Machine Learning Project -2 Report:

Name: Amandeep Singh

SRN: PES1UG19EC034

Section: A

Problem Statement:

Given the features and the target, build a **Machine Learning Regression model** that can **predict** the price of the diamond from a given set of features.

Note 1: Please divide the dataset into 70% for training, 20% for validation and keep the rest 10% for testing.

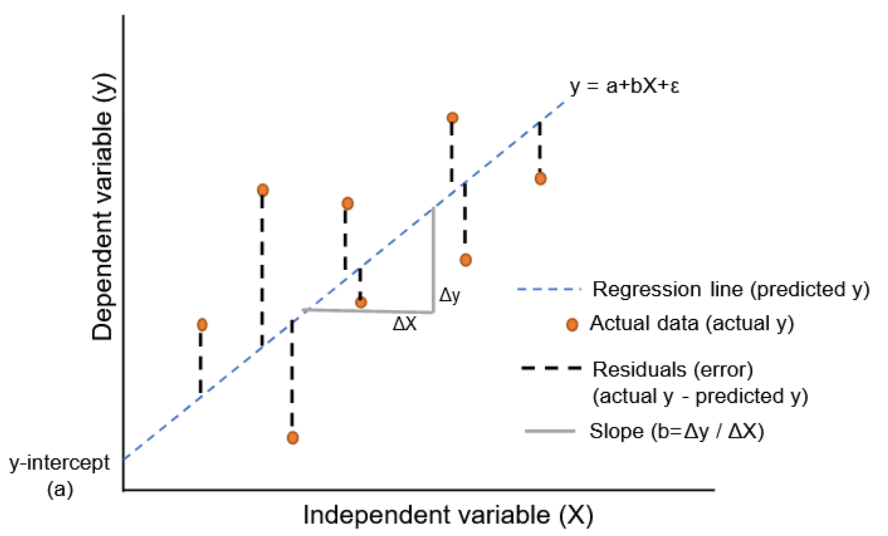
Note 2: Upon observation some of the data is of string type(Eg: Color). Since python or scikit learn works with numbers, it will be required to convert the string type of data into numbers. ***Clue: Check out Label Encoding using Scikit Learn***

You can use any regression algorithm. Please make sure you answer the following are in your report with the code:

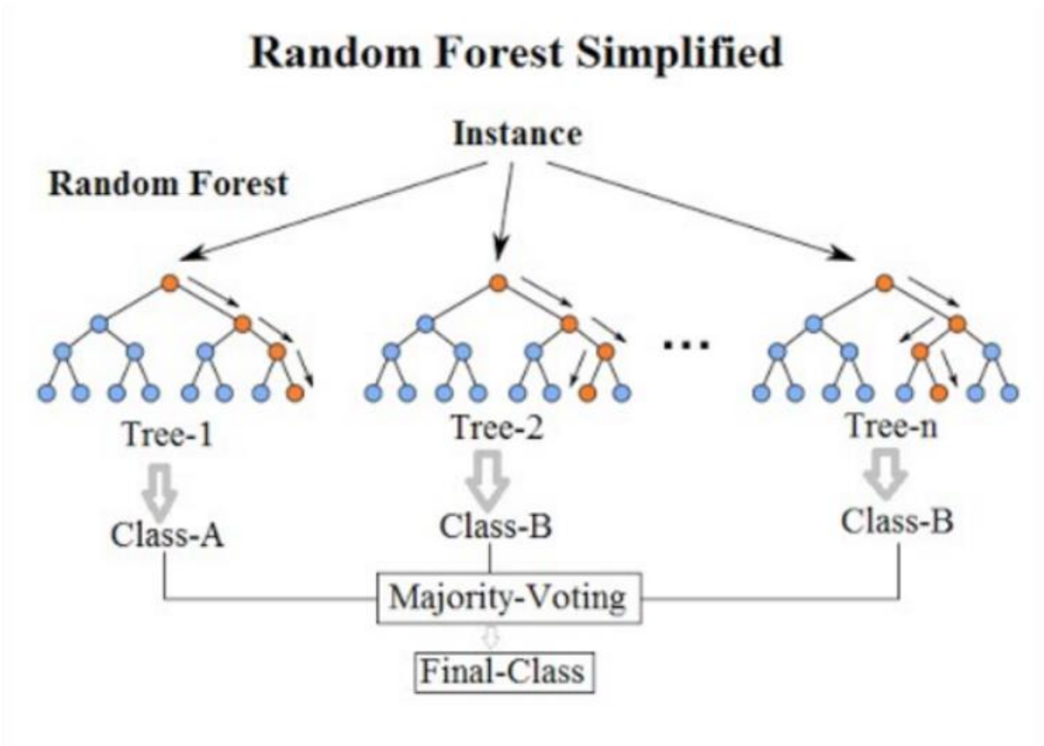
- Briefly explain the algorithm.
- What is the Mean Sqaured Error and Mean Absolute Error obtained?
- Observe and note changes in accuracy as you **vary parameters**. Ex. Number of Nodes in Random Forest regressor.

Optional: An interesting extra task that can be done after you are able to predict is **compare the performance** on different algorithms. Also looking into **Exploratory data Analysis** will be useful.

1) <u>**The Algorithms used:**</u>
    a. Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable. For example, a modeler might want to relate the weights of individuals to their heights using a linear regression model.



b. A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as **bagging**. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.
Random Forest has multiple decision trees as base learning models. We randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model. This part is called Bootstrap.



2) Mean Squared Error and Mean Absolute Error obtained:

Algorithm	Data Set	MAE	MSE	Regressor Score
Linear Regression	**Test**	854.8412490260101	1762989.8616778932	0.8918478832530993
Linear Regression	**Validation**	867.1339753208684	1857076.3407274585	0.885470233579283
Random Forest Regressor	**Test**	289.00106815203145	329616.43856319407	0.9797793984412048
Random Forest Regressor	**Validation**	290.78118485379576	325350.52193671477	0.9799349555723306

Program Code and Outputs:

In [10]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
d=pd.read_csv("E:\ML Workshop\diamonds.csv")
```

In [11]:
```python
d
```

Out[11]:

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...	...
53935	53936	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53936	53937	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53937	53938	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56
53938	53939	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53939	53940	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64

53940 rows × 11 columns

In [12]:
```python
d=d.drop(["Unnamed: 0"],axis=1)
```

In [13]:
```python
d
```

Out[13]:

carat cut color clarity depth table price x y z

In [12]:
```python
d=d.drop(["Unnamed: 0"],axis=1)
```

In [13]:
```python
d
```

Out[13]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...
53935	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53936	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53937	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56
53938	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53939	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64

53940 rows × 10 columns

In [14]:
```python
from sklearn import preprocessing
le=preprocessing.LabelEncoder()
d["cut"]=le.fit_transform(d["cut"])
d["color"]=le.fit_transform(d["color"])
d["clarity"]=le.fit_transform(d["clarity"])
```

In [15]:
```python
y=d["price"]
x=d.drop(["price"],axis=1)
```

In [16]:
```python
from sklearn.model_selection import train_test_split
x_train,x_left,y_train,y_left=train_test_split(x,y,train_size=0.7,random_state=56)
x_vali,x_test,y_vali,y_test=train_test_split(x_left,y_left,test_size=0.33,random_state=56)
```

In [15]:
```
y=d["price"]
x=d.drop(["price"],axis=1)
```

In [16]:
```
from sklearn.model_selection import train_test_split
x_train,x_left,y_train,y_left=train_test_split(x,y,train_size=0.7,random_state=56)
x_vali,x_test,y_vali,y_test=train_test_split(x_left,y_left,test_size=0.33,random_state=56)
```

In [17]:
```
from sklearn.linear_model import LinearRegression

lg=LinearRegression()

lg.fit(x_train,y_train)

ytestpred=lg.predict(x_test)
```

In [18]:
```

test=pd.DataFrame({'Actual:':y_test,'Predicted:':ytestpred})
test
```

Out[18]:

	Actual:	Predicted:
37529	987	634.189680
23925	12052	12402.937162
48878	2036	4625.184121
7234	4194	4373.200090
39274	490	-399.158163
...	...	...
52811	2572	2546.691629
16617	6640	5735.247845
1262	2947	3981.553921
23796	11854	13352.806806
775	2859	3025.539307

5341 rows × 2 columns

In [19]:
```
from sklearn.metrics import mean_squared_error as mse
print("MSE on test set:",mse(y_test,ytestpred))
from sklearn.metrics import mean_absolute_error as mae
print("MAE on test set:",mae(y_test,ytestpred))
```

```
MSE on test set: 1762989.8616778932
MAE on test set: 854.8412490260101
```

In [20]:
```
print("Regressor score on test set is:",lg.score(x_test,y_test))
```

```
Regressor score on test set is: 0.8918478832530993
```

In [21]:
```
y_valid_pred=lg.predict(x_vali)
```

In [22]:
```
df_valid=pd.DataFrame({'Actual:':y_vali,'Predicted:':y_valid_pred})
df_valid
```

Out[22]:

	Actual:	Predicted:
47528	1869	2380.102928
53002	2596	3222.405674
11773	5082	6389.462665
32262	789	732.047810
2210	3142	5194.234826
...	...	...
33308	827	620.178922
34634	872	1119.828818
42966	1365	2280.630750
6795	4115	4423.260941
26385	15878	8438.397462

10841 rows × 2 columns

In [23]:
```
print("MSE on validation set:",mse(y_vali,y_valid_pred))
```

In [23]:
```python
print("MSE on validation set:",mse(y_vali,y_valid_pred))
print("MAE on validation set:",mae(y_vali,y_valid_pred))
```

```
MSE on validation set: 1857076.3407274585
MAE on validation set: 867.1339753208684
```

In [24]:
```python
print("Regressor score on validation set is:",lg.score(x_vali,y_vali))
```

```
Regressor score on validation set is: 0.885470233579283
```

In [25]:
```python
from sklearn.ensemble import RandomForestRegressor

rf=RandomForestRegressor(n_estimators=10)
rf.fit(x_train,y_train)
y_test_pred1=rf.predict(x_test)
y_valid_pred1=rf.predict(x_vali)
print("MSE on test set:",mse(y_test,y_test_pred1))
print("MAE on test set:",mae(y_test,y_test_pred1))
print("Regressor score on test set is:",rf.score(x_test,y_test))
print()
print("MSE on validation set:",mse(y_vali,y_valid_pred1))
print("MAE on validation set:",mae(y_vali,y_valid_pred1))
print("Regressor score on validation set is:",rf.score(x_vali,y_vali))
```

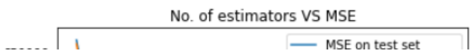
```
MSE on test set: 329616.43856319407
MAE on test set: 289.00106815203145
Regressor score on test set is: 0.9797793984412048

MSE on validation set: 325350.52193671477
MAE on validation set: 290.78118485379576
Regressor score on validation set is: 0.9799349555723306
```

In [26]:
```python
k=[k for k in range(1,15)]
mse_t,mae_t,score_t,mse_v,mae_v,score_v=[],[],[],[],[],[]
for i in k:
    rf1=RandomForestRegressor(n_estimators=i)
    rf1.fit(x_train,y_train)
    y_test_pred=rf1.predict(x_test)
    y_valid_pred=rf1.predict(x_vali)
```

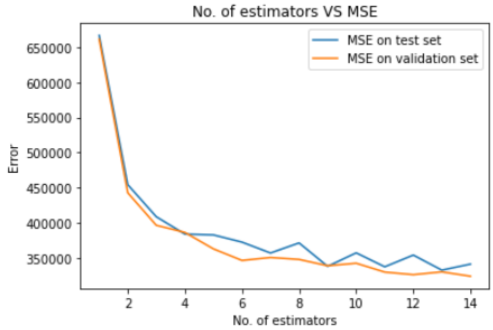

```python
mse_t,mae_t,score_t,mse_v,mae_v,score_v=[],[],[],[],[],[]
for i in k:
    rf1=RandomForestRegressor(n_estimators=i)
    rf1.fit(x_train,y_train)
    y_test_pred=rf1.predict(x_test)
    y_valid_pred=rf1.predict(x_vali)
```

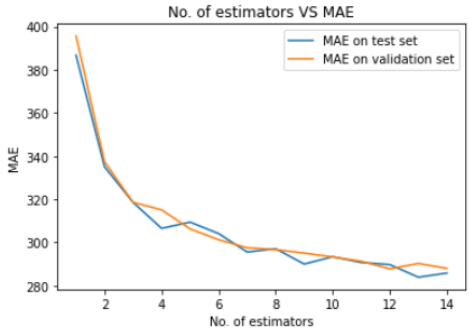

```python

    mse_t.append(mse(y_test,y_test_pred))
    mae_t.append(mae(y_test,y_test_pred))
    score_t.append(rf1.score(x_test,y_test))

    mse_v.append(mse(y_vali,y_valid_pred))
    mae_v.append(mae(y_vali,y_valid_pred))
    score_v.append(rf1.score(x_vali,y_vali))
```

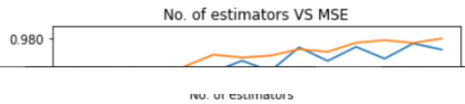
In [27]:
```python
plt.title("No. of estimators VS MSE")
plt.plot(k,mse_t,label="MSE on test set")
plt.plot(k,mse_v,label="MSE on validation set")
plt.legend()
plt.xlabel("No. of estimators")
plt.ylabel("Error")
plt.show()

plt.title("No. of estimators VS MAE")
plt.plot(k,mae_t,label="MAE on test set")
plt.plot(k,mae_v,label="MAE on validation set")
plt.legend()
plt.xlabel("No. of estimators")
plt.ylabel("MAE")
plt.show()

plt.title("No. of estimators VS MSE")
plt.plot(k,score_t,label="Score on test set")
plt.plot(k,score_v,label="Score on validation set")
plt.legend()
plt.xlabel("No. of estimators")
plt.ylabel("Score")
plt.show()
```

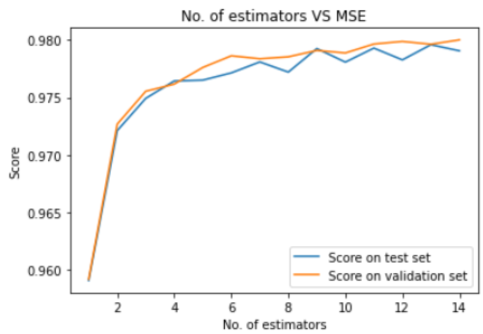


```
plt.legend()
plt.xlabel("No. of estimators")
plt.ylabel("Score")
plt.show()
```









Results:

We use Linear Regression and Random Forest Regressor based Supervised models to predict the prices of the diamond.

Algorithm	Data Set	MAE	MSE	Regressor Score
Linear Regression	**Test**	854.8412490260101	1762989.8616778932	0.8918478832530993
Linear Regression	**Validation**	867.1339753208684	1857076.3407274585	0.885470233579283

Random Forest Regressor	**Test**	289.00106815203145	329616.43856319407	0.9797793984412048
Random Forest Regressor	**Validation**	290.78118485379576	325350.52193671477	0.9799349555723306

The changes in predictions when the estimators are changed in Random Forest Regressor are shown by graphs.