Student name: Roja Kamble [11454258]

Assignment: Regression

Part 1: Data Wrangling

You have to write code to answer the questions below

- Import pandas library
- Read the data stored in your local machine https://www.kaggle.com/datasets/nancyalaswad90/diamonds-prices (https://www.kaggle.com/datasets/nancyalaswad90/diamonds-prices)

Read the excel and print the values

In [70]:
```
# importing libraries

import pandas as pd
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import sys, threading
sys.setrecursionlimit(10**7) # max depth of recursion
```

In [71]:
```
dataframe1 = pd.read_csv(r'/Users/roja/Downloads/Machine Learning HW/Diamonds Prices2022.csv')

dataframe1
```

Out[71]:

	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
...	...	...	...	...	...	...	...	...	...	...	...
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	53941	0.71	Premium	E	SI1	60.5	55.0	2756	5.79	5.74	3.49
53941	53942	0.71	Premium	F	SI1	59.8	62.0	2756	5.74	5.73	3.43
53942	53943	0.70	Very Good	E	VS2	60.5	59.0	2757	5.71	5.76	3.47

53943 rows × 11 columns

Drop the first column: Unnamed: 0 and show the new dataset

In [72]:
```
# inplace = false means it will copy in new file, otherwise it will copy in same file.

new_data = dataframe1.drop(dataframe1.columns[0], axis=1, inplace=False)

new_data
```

Out[72]:

	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
...	...	...	...	...	...	...	...	...	...	...
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	0.71	Premium	E	SI1	60.5	55.0	2756	5.79	5.74	3.49
53941	0.71	Premium	F	SI1	59.8	62.0	2756	5.74	5.73	3.43
53942	0.70	Very Good	E	VS2	60.5	59.0	2757	5.71	5.76	3.47

53943 rows × 10 columns

Show information about the dataset such as number of columns and data types, memory usage, any null values, etc.

In [73]:
```
# Display the file information

dataframe1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53943 entries, 0 to 53942
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  53943 non-null  int64  
 1   carat       53943 non-null  float64
 2   cut         53943 non-null  object 
 3   color       53943 non-null  object 
 4   clarity     53943 non-null  object 
 5   depth       53943 non-null  float64
 6   table       53943 non-null  float64
 7   price       53943 non-null  int64  
 8   x           53943 non-null  float64
 9   y           53943 non-null  float64
 10  z           53943 non-null  float64
dtypes: float64(6), int64(2), object(3)
memory usage: 4.5+ MB
```

Show unique values of column that are of object type
Reference: https://stackoverflow.com/questions/25039626/how-do-i-find-numeric-columns-in-pandas (https://stackoverflow.com/questions/25039626/how-do-i-find-numeric-columns-in-pandas)
Note: You may need to use the flag exclude='number' instead of "include"

In [74]:
```
# First add the d_types which you want to exclude

d_type = ['float64', 'int64']

# dataframe1.select_dtypes(exclude = d_type).columns.tolist()

object_type = new_data.select_dtypes(exclude = d_type)

object_type
```

Out[74]:

	cut	color	clarity
0	Ideal	E	SI2
1	Premium	E	SI1
2	Good	E	VS1
3	Premium	I	VS2
4	Good	J	SI2
...	...	...	...
53938	Premium	H	SI2
53939	Ideal	D	SI2
53940	Premium	E	SI1
53941	Premium	F	SI1
53942	Very Good	E	VS2

53943 rows × 3 columns

In [75]:
```
# printing unique values

types =["int64","float64"]

df1 = new_data.select_dtypes(exclude=types)

for j in df1:
    print("column "+j+":")
    print(df1[j].unique())
```

```
column cut:
['Ideal' 'Premium' 'Good' 'Very Good' 'Fair']
column color:
['E' 'I' 'J' 'H' 'F' 'G' 'D']
column clarity:
['SI2' 'SI1' 'VS1' 'VS2' 'VVS2' 'VVS1' 'I1' 'IF']
```

Ordinally encode the column(s) of which values are ordinal. For those of which values are still categorical but not ordered, one-hot-encode them
Reference (you may need incognito mode to browse the pages):
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html)
https://towardsdatascience.com/guide-to-encoding-categorical-features-using-scikit-learn-for-machine-learning-5048997a5c79 (https://towardsdatascience.com/guide-to-encoding-categorical-features-using-scikit-learn-for-machine-learning-5048997a5c79)
https://stackoverflow.com/questions/56502864/using-ordinalencoder-to-transform-categorical-values (https://stackoverflow.com/questions/56502864/using-ordinalencoder-to-transform-categorical-values)
https://stackoverflow.com/questions/37292872/how-can-i-one-hot-encode-in-python (https://stackoverflow.com/questions/37292872/how-can-i-one-hot-encode-in-python)
https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html (https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html)

In [76]:

```
df1 = new_data
enc = OrdinalEncoder()
r = enc.fit_transform(df1[['cut']])
df1['cut'] = r
df1
```

Out[76]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	2.0	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	3.0	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	1.0	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	3.0	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	1.0	J	SI2	63.3	58.0	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...
53938	0.86	3.0	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53939	0.75	2.0	D	SI2	62.2	55.0	2757	5.83	5.87	3.64
53940	0.71	3.0	E	SI1	60.5	55.0	2756	5.79	5.74	3.49
53941	0.71	3.0	F	SI1	59.8	62.0	2756	5.74	5.73	3.43
53942	0.70	4.0	E	VS2	60.5	59.0	2757	5.71	5.76	3.47

53943 rows × 10 columns

In [77]:
```
for j in ["color","clarity"]:
    one_hot = pd.get_dummies(df1[j])
    df1 = df1.drop(j,axis = 1)
    df1 = df1.join(one_hot)
df1
```

Out[77]:

	carat	cut	depth	table	price	x	y	z	D	E	...	I	J	I1	IF	SI1	SI2	VS1	VS2	VVS1	VVS2
0	0.23	2.0	61.5	55.0	326	3.95	3.98	2.43	0	1	...	0	0	0	0	0	1	0	0	0	0
1	0.21	3.0	59.8	61.0	326	3.89	3.84	2.31	0	1	...	0	0	0	0	1	0	0	0	0	0
2	0.23	1.0	56.9	65.0	327	4.05	4.07	2.31	0	1	...	0	0	0	0	0	0	1	0	0	0
3	0.29	3.0	62.4	58.0	334	4.20	4.23	2.63	0	0	...	1	0	0	0	0	0	0	1	0	0
4	0.31	1.0	63.3	58.0	335	4.34	4.35	2.75	0	0	...	0	1	0	0	0	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
53938	0.86	3.0	61.0	58.0	2757	6.15	6.12	3.74	0	0	...	0	0	0	0	0	1	0	0	0	0
53939	0.75	2.0	62.2	55.0	2757	5.83	5.87	3.64	1	0	...	0	0	0	0	0	1	0	0	0	0
53940	0.71	3.0	60.5	55.0	2756	5.79	5.74	3.49	0	1	...	0	0	0	0	1	0	0	0	0	0
53941	0.71	3.0	59.8	62.0	2756	5.74	5.73	3.43	0	0	...	0	0	0	0	1	0	0	0	0	0
53942	0.70	4.0	60.5	59.0	2757	5.71	5.76	3.47	0	1	...	0	0	0	0	0	0	0	1	0	0

53943 rows × 23 columns

Show the last 10 rows

In [78]:
```
# Getting last 3 rows from df
data_last_10 = df1.tail(10)

# Printing df_last_3
data_last_10
```

Out[78]:

	carat	cut	depth	table	price	x	y	z	D	E	...	I	J	I1	IF	SI1	SI2	VS1	VS2	VVS1	VVS2
53933	0.70	4.0	61.2	59.0	2757	5.69	5.72	3.49	0	1	...	0	0	0	0	0	0	0	1	0	0
53934	0.72	3.0	62.7	59.0	2757	5.69	5.73	3.58	1	0	...	0	0	0	0	1	0	0	0	0	0
53935	0.72	2.0	60.8	57.0	2757	5.75	5.76	3.50	1	0	...	0	0	0	0	1	0	0	0	0	0
53936	0.72	1.0	63.1	55.0	2757	5.69	5.75	3.61	1	0	...	0	0	0	0	1	0	0	0	0	0
53937	0.70	4.0	62.8	60.0	2757	5.66	5.68	3.56	1	0	...	0	0	0	0	1	0	0	0	0	0
53938	0.86	3.0	61.0	58.0	2757	6.15	6.12	3.74	0	0	...	0	0	0	0	0	1	0	0	0	0
53939	0.75	2.0	62.2	55.0	2757	5.83	5.87	3.64	1	0	...	0	0	0	0	0	1	0	0	0	0
53940	0.71	3.0	60.5	55.0	2756	5.79	5.74	3.49	0	1	...	0	0	0	0	1	0	0	0	0	0
53941	0.71	3.0	59.8	62.0	2756	5.74	5.73	3.43	0	0	...	0	0	0	0	1	0	0	0	0	0
53942	0.70	4.0	60.5	59.0	2757	5.71	5.76	3.47	0	1	...	0	0	0	0	0	0	0	1	0	0

10 rows × 23 columns

Reset the index such that it starts from 1 (instead of 0) and print the first five rows

In [79]:
```
dataframe1.reset_index()

dataframe1.index = dataframe1.index+1
dataframe1[:5]
```

Out[79]:

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
1	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
2	2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
3	3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
4	4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
5	5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

Lowercase all columns and name the dataset df

In [80]:
```
lower = df1.applymap(lambda l: l.lower() if type(l) == str else l)
lower
```

Out[80]:

	carat	cut	depth	table	price	x	y	z	D	E	...	I	J	I1	IF	SI1	SI2	VS1	VS2	VVS1	VVS2
0	0.23	2.0	61.5	55.0	326	3.95	3.98	2.43	0	1	...	0	0	0	0	0	1	0	0	0	0
1	0.21	3.0	59.8	61.0	326	3.89	3.84	2.31	0	1	...	0	0	0	0	1	0	0	0	0	0
2	0.23	1.0	56.9	65.0	327	4.05	4.07	2.31	0	1	...	0	0	0	0	0	0	1	0	0	0
3	0.29	3.0	62.4	58.0	334	4.20	4.23	2.63	0	0	...	1	0	0	0	0	0	0	1	0	0
4	0.31	1.0	63.3	58.0	335	4.34	4.35	2.75	0	0	...	0	1	0	0	0	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
53938	0.86	3.0	61.0	58.0	2757	6.15	6.12	3.74	0	0	...	0	0	0	0	0	1	0	0	0	0
53939	0.75	2.0	62.2	55.0	2757	5.83	5.87	3.64	1	0	...	0	0	0	0	0	1	0	0	0	0
53940	0.71	3.0	60.5	55.0	2756	5.79	5.74	3.49	0	1	...	0	0	0	0	1	0	0	0	0	0
53941	0.71	3.0	59.8	62.0	2756	5.74	5.73	3.43	0	0	...	0	0	0	0	1	0	0	0	0	0
53942	0.70	4.0	60.5	59.0	2757	5.71	5.76	3.47	0	1	...	0	0	0	0	0	0	0	1	0	0

53943 rows × 23 columns

Return a boolean value indicating whether the dataset has missing values. Do not overwrite df.

In [81]:
```
# creating bool series True for NaN values - partial execution
bool_data = df1.isnull()

# filtering data and displaying data only with Gender = NaN
bool_data
```

Out[81]:

	carat	cut	depth	table	price	x	y	z	D	E	...	I	J	I1	IF	SI1	SI2	VS1	VS2	VVS1	V
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
53938	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	
53939	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	
53940	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	
53941	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	
53942	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	

53943 rows × 23 columns

Show average of all columns grouped by "cut" in a same DataFrame table. Do not overwrite df.

In [82]:
```
# compute avg and groupy - partial exceution notb applying to all columns

avg = df1.groupby(['cut']).mean()

avg
```

Out[82]:

cut	carat	depth	table	price	x	y	z	D	E	F	...	I	J	
0.0	1.046137	64.041677	59.053789	4358.757764	6.246894	6.182652	3.982770	0.101242	0.139130	0.193789	...	0.108696	0.073913	0.1
1.0	0.849185	62.365879	58.694639	3928.864452	5.838785	5.850744	3.639507	0.134937	0.190175	0.185283	...	0.106400	0.062576	0.0
2.0	0.702837	61.709401	55.951668	3457.541970	5.507451	5.520080	3.401448	0.131502	0.181105	0.177532	...	0.097118	0.041576	0.0
3.0	0.891929	61.264511	58.746060	4583.992605	5.973857	5.944848	3.647097	0.116218	0.169506	0.169071	...	0.103531	0.058580	0.0
4.0	0.806373	61.818166	57.956236	3981.658529	5.740694	5.770025	3.559794	0.125217	0.198709	0.179095	...	0.099644	0.056112	0.0

5 rows × 22 columns

Show the sum of carat and mean of price grouped by "cut" in a same DataFrame table. Do not overwrite df.

In [83]:
```
# sum and mean of carat and cut columns

sum_cut = df1.groupby(['cut']).sum()

new = sum_cut['carat'], avg['price']

new
```

Out[83]:
```
(cut
 0.0     1684.28
 1.0     4166.10
 2.0    15146.84
 3.0    12302.37
 4.0     9743.40
 Name: carat, dtype: float64,
 cut
 0.0    4358.757764
 1.0    3928.864452
 2.0    3457.541970
 3.0    4583.992605
 4.0    3981.658529
 Name: price, dtype: float64)
```

Print the (filtered) dataset such that (column x < 6 and cut is even). Do not overwrite df.

In [84]:
```
filtered_df = df1[(df1.x < 6) & (df1.cut%2 ==0)]
filtered_df
```

Out[84]:

	carat	cut	depth	table	price	x	y	z	D	E	...	I	J	I1	IF	SI1	SI2	VS1	VS2	VVS1	VVS2
0	0.23	2.0	61.5	55.0	326	3.95	3.98	2.43	0	1	...	0	0	0	0	0	1	0	0	0	0
5	0.24	4.0	62.8	57.0	336	3.94	3.96	2.48	0	0	...	0	1	0	0	0	0	0	0	0	1
6	0.24	4.0	62.3	57.0	336	3.95	3.98	2.47	0	0	...	1	0	0	0	0	0	0	0	1	0
7	0.26	4.0	61.9	55.0	337	4.07	4.11	2.53	0	0	...	0	0	0	0	1	0	0	0	0	0
8	0.22	0.0	65.1	61.0	337	3.87	3.78	2.49	0	1	...	0	0	0	0	0	0	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
53933	0.70	4.0	61.2	59.0	2757	5.69	5.72	3.49	0	1	...	0	0	0	0	0	0	0	1	0	0
53935	0.72	2.0	60.8	57.0	2757	5.75	5.76	3.50	1	0	...	0	0	0	0	1	0	0	0	0	0
53937	0.70	4.0	62.8	60.0	2757	5.66	5.68	3.56	1	0	...	0	0	0	0	1	0	0	0	0	0
53939	0.75	2.0	62.2	55.0	2757	5.83	5.87	3.64	1	0	...	0	0	0	0	0	1	0	0	0	0
53942	0.70	4.0	60.5	59.0	2757	5.71	5.76	3.47	0	1	...	0	0	0	0	0	0	0	1	0	0

22383 rows × 23 columns

Part 2: Regression

Assign X to be the whole df without column price and y to be the column price. Split X and y into X_train, X_test, y_train, and y_test with **random_state=1** and test_size=0.2.
Reference: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)

In [85]:
```
X = df1.loc[:, df1.columns!='price'].values
y = df1['price'].values

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=1)
```

Write a class My_LinearR that implements LinearRegression algorithm. You are required to have the following attributes

- Method:
  - fit
  - predict

Reference: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)
Using a pre-built library yields no credit. You have to write everything from scratch

In [86]:
```
class My_LinearR:
# x - input, b - bias, y - output, w - weight
# epoch - Value to start with can be any, on this bases further calculations are made for every weights

    def Cost(self,x,y,w,b):

        cost = np.sum((((x.dot(w) + b) - y) ** 2) / (2*len(y)))

        return cost

    def fit(self,x, y):

        w = np.zeros(X_train.shape[1])
        b = 0
        epoch = 10000
        cost_list = [0]*10000

        for e in range(0,epoch):

            h = x.dot(w) + b
            loss = h - y
            l = len(y)

            weight = x.T.dot(loss) / l
            bias = np.sum(loss) / l

            w = w - 0.002*weight
            b = b - 0.002*bias

            cost = self.Cost(x, y, w, b)

            cost_list[e] = cost

            if (e%(epoch/10)==0):
                print("Cost of the model at the epoch number "+ str(e) +" is:",cost)

        return w, b, cost_list



    def predict(self,x,w,b):
```

```
        return x.dot(w) + b

    pass
```

In [87]:
```
# Run the code
reg = My_LinearR()
w,b,c = reg.fit(X_train,y_train)
y_pred = reg.predict(X_test,w,b)
```

```
Cost of the model at the epoch number 0 is: 15576889.319478331
Cost of the model at the epoch number 1000 is: 1228486.0267718357
Cost of the model at the epoch number 2000 is: 987562.553648995
Cost of the model at the epoch number 3000 is: 929208.2128888781
Cost of the model at the epoch number 4000 is: 884437.092900087
Cost of the model at the epoch number 5000 is: 847618.3185756484
Cost of the model at the epoch number 6000 is: 817214.6288517416
Cost of the model at the epoch number 7000 is: 792062.0554145807
Cost of the model at the epoch number 8000 is: 771213.7849361943
Cost of the model at the epoch number 9000 is: 753896.5123140505
```

Part 3: Metric

Use three of regression metrics in https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics) to compute errors between y_test and y_pred

In [88]:
```
from sklearn.metrics import mean_absolute_error

mean_absolute_error(y_test, y_pred)
```

Out[88]:
```
865.7149983541623
```

In [89]:
```
from sklearn.metrics import mean_squared_error

mean_squared_error(y_test, y_pred)
```

Out[89]:
```
1455206.331700349
```

In [90]:
```
from sklearn.metrics import r2_score

r2_score(y_test, y_pred)
```

Out[90]:
```
0.907009830825203
```

Which one do you think is the best metric of the three? Explain.
Answer:

Answer:

I guess, score R2 is much better than having a fixed boundary which yields understanding the system accuracy.

In [ ]: