Name :Suraj Sawant

Roll no. TEB38

Practical A-4

```
    Create a Linear Regression Model using Python/R to predict home prices using Boston Housing #Dataset
(https://www.kaggle.com/c/boston-housing). The Boston Housing dataset contains
#information about various houses in Boston through different parameters. There are 506 samples #and
14 feature variables in this dataset.
#The objective is to predict the value of prices of the house using the given features.
```

```
import pandas as pd df =
pd.read_csv("/content/HousingData.csv")
df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	NaN	36.2

```
# crim: per capita crime rate by town
# zn: proportion of residential land zoned for lots over 25,000 sq.ft.
# indus: proportion of non-retail business acres per town.
# chas: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).
# nox: nitrogen oxides concentration (parts per 10 million).
# rm: average number of rooms per dwelling.
# age: proportion of owner-occupied units built prior to 1940.
# dis: weighted mean of distances to five Boston employment centres.
# rad: index of accessibility to radial highways.
# tax: full-value property-tax rate per $10,000.
# ptratio: pupil-teacher ratio by town.
# black: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town.
# lstat: lower status of the population (percent).
#: medv: median value of owner-occupied homes in $1000s.
```

```
df.shape
```

```
(506, 14)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505 Data
columns (total 14 columns):
 # Column Non-Null Count Dtype
---        ---------------
 0   CRIM    486 non-null       float64
 1   ZN 486 non-null float64 2 INDUS 486 non-null float64
  3         CHAS      486 non-null float64
  4         NOX       506 non-null float64
  5         RM        506 non-null float64
  6         AGE       486 non-null float64
  7         DIS       506 non-null float64
  8         RAD       506 non-null int64
  9         TAX       506 non-null int64
  10        PTRATIO 506 non-null float64
  11        B         506 non-null float64
  12        LSTAT     486 non-null float64 13 MEDV 506 non-null float64 dtypes: float64(12),
 int64(2) memory usage: 55.5 KB
```

```
df.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO

count	486.000000	486.000000	486.000000	486.000000	506.000000	506.000000	486.000000	506.000000	506.000000	506.000000	506.000000	506
mean	3.611874	11.211934	11.083992	0.069959	0.554695	6.284634	68.518519	3.795043	9.549407	408.237154	18.455534	356
std	8.720192	23.388876	6.835896	0.255340	0.115878	0.702617	27.999513	2.105710	8.707259	168.537116	2.164946	91
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0
25%	0.081900	0.000000	5.190000	0.000000	0.449000	5.885500	45.175000	2.100175	4.000000	279.000000	17.400000	375
50%	0.253715	0.000000	9.690000	0.000000	0.538000	6.208500	76.800000	3.207450	5.000000	330.000000	19.050000	391
75% max	3.560263	12.500000	18.100000	0.000000	0.624000	6.623500	93.975000	5.188425	24.000000	666.000000	20.200000	396
	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396

```
# Data Cleaning
df=df.fillna(df.mean())
df.info()
```

```
<class  'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505 Data
columns (total 14 columns):
    #  Column Non-Null Count  Dtype
 ---            ---------------
  0         CRIM      506 non-null        float64
  1         ZN        506 non-null  float64
  2         INDUS     506 non-null  float64
  3         CHAS      506 non-null  float64
  4         NOX       506 non-null  float64
  5         RM        506 non-null  float64
  6         AGE       506 non-null  float64
  7         DIS       506 non-null  float64
  8         RAD       506 non-null  int64
  9         TAX       506 non-null  int64
  10        PTRATIO 506 non-null  float64
  11        B         506 non-null  float64
  12        LSTAT     506 non-null  float64
  13        MEDV      506 non-null  float64 dtypes: float64(12), int64(2) memory usage: 55.5 KB
```

```
corr=df.corr() #Find the correlation (relationship) between each column in the DataFrame
corr
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
CRIM	1.000000	-0.182930	0.391161	-0.052223	0.410377	-0.215434	0.344934	-0.366523	0.608886	0.566528	0.273384	-0.370163	0.434
ZN	-0.182930	1.000000	-0.513336	-0.036147	-0.502287	0.316550	-0.541274	0.638388	-0.306316	-0.308334	-0.403085	0.167431	-0.407
INDUS	0.391161	-0.513336	1.000000	0.058035	0.740965	-0.381457	0.614592	-0.699639	0.593176	0.716062	0.384806	-0.354597	0.567
CHAS	-0.052223	-0.036147	0.058035	1.000000	0.073286	0.102284	0.075206	-0.091680	0.001425	-0.031483	-0.109310	0.050055	-0.046
NOX	0.410377	-0.502287	0.740965	0.073286	1.000000	-0.302188	0.711461	-0.769230	0.611441	0.668023	0.188933	-0.380051	0.572
RM	-0.215434	0.316550	-0.381457	0.102284	-0.302188	1.000000	-0.241351	0.205246	-0.209847	-0.292048	-0.355501	0.128069	-0.602
AGE	0.344934	-0.541274	0.614592	0.075206	0.711461	-0.241351	1.000000	-0.724353	0.449989	0.500589	0.262723	-0.265282	0.574
DIS	-0.366523	0.638388	-0.699639	-0.091680	-0.769230	0.205246	-0.724353	1.000000	-0.494588	-0.534432	-0.232471	0.291512	-0.483
RAD	0.608886	-0.306316	0.593176	0.001425	0.611441	-0.209847	0.449989	-0.494588	1.000000	0.910228	0.464741	-0.444413	0.468
TAX	0.566528	-0.308334	0.716062	-0.031483	0.668023	-0.292048	0.500589	-0.534432	0.910228	1.000000	0.460853	-0.441808	0.524
PTRATIO	0.273384	-0.403085	0.384806	-0.109310	0.188933	-0.355501	0.262723	-0.232471	0.464741	0.460853	1.000000	-0.177383	0.373
B	-0.370163	0.167431	-0.354597	0.050055	-0.380051	0.128069	-0.265282	0.291512	-0.444413	-0.441808	-0.177383	1.000000	-0.368
LSTAT	0.434044	-0.407549	0.567354	-0.046166	0.572379	-0.602962	0.574893	-0.483429	0.468440	0.524545	0.373343	-0.368886	1.000
MEDV	-0.379695	0.365943	-0.478657	0.179882	-0.427321	0.695360	-0.380223	0.249929	-0.381626	-0.468536	-0.507787	0.333461	-0.721

```
df.rename(columns={'MEDV':'PRICE'}, inplace=True)

df.info()
```

```
<class  'pandas.core.frame.DataFrame'>
RangeIndex:  506  entries,  0  to  505
Data columns (total 14 columns):
 # Column Non-Null Count Dtype


 0   CRIM    506 non-null        float64
```

```
1   ZN      506 non-null       float64
2   INDUS   506 non-null       float64 3 CHAS      506 non-null        float64
4   NOX     506 non-null       float64
5   RM      506 non-null       float64
6   AGE     506 non-null       float64
7   DIS     506 non-null       float64
8   RAD     506 non-null       int64
9   TAX     506 non-null       int64
10  PTRATIO 506 non-null       float64
11  B       506 non-null       float64
12  LSTAT   506 non-null       float64
13  PRICE   506 non-null       float64
dtypes: float64(12), int64(2) memory
usage: 55.5 KB
```

```
x=df.drop('PRICE',axis=1) # Independent Columns
y=df['PRICE'] #Target Column Price
```

```
x.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.980000
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.140000
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.030000
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.940000
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	12.715432

```
y.head()
```

	PRICE
0	24.0
1	21.6
2	34.7
3	33.4
4	36.2

dtype: float64

```
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest=train_test_split(x,y,test_size=0.2, random_state=0)

xtrain.shape
```

```
(404, 13)
```

```
ytrain.shape
```

```
(404,)
```

```
xtest.shape
```

```
(102, 13)
```

```
ytest.shape
```

```
(102,)
```

```
from sklearn.linear_model import LinearRegression


# Fitting Multi Linear regression model to training model
regressor=LinearRegression()         regressor.fit(xtrain,
ytrain)
```

```
▾ LinearRegression
LinearRegression()
```

```
# predicting the test set results ypred=regressor.predict(xtest)

from sklearn.metrics import mean_squared_error, mean_absolute_error
mse = mean_squared_error(ytest, ypred) mae =
```

```
mean_absolute_error(ytest,ypred) print("Mean Square Error : ", mse)
print("Mean Absolute Error : ", mae)
```

```
Mean Square Error : 34.987389544238766
 Mean Absolute Error : 3.961621123959121
```

```
from sklearn.metrics import r2_score
```

```
r2=r2_score(ytest, ypred)
```

```
r2
```

```
0.5703296053895559
```

```
# An R² of 0.57 (57%) suggests a moderate fit of the model.
#The model is reasonably good at explaining the variability in the data,
#but there is still a significant portion (43%) that is not explained by the model
```