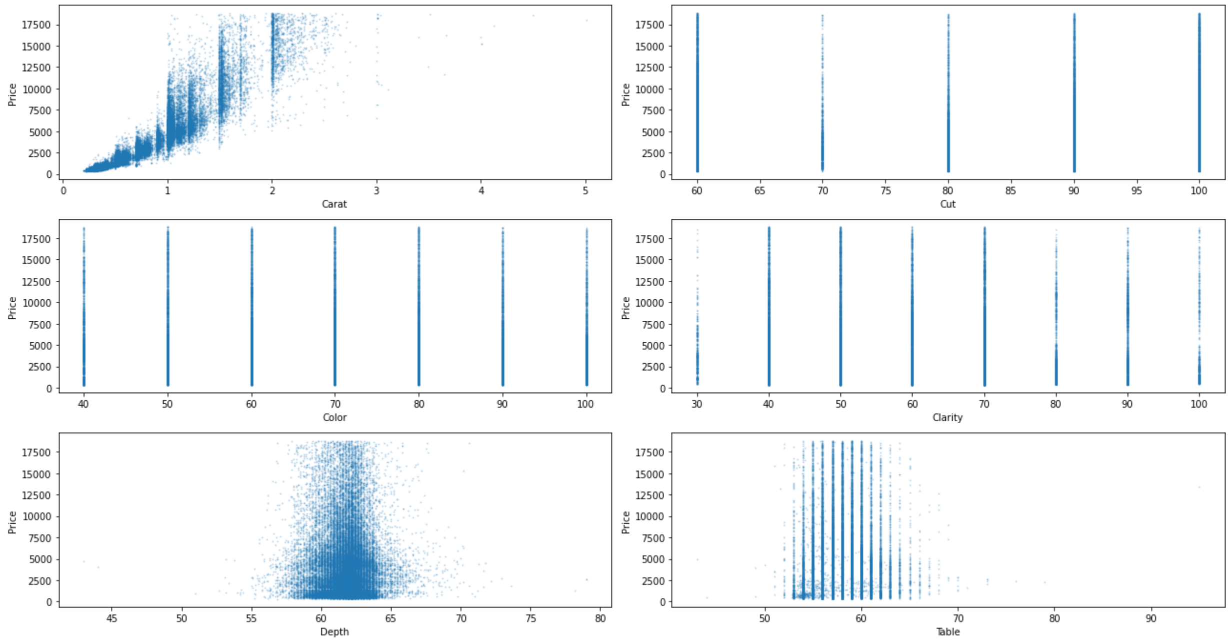
In [ ]:

```python
#Split data into train and test set
Trainset,Testset = train_test_split(dataset, test_size = 0.3, random_state = 

#Set target (y)
train_data_temporary = Trainset.copy()
X_train = train_data_temporary.drop(["price"],axis=1)
y_train = Trainset["price"]

test_data_temporary=Testset.copy()
X_test = test_data_temporary.drop(["price"],axis=1)
y_test = test_data_temporary["price"]
```

In [ ]:

```python
#Observe data

fig, axs = plt.subplots(3, 2, figsize=(18, 10))
axs = axs.ravel()
axs[0].scatter(Trainset.carat, Trainset.price, alpha = 0.2, s = 1)
axs[0].set_xlabel('Carat')
```

```
axs[1].scatter(Trainset.cut, Trainset.price, alpha = 0.2, s = 1)
axs[1].set_xlabel('Cut')
axs[2].scatter(Trainset.color, Trainset.price, alpha = 0.2, s = 1)
axs[2].set_xlabel('Color')
axs[3].scatter(Trainset.clarity, Trainset.price, alpha = 0.2, s = 1)
axs[3].set_xlabel('Clarity')
axs[4].scatter(Trainset.depth, Trainset.price, alpha = 0.2, s = 1)
axs[4].set_xlabel('Depth')
axs[5].scatter(Trainset.table, Trainset.price, alpha = 0.2, s = 1)
axs[5].set_xlabel('Table')
for i in range(6):
    axs[i].set_ylabel('Price')
    axs[i].set_xlim(auto = True)
    axs[i].set_ylim(auto = True)

plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```



In [ ]:

```
#Standardise
Xs_train_set_mean = X_train.mean()
Xs_train_set_std  = X_train.std()
Xs_train_set      = (X_train - Xs_train_set_mean) / Xs_train_set_std
Xs_test_set       = (X_test  - Xs_train_set_mean) / Xs_train_set_std
```

In [ ]:

```
#Method to print regression result
def PrintResult(regressor, label):
    start_time = datetime.datetime.now() #start timer
    regressor.fit(Xs_train_set, y_train) #get param regressor to fit data
    y_pred = regressor.predict(Xs_test_set) #get predictions after data has b
    end_time = datetime.datetime.now() #get time at end of regression fitting
    duration = (end_time - start_time).total_seconds() #get total time it tak

    MSE = mean_squared_error(y_test, y_pred) # mean squared error
    RMSE = math.pow(mean_squared_error(y_test, y_pred), 0.5) # root mean squa
    RSE = r2_score(y_test, y_pred) # relative squared error
    MAE = mean_absolute_error(y_test, y_pred) #mean absolute error

    print("-------------------------")
    print("Regressor: " + label)
    print(label + ' MSE : %0.2f ' % MSE)
    print(label + ' RMSE : %0.2f ' % RMSE)
    print(label + ' RSE : %0.2f ' % RSE)
```

```python
    print(label + ' MAE : %0.2f ' % MAE)
    print("Execution time: {t:.3f} seconds".format(t = duration))
    print("--------------------------")
```

In [ ]:

```python
# (1) linear regression
PrintResult(LinearRegression(), "Linear Regression")
```

```
--------------------------
Regressor: Linear Regression
Linear Regression MSE : 1722624.90
Linear Regression RMSE : 1312.49
Linear Regression RSE : 0.89
Linear Regression MAE : 893.64
Execution time: 0.021 seconds
--------------------------
```

In [ ]:

```python
# (2) k-neighbors regression
PrintResult(KNeighborsRegressor(), "K-Neighbors Regression")
```

```
--------------------------
Regressor: K-Neighbors Regression
K-Neighbors Regression MSE : 719912.91
K-Neighbors Regression RMSE : 848.48
K-Neighbors Regression RSE : 0.96
K-Neighbors Regression MAE : 476.29
Execution time: 1.315 seconds
--------------------------
```

In [ ]:

```python
# (3) Ridge regression
PrintResult(Ridge(), "Ridge Regression")
```

```
--------------------------
Regressor: Ridge Regression
Ridge Regression MSE : 1722639.68
Ridge Regression RMSE : 1312.49
Ridge Regression RSE : 0.89
Ridge Regression MAE : 893.63
Execution time: 0.011 seconds
--------------------------
```

In [ ]:

```python
# (4) decision tree regression
PrintResult(DecisionTreeRegressor(), "Decision Tree Regression")
```

```
--------------------------
Regressor: Decision Tree Regression
Decision Tree Regression MSE : 524527.25
Decision Tree Regression RMSE : 724.24
Decision Tree Regression RSE : 0.97
Decision Tree Regression MAE : 360.41
Execution time: 0.189 seconds
--------------------------
```

In [ ]:

```python
# (5) random forest regression
PrintResult(RandomForestRegressor(), "Random Forest Regression")
```

```
--------------------------
Regressor: Random Forest Regression
Random Forest Regression MSE : 304683.91
Random Forest Regression RMSE : 551.98
Random Forest Regression RSE : 0.98
Random Forest Regression MAE : 284.41
```

```
Execution time: 9.401 seconds
--------------------------
```

In [ ]:

```python
# (6) gradient Boosting regression
PrintResult(GradientBoostingRegressor(), "Gradient Boosting Regression")
```

```
--------------------------
Regressor: Gradient Boosting Regression
Gradient Boosting Regression MSE : 430038.49
Gradient Boosting Regression RMSE : 655.77
Gradient Boosting Regression RSE : 0.97
Gradient Boosting Regression MAE : 359.63
Execution time: 3.651 seconds
--------------------------
```

In [ ]:

```python
# (7) SGD regression
PrintResult(SGDRegressor(), "SGD Regression")
```

```
--------------------------
Regressor: SGD Regression
SGD Regression MSE : 1721526.91
SGD Regression RMSE : 1312.07
SGD Regression RSE : 0.89
SGD Regression MAE : 897.04
Execution time: 0.171 seconds
--------------------------
```

In [ ]:

```python
# (8) support vector regression (SVR)
PrintResult(SVR(), "SVR Regression")
```

```
--------------------------
Regressor: SVR Regression
SVR Regression MSE : 10151657.69
SVR Regression RMSE : 3186.17
SVR Regression RSE : 0.38
SVR Regression MAE : 1725.24
Execution time: 193.990 seconds
--------------------------
```

In [ ]:

```python
# (9) linear SVR
PrintResult(LinearSVR(), "Linear SVR Regression")
```

```
--------------------------
Regressor: Linear SVR Regression
Linear SVR Regression MSE : 2663564.22
Linear SVR Regression RMSE : 1632.04
Linear SVR Regression RSE : 0.84
Linear SVR Regression MAE : 872.50
Execution time: 0.081 seconds
--------------------------
```

In [ ]:

```python
# (10) multi-layer perceptron regression
PrintResult(MLPRegressor(), "MLP Regression")
```

```
--------------------------
Regressor: MLP Regression
MLP Regression MSE : 1028487.52
MLP Regression RMSE : 1014.14
MLP Regression RSE : 0.94
MLP Regression MAE : 595.06
Execution time: 35.478 seconds
--------------------------
```

In [ ]:

```
# Optimisation
# (8-op) support vector regression (SVR)
PrintResult(SVR(C=500), "SVR Regression Optimised")

# (9-op) linear SVR
PrintResult(LinearSVR(C=5, loss='squared_epsilon_insensitive', dual=True), "L

# (10-op) multi-layer perceptron regression
PrintResult(MLPRegressor(activation = 'relu', solver='lbfgs', learning_rate=
```

```
--------------------------
Regressor: SVR Regression Optimised
SVR Regression Optimised MSE : 827959.31
SVR Regression Optimised RMSE : 909.92
SVR Regression Optimised RSE : 0.95
SVR Regression Optimised MAE : 474.64
Execution time: 185.064 seconds
--------------------------
--------------------------
Regressor: Linear SVR Regression Optimised
Linear SVR Regression Optimised MSE : 1721734.01
Linear SVR Regression Optimised RMSE : 1312.15
Linear SVR Regression Optimised RSE : 0.89
Linear SVR Regression Optimised MAE : 894.20
Execution time: 6.197 seconds
--------------------------
--------------------------
Regressor: MLP Regression Optimised
MLP Regression Optimised MSE : 766972.63
MLP Regression Optimised RMSE : 875.77
MLP Regression Optimised RSE : 0.95
MLP Regression Optimised MAE : 497.58
Execution time: 18.665 seconds
--------------------------
```