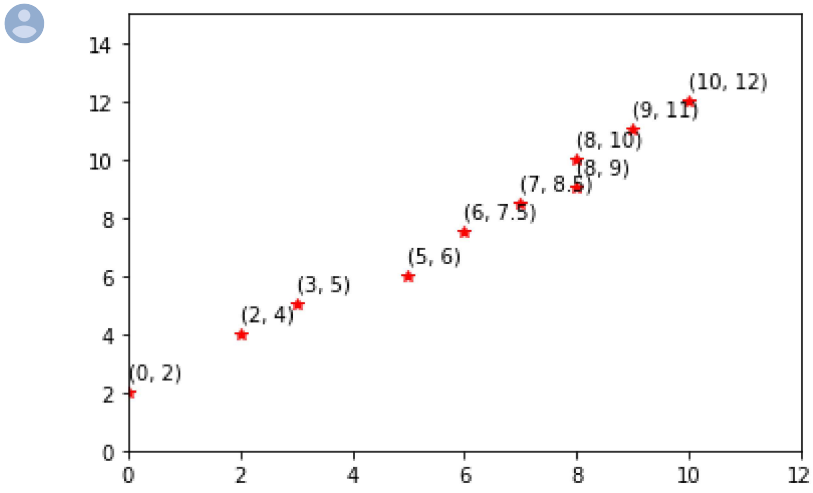
Practical 4 PART A and B

> B-76

> Shivam Gupta

PART(A): Linear Regression

```
# 1. Consider a set of points x and the predicted values y as given below:
# x y
# 0 2
# 2 4
# 3 5
# 5 6
# 6 7.5
# 7 8.5
# 8 9
# 8 10
# 9 11
# 10 12

# (a) Plot the data.

x_org=[0,2,3,5,6,7,8,8,9,10]
y_org=[2,4,5,6,7.5,8.5,9,10,11,12]

import matplotlib.pyplot as plt
plt.plot(x_org, y_org, 'r*')
plt.axis([0, 12, 0, 15])

for i, j in zip(x_org, y_org):
   plt.text(i, j+0.5, '({}, {})'.format(i, j))

plt.show()
```



```
# (b) Use LinearRegression model from sklearn library to perform linear regression.

import numpy as np
from sklearn.linear_model import LinearRegression
x = np.array(x_org).reshape((-1, 1))
y = np.array(y_org)
print('x is',x)
print('y is',y)

model = LinearRegression()
model.fit(x, y)

r_sq = model.score(x, y)
print(f"coefficient of determination: {r_sq}")

print(f"intercept: {model.intercept_}")

print(f"slope: {model.coef_}")
```

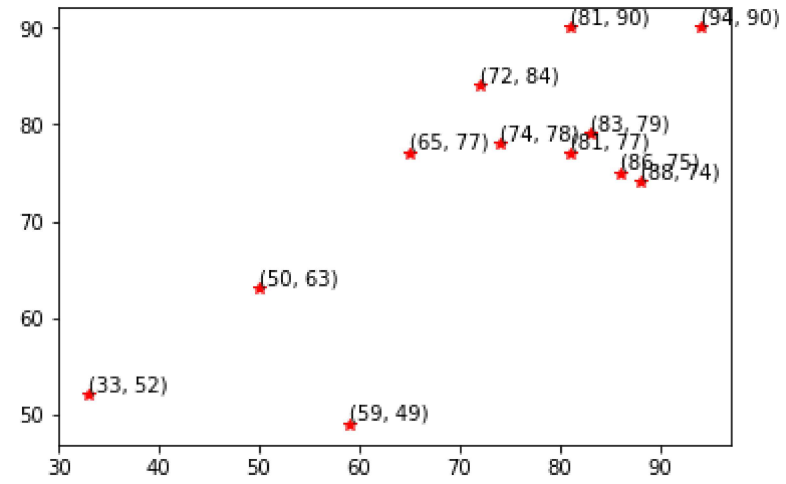
```
x is [[ 0]
 [ 2]
 [ 3]
 [ 5]
 [ 6]
 [ 7]
 [ 8]
 [ 8]
 [ 9]
 [10]]
y is [ 2.   4.   5.   6.   7.5  8.5  9.  10.  11.  12. ]
coefficient of determination: 0.9832917172807847
intercept: 1.8274058577405858
slope: [0.97803347]
```

```
# (c) Consider a data value for x and predict the value of y using the above model.

y_pred = model.predict(x)
print(f"predicted response:\n{y_pred}")
```

```
predicted response:
[ 1.82740586  3.7834728   4.76150628  6.71757322  7.69560669  8.67364017
  9.65167364  9.65167364 10.62970711 11.60774059]
```

```
# 2. The given table shows the midterm and final exam grades obtained for students in a
# database course.
# X [midterm exam] Y [Final Exam]
# 72 84
# 50 63
# 81 77
# 74 78

# 94 90
# 86 75
# 59 49
# 83 79
```

```
# 65 77
# 33 52
# 88 74
# 81 90

# (a) Plot the data. Do x and y seem to have a linear relationship?

x=[72,50,81,74,94,86,59,83,65,33,88,81]
y=[84,63,77,78,90,75,49,79,77,52,74,90]

plt.plot(x, y, 'r*')

for i, j in zip(x, y):
   plt.text(i, j+0.5, '({}, {})'.format(i, j))

plt.show()
```



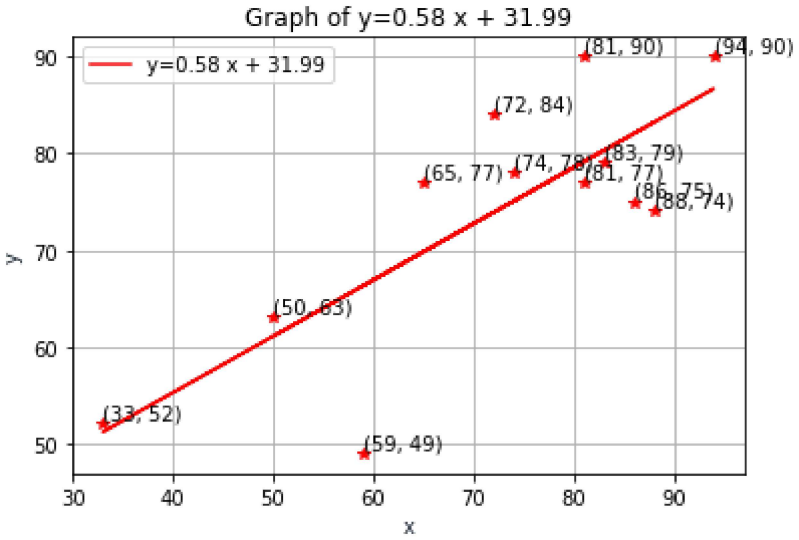
Not linear

```
# (b) Use the method of least squares to find an equation for the prediction of a student’
# final exam grade based on the student’s midterm grade in the course.
# Write a function in python to compute the coefficients and equation. [Do not use the
# inbuild library method.]

n=len(x)
sumx=sum(x)
sumy=sum(y)
sumxy=0
xsqsum=0

for i in range(n):
  sumxy+=x[i]*y[i]
  xsqsum+=x[i]*x[i]
sumxsq=sumx*sumx

m=(n*sumxy-(sumx)*(sumy))/(n*xsqsum-sumxsq)


print("Slope: m = ",m)

xmean=np.mean(x)
```

```
ymean=np.mean(y)

c=ymean-xmean*m
print("Y-intercept: c=",c)
```

```
Slope: m =  0.5816000773918932
Y-intercept: c= 32.027861081551706
```

```
# (c) Also show the plot with the datapoints and the obtained linear equation line.

x=[72,50,81,74,94,86,59,83,65,33,88,81]
y=[84,63,77,78,90,75,49,79,77,52,74,90]

plt.plot(x, y, 'r*')

for i, j in zip(x, y):
   plt.text(i, j+0.5, '({}, {})'.format(i, j))

x = np.array(x)
y = m*x+c
plt.plot(x, y, '-r', label='y=0.58 x + 31.99')
plt.title(f'Graph of y=0.58 x + 31.99')
plt.xlabel('x', color='#1C2833')
plt.ylabel('y', color='#1C2833')
plt.legend(loc='upper left')
plt.grid()
plt.show()
```




```
# (d) Predict the final exam grade of student who received an 86 in the midterm exam
# based on the equation of least squares.

print("final exam grade of student who received an 86 in the midterm exam is ",m*86+c)
```

```
final exam grade of student who received an 86 in the midterm exam is  82.04546773725
```

```
# Do as Directed [Multiple Linear regression]:
```

```
# 3. Perform Multiple Linear regression on cars.csv dataset.
# (a) Analyse each column of the datset using appropriate visualization technique.

import pandas as pd
import matplotlib.pyplot as mp
import seaborn as sns # for visualization
import matplotlib.pyplot as plt

# take data
df = pd.read_csv("cars.csv")
print(df)

c = df.corr()
print("\n\n",c)
sns.heatmap(c,annot=True,cmap='Blues')
plt.show()
```

	Car	Model	Volume	Weight	CO2
0	Toyoty	Aygo	1000	790	99
1	Mitsubishi	Space Star	1200	1160	95
2	Skoda	Citigo	1000	929	95
3	Fiat	500	900	865	90
4	Mini	Cooper	1500	1140	105
5	VW	Up!	1000	929	105
6	Skoda	Fabia	1400	1109	90
7	Mercedes	A-Class	1500	1365	92
8	Ford	Fiesta	1500	1112	98
9	Audi	A1	1600	1150	99
10	Hyundai	I20	1100	980	99
11	Suzuki	Swift	1300	990	101
12	Ford	Fiesta	1000	1112	99
13	Honda	Civic	1600	1252	94
14	Hundai	I30	1600	1326	97

```
# (b) Consider only weight and volume columns as multiple variables to predict the
# CO2 emission.

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import linear_model

shape=df.shape
print(shape,"\n")
print(df.columns,"\n")
print(df.describe(),"\n\n")

# spliting Dataset in Dependent & Independent Variables
X = df[['Weight','Volume']]
y = df['CO2']

print("Independent Variables: \n\n",X)
print("Dependent Variables: \n\n",y)


x_train,x_test,y_train,y_test = train_test_split(X,y,train_size=0.7,random_state=0)

regr = linear_model.LinearRegression()
regr.fit(X, y)
print('\nModel has been trained successfully')

y_pred = regr.predict(x_test)
print("\nPredicted CO2:\n",y_pred)
```

```
(36, 5)

Index(['Car', 'Model', 'Volume', 'Weight', 'CO2'], dtype='object')
```

	Volume	Weight	CO2
count	36.000000	36.000000	36.000000
mean	1611.111111	1292.277778	102.027778
std	388.975047	242.123889	7.454571
min	900.000000	790.000000	90.000000
25%	1475.000000	1117.250000	97.750000
50%	1600.000000	1329.000000	99.000000

```
75%    2000.000000  1418.250000  105.000000
max    2500.000000  1746.000000  120.000000


Independent Variables:

     Weight  Volume
0       790    1000
1      1160    1200
2       929    1000
3       865     900
4      1140    1500
5       929    1000
6      1109    1400
7      1365    1500
8      1112    1500
9      1150    1600
10      980    1100
11      990    1300
12     1112    1000
13     1252    1600
14     1326    1600
15     1330    1600
16     1365    1600
17     1280    2200
18     1119    1600
19     1328    2000
20     1584    1600
21     1428    2000
22     1365    2100
23     1415    1600
24     1415    2000
25     1465    1500
26     1490    2000
27     1725    2000
28     1523    1600
29     1705    2000
30     1605    2100
31     1746    2000
32     1235    1600
33     1390    1600
34     1405    1600
35     1395    2500
Dependent Variables:

 0      99
```

```
# (c) Predict the CO2 emission of a car where the weight is 2300kg, and the volume is
# 1300ccm. Also conclude about the obtained result.

x_train,x_test,y_train,y_test = train_test_split(X,y,train_size=0.7,random_state=0)

reg = linear_model.LinearRegression()
reg.fit(X, y)

prediction = reg.predict([[2300, 1300]])
print("\n\nCO2 emission of car for weight=2300kg, volume=1300cm3 is => ",prediction,"\n\n"
```

```
CO2 emission of car for weight=2300kg, volume=1300cm3 is =>  [107.2087328]


C:\Users\Dell\AppData\Roaming\Python\Python39\site-packages\sklearn\base.py:450: User
  warnings.warn(
```

```
# (d) Perform prediction on few more values.

prediction = reg.predict([[1300, 300]])
print("\nCO2 emission of car for weight=2300kg, volume=1300cm3 is => ",prediction,"\n")

prediction = reg.predict([[200, 130]])
print("\nCO2 emission of car for weight=2300kg, volume=1300cm3 is => ",prediction,"\n")

prediction = reg.predict([[330, 83]])
print("\nCO2 emission of car for weight=2300kg, volume=1300cm3 is => ",prediction,"\n")
```

```
CO2 emission of car for weight=2300kg, volume=1300cm3 is =>  [91.852528]


CO2 emission of car for weight=2300kg, volume=1300cm3 is =>  [82.21959222]


CO2 emission of car for weight=2300kg, volume=1300cm3 is =>  [82.83436827]

C:\Users\Dell\AppData\Roaming\Python\Python39\site-packages\sklearn\base.py:450: User
  warnings.warn(
C:\Users\Dell\AppData\Roaming\Python\Python39\site-packages\sklearn\base.py:450: User
  warnings.warn(
C:\Users\Dell\AppData\Roaming\Python\Python39\site-packages\sklearn\base.py:450: User
  warnings.warn(
```

```
# (e) Show the coefficient obtained and conclude.
print("Coefficient obtained of multiple regression are = ",reg.coef_)
```

```
Coefficient obtained of multiple regression are =  [0.00755095 0.00780526]
```

```
# 4. Perform linear regression on the dataset [use dataset: kc_house_data.csv]

# (a) Load the dataset, display it, visualize various columns and explain the dataset
# composition.

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv("kc_house_data.csv")
df
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7
2	5631500400	20150225T000000	180000.0	2	1.00	770	10
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8
...	...	...	...	...	...	...	...
21608	263000018	20140521T000000	360000.0	3	2.50	1530	
21609	6600060120	20150223T000000	400000.0	4	2.50	2310	5
21610	1523300141	20140623T000000	402101.0	2	0.75	1020	1
21611	291310100	20150116T000000	400000.0	3	2.50	1600	2
21612	1523300157	20141015T000000	325000.0	2	0.75	1020	1

21613 rows × 21 columns

```
print("Summary of the DataFrame:\n")
df.info()
```

```
Summary of the DataFrame:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21613 non-null  int64  
 1   date           21613 non-null  object 
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64  
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64  
 6   sqft_lot       21613 non-null  int64  
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64  
 9   view           21613 non-null  int64  
 10  condition      21613 non-null  int64  
 11  grade          21613 non-null  int64  
 12  sqft_above     21613 non-null  int64  
 13  sqft_basement  21613 non-null  int64  
 14  yr_built       21613 non-null  int64  
 15  yr_renovated   21613 non-null  int64  
 16  zipcode        21613 non-null  int64  
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64  
 20  sqft_lot15     21613 non-null  int64  
```

```
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

```
# (b) Describe the dataset

df.describe()
```

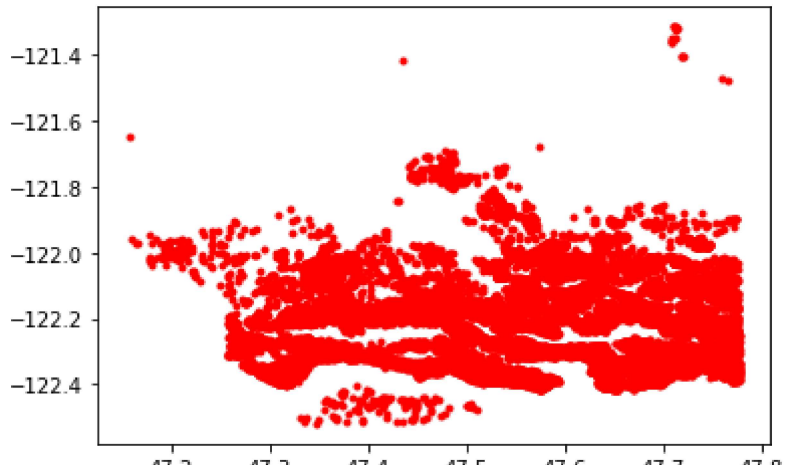
	id	price	bedrooms	bathrooms	sqft_living	sqft_
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e-
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e-
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e-
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e-
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e-
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e-
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e-
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e-

```
# (c) Houses with how many bedrooms are most sold?

print("Houses with 3 Bedrooms are most sold.")
df['bedrooms'].value_counts()
```

```
Houses with 3 Bedrooms are most sold.
3     9824
4     6882
2     2760
5     1601
6      272
1      199
7       38
0       13
8       13
9        6
10       3
11       1
33       1
Name: bedrooms, dtype: int64
```

```
# (d) Visualizing the location of the houses based on latitude and longitude. Use
# appropriate plot.

lati=df['lat']
longi=df['long']
```

```
plt.plot(lati, longi, 'r.')
plt.show()
```




```
# (e) Find the correlated features/ columns

c = df.corr()
print("\n",c)
sns.heatmap(c,cmap='Blues')
plt.show()
```

```
                          id     price  bedrooms  bathrooms  sqft_living  sqft_lot  \
id             1.000000 -0.016762  0.001286   0.005160    -0.012258 -0.132109
price         -0.016762  1.000000  0.308350   0.525138     0.702035  0.089661
bedrooms       0.001286  0.308350  1.000000   0.515884     0.576671  0.031703
bathrooms      0.005160  0.525138  0.515884   1.000000     0.754665  0.087740
sqft_living   -0.012258  0.702035  0.576671   0.754665     1.000000  0.172826
sqft_lot      -0.132109  0.089661  0.031703   0.087740     0.172826  1.000000
floors         0.018525  0.256794  0.175429   0.500653     0.353949 -0.005201
waterfront    -0.002721  0.266369 -0.006582   0.063744     0.103818  0.021604
view           0.011592  0.397293  0.079532   0.187737     0.284611  0.074710
condition     -0.023783  0.036362  0.028472  -0.124982    -0.058753 -0.008958
grade          0.008130  0.667434  0.356967   0.664983     0.762704  0.113621
sqft_above    -0.010842  0.605567  0.477600   0.685342     0.876597  0.183512
sqft_basement -0.005151  0.323816  0.303093   0.283770     0.435043  0.015286
yr_built       0.021380  0.054012  0.154178   0.506019     0.318049  0.053080
yr_renovated  -0.016907  0.126434  0.018841   0.050739     0.055363  0.007644
zipcode       -0.008224 -0.053203 -0.152668  -0.203866    -0.199430 -0.129574
lat           -0.001891  0.307003 -0.008931   0.024573     0.052529 -0.085683
long           0.020799  0.021626  0.129473   0.223042     0.240223  0.229521
sqft_living15 -0.002901  0.585379  0.391638   0.568634     0.756420  0.144608
sqft_lot15    -0.138798  0.082447  0.029244   0.087175     0.183286  0.718557
```

```
# (f) Find null values and fill with mean value for all columns

l=[]
for i in df.columns:
  l.append(i)
l=l[2:]

print(l)

for col in l:
  mean=df[col].mean()
  df[col].fillna(value=mean,inplace=True)

df
```

```
['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront',
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7
2	5631500400	20150225T000000	180000.0	2	1.00	770	10

```
# (g) Find dependant and independent data (place in X and y)

X = df[l[1:]]
y = df['price']

print("Independent Variables: \n\n",X)
print("Dependent Variables: \n\n",y)
```

```
Independent Variables:

        bedrooms  bathrooms  sqft_living  sqft_lot  floors  waterfront  view  \
0              3       1.00         1180      5650     1.0           0     0
1              3       2.25         2570      7242     2.0           0     0
2              2       1.00          770     10000     1.0           0     0
3              4       3.00         1960      5000     1.0           0     0
4              3       2.00         1680      8080     1.0           0     0
...          ...        ...          ...       ...     ...         ...   ...
21608          3       2.50         1530      1131     3.0           0     0
21609          4       2.50         2310      5813     2.0           0     0
21610          2       0.75         1020      1350     2.0           0     0
21611          3       2.50         1600      2388     2.0           0     0
21612          2       0.75         1020      1076     2.0           0     0

       condition  grade  sqft_above  sqft_basement  yr_built  yr_renovated  \
0              3      7        1180              0      1955             0
1              3      7        2170            400      1951          1991
2              3      6         770              0      1933             0
3              5      7        1050            910      1965             0
4              3      8        1680              0      1987             0
...          ...    ...         ...            ...       ...           ...
21608          3      8        1530              0      2009             0
21609          3      8        2310              0      2014             0
21610          3      7        1020              0      2009             0
21611          3      8        1600              0      2004             0
21612          3      7        1020              0      2008             0

       zipcode      lat     long  sqft_living15  sqft_lot15
0        98178  47.5112 -122.257           1340        5650
1        98125  47.7210 -122.319           1690        7639
2        98028  47.7379 -122.233           2720        8062
3        98136  47.5208 -122.393           1360        5000
4        98074  47.6168 -122.045           1800        7503
...        ...      ...      ...            ...         ...
21608    98103  47.6993 -122.346           1530        1509
21609    98146  47.5107 -122.362           1830        7200
21610    98144  47.5944 -122.299           1020        2007
21611    98027  47.5345 -122.069           1410        1287
21612    98144  47.5941 -122.299           1020        1357

[21613 rows x 18 columns]
```

```
Dependent Variables:

 0        221900.0
1        538000.0
2        180000.0
3        604000.0
4        510000.0
           ...
21608    360000.0
21609    400000.0
21610    402101.0
21611    400000.0
21612    325000.0
Name: price, Length: 21613, dtype: float64


# (h) Split train and test data

x_train,x_test,y_train,y_test = train_test_split(X,y,train_size=0.7,random_state=0)
print(x_test)
print(y_test)
```

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	\
17384	2	1.50	1430	1650	3.0	0	0	
722	4	3.25	4670	51836	2.0	0	0	
2680	2	0.75	1440	3700	1.0	0	0	
18754	2	1.00	1130	2640	1.0	0	0	
14554	4	2.50	3180	9603	2.0	0	2	
...	...	...	...	...	...	...	...	
18588	3	2.25	1560	8570	1.0	0	0	
6784	4	1.75	2360	6000	1.0	0	0	
14510	3	2.50	2150	25705	1.5	0	0	
18917	3	1.75	1480	8009	1.0	0	0	
11750	3	1.00	1320	9239	1.0	0	0	

	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	\
17384	3	7	1430	0	1999	0	
722	4	12	4670	0	1988	0	
2680	3	7	1200	240	1914	0	
18754	4	8	1130	0	1927	0	
14554	3	9	3180	0	2002	0	
...	...	...	...	...	...	...	
18588	5	7	1080	480	1977	0	
6784	3	7	1280	1080	1955	0	
14510	3	6	2150	0	1980	2009	
18917	3	7	980	500	1980	0	
11750	4	7	1320	0	1968	0	

	zipcode	lat	long	sqft_living15	sqft_lot15
17384	98125	47.7222	-122.290	1430	1650
722	98005	47.6350	-122.164	4230	41075
2680	98107	47.6707	-122.364	1440	4300
18754	98109	47.6438	-122.357	1680	3200
14554	98155	47.7717	-122.277	2440	15261
...	...	...	...	...	...
18588	98004	47.6155	-122.210	2660	9621
6784	98109	47.6465	-122.357	1700	3460
14510	98058	47.4514	-122.089	1850	20160
18917	98032	47.3657	-122.280	1790	7678

```
11750    98092  47.3120 -122.183           1320        8415

[6484 rows x 18 columns]
17384     297000.0
722      1578000.0
2680      562100.0
18754     631500.0
14554     780000.0
           ...
18588    1100000.0
6784      700000.0
14510     380000.0
18917     268000.0
11750     206000.0
Name: price, Length: 6484, dtype: float64
```

```
# (i) Train the model and test it. Find the accuracy.

reg = linear_model.LinearRegression()
reg.fit(X, y)

print('\nModel has been trained successfully')
print('R-squared (testing) ', reg.score(x_test, y_test))
```

```
Model has been trained successfully
R-squared (testing)  0.689837634636076
```

```
# (j) Test the model using some arbitrary input.

# 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'view', 'con
prediction = reg.predict([[4,1.5,1430,1650,3,0,0,3,7,1430,0,2000,0,98125, 47.7,-122.5,1330
print("\nHouse Price Prediction for arbitrary input is => ",prediction,"\n")
```

```
House Price Prediction for arbitrary input is =>  [332067.18606665]

C:\Users\Dell\AppData\Roaming\Python\Python39\site-packages\sklearn\base.py:450: User
  warnings.warn(
```

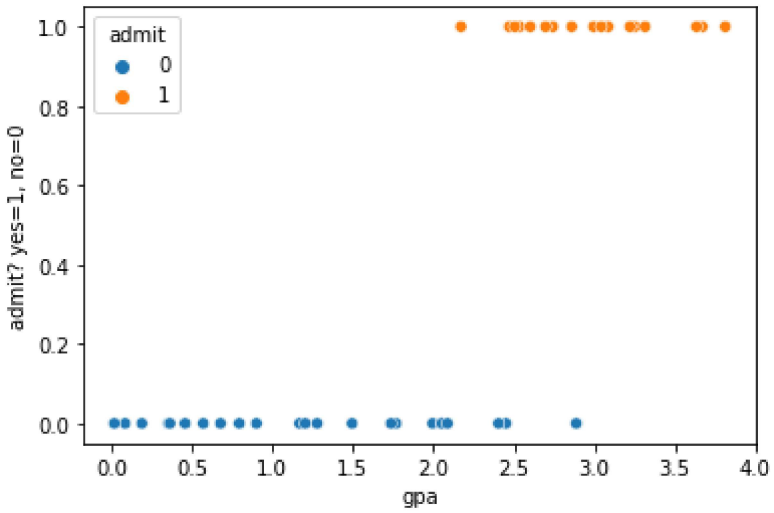
PART(B): Logistic Regression

```
# Do as Directed:
# 5. Perform logistic regression on the admission dataset
# a) Import nyu_admission_fake.csv dataset, display it, visualize various columns

df = pd.read_csv('nyu_admission_fake.csv')
df
```

	student	gpa	admit
0	0	3.085283	1
1	1	0.083008	0
2	2	2.534593	1
3	3	2.995216	1
4	4	1.994028	0
5	5	0.899187	0
6	6	0.792251	0
7	7	3.042123	1
8	8	0.676443	0
9	9	0.353359	0
10	10	2.741439	1
11	11	3.813573	1
12	12	0.015793	0
13	13	2.048769	0
14	14	3.250484	1
15	15	2.450104	0
16	16	2.887021	0
17	17	1.167504	0
18	18	3.671096	1
19	19	2.858303	1
20	20	2.170177	1
21	21	0.568680	0
22	22	1.493363	0
23	23	2.696534	1
24	24	1.767333	0
25	25	1.736056	0
26	26	2.471068	1
27	27	2.052553	0
28	28	2.601589	1
29	29	2.404156	0
30	30	2.220803	1

```
# b) Plot the dataset on gpa vs. admit score.
```

```
import statsmodels.formula.api as smf

sns.scatterplot(x='gpa', y='admit', hue='admit', data=df)
plt.xlabel("gpa")
plt.ylabel("admit? yes=1, no=0")
plt.show()
fig=plt.gcf()
```




```
<Figure size 432x288 with 0 Axes>
```

```
# c) Find the slope and intercept of the line to fit.

lr=smf.ols(formula="admit ~ gpa", data=df).fit()
sns.regplot(x='gpa',y='admit', scatter=False,data=df, color='gray')
sns.scatterplot(x='gpa', y='admit', hue='admit', data=df)
plt.xlabel("gpa")
plt.ylabel("admit? yes=1, no=0")
plt.show()
fig=plt.gcf()


print(lr.params)
```


```
# d) Compute the log odds for each entry.
#    Merge the results with the data as a new column.

# df.drop('log_odds', inplace=True,axis=1)
# df.drop('P', inplace=True,axis=1)

df['log_odds']=2.0*df['gpa']-3
df
```

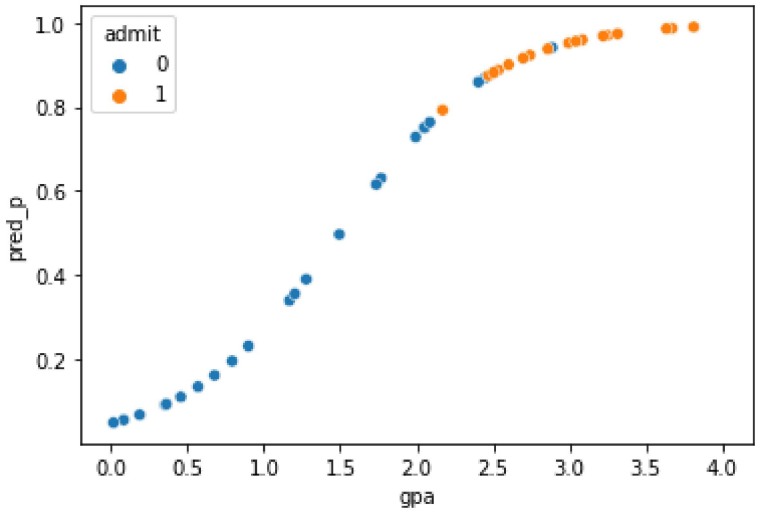
	student	gpa	admit	log_odds
0	0	3.085283	1	3.170566
1	1	0.083008	0	-2.833984
2	2	2.534593	1	2.069186
3	3	2.995216	1	2.990432

```
# e) Using the log odds compute the probability for each entry.

df['pred_p']=np.exp(df['log_odds'])/(1+np.exp(df['log_odds']))
df
```

	student	gpa	admit	log_odds	pred_p
0	0	3.085283	1	3.170566	0.959711
1	1	0.083008	0	-2.833984	0.055515
2	2	2.534593	1	2.069186	0.887872
3	3	2.995216	1	2.990432	0.952140
4	4	1.994028	0	0.988056	0.728704
5	5	0.899187	0	-1.201626	0.231186

```
# f) Plot the probabilities vs gpa graph.

x = np.linspace(0.0,4.0)
y = 1/(1+np.exp(-2.0*x+3))
sns.scatterplot(x='gpa',y='pred_p',hue='admit',data=df)
plt.plot(x,y,'k--',linewidth=0)
```

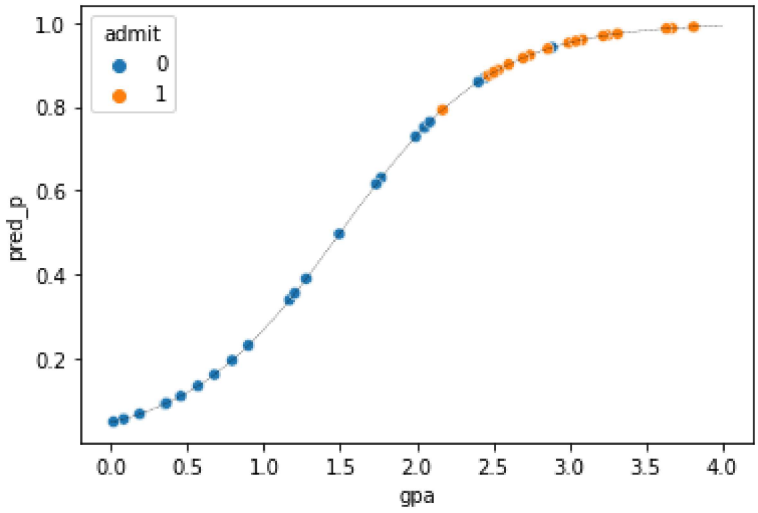
```
[<matplotlib.lines.Line2D at 0x216101f1130>]
```



20	20	2.170177	1	1.340354	0.792548

```
# g) Show the decision boundary of the regression model.

from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris

X = np.array(df['gpa'].tolist()).reshape(-1,1)
y = np.array(df['admit'].tolist())

x1_train, x1_test, y1_train, y1_test = train_test_split(X, y,train_size=0.7)

logreg = LogisticRegression().fit(x1_train,y1_train)
logreg.fit(X,y)

x = np.linspace(0.0,4.0)
y = 1/(1+np.exp(-2.0*x+3))
sns.scatterplot(x='gpa',y='pred_p',hue='admit',data=df)
plt.plot(x,y,'k--',linewidth=0.3)
```

```
[<matplotlib.lines.Line2D at 0x216115f2f70>]
```




```
# h) Show the accuracy of the regressor model.

print("Training set score: {:.3f}".format(logreg.score(x1_train,y1_train)))
print("Test set score: {:.3f}".format(logreg.score(x1_test,y1_test)))
```

```
Training set score: 0.893
Test set score: 0.917
```

```
# 6. Perform logistic regression on the credit card dataset
# a) Download credit card fraud detection dataset using
# https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud?resource=downloadb)

# b) Load the dataset, visualize it, show the data headers

df=pd.read_csv('creditcard.csv')
print("shape = ",df.shape)
df
```

```
shape =  (284807, 31)
```

	Time	V1	V2	V3	V4	V5	V6	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.23
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.07

```
display("The colomn headers: ")
display(df.columns.values)
```

```
'The colomn headers: '
array(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9',
       'V10', 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18',
       'V19', 'V20', 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27',
       'V28', 'Amount', 'Class'], dtype=object)
```

```
# c) Preprocess the dataset if required
# i. Check duplicate data. If found remove it.

df2 = df.drop_duplicates()
df2.shape
```

```
(283726, 31)
```

```
# ii. Remove such colums which are not important. For example: Time

df2.drop(['Time'], axis=1)
```

	V1	V2	V3	V4	V5	V6	V7	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.09
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.08
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.24
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.37
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.27
...	...	...	...	...	...	...	...	
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.30
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.29
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.70
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.67
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.4

283726 rows × 30 columns

```
# iii. Separate the dataset into feature column and target column. The class
# column is the target column and everything else is a feature
```

```
X=df2[['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9',
       'V10', 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18',
       'V19', 'V20', 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27',
       'V28', 'Amount']]
y=df2['Class']

display(X)
display(y)
```

	Time	V1	V2	V3	V4	V5	V6	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.23
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.07
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.79
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.23
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.59
...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.91
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.02
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.29
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.68
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.57

283726 rows × 30 columns

```
0         0
1         0
2         0
3         0
4         0
         ..
284802    0
284803    0
284804    0
284805    0
284806    0
Name: Class, Length: 283726, dtype: int64
```

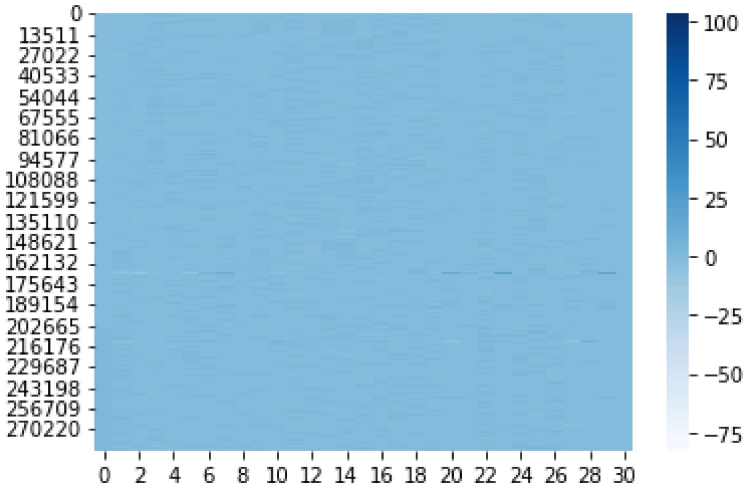
```
# iv. Scale the dataset using standard scaling mechanism.

from numpy import asarray
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
# transform data
df2 = scaler.fit_transform(df2)
print(df2)
```

```
[[-1.99682292 -0.70108232 -0.04168726 ... -0.06584955  0.24419951
```

```
 -0.04086423]
[-1.99682292  0.60879165  0.16413764 ...  0.0432187  -0.34258399
 -0.04086423]
[-1.99680186 -0.7003364  -0.81133678 ... -0.18382429  1.15889967
 -0.04086423]
...
[ 1.64227757  0.98235398 -0.18043304 ... -0.08264021 -0.0822395
 -0.04086423]
[ 1.64227757 -0.12646526  0.32465977 ...  0.31700384 -0.31339058
 -0.04086423]
[ 1.64236181 -0.27686005 -0.1127094  ...  0.03994074  0.51329005
 -0.04086423]]
```

```
# v. Partition dataset into training and testing set as 80%-20%

from sklearn.model_selection import train_test_split
np.random.seed(123)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.80, test_size = 0
```

```
# d) Plot histograms/heatmaps to understand the values of each variable.

sns.heatmap(df2,cmap='Blues')
plt.show()
```




```
# e) Train the model using logistic regression.

from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train, y_train)
```

```
C:\Users\Dell\AppData\Roaming\Python\Python39\site-packages\sklearn\linear_model\_log
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
# f) Obtain the training accuracy.

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_a
import scikitplot as skplt

pred = lr.predict(X_train)
matrix = confusion_matrix(y_train, pred)
print("Training Accuracy is : ")
accuracy_score(y_train, pred)
```

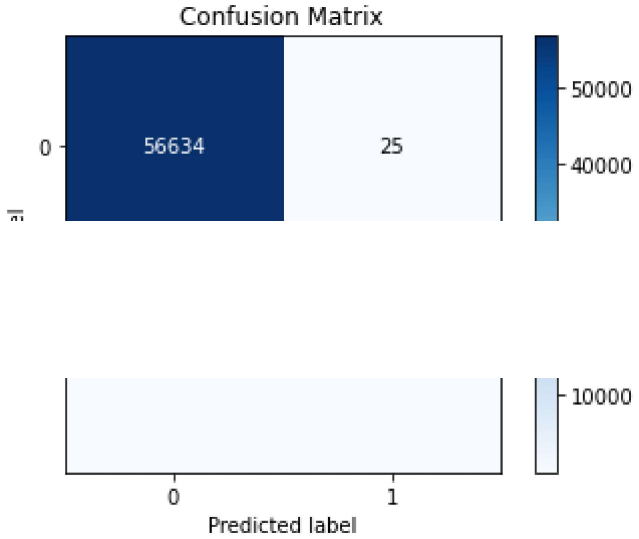
```
Training Accuracy is :  0.9989117983963345
```

```
# g) Test the model. Obtain the testing accuracy.

pred = lr.predict(X_test)
matrix = confusion_matrix(y_test, pred)
print("Testing Accuracy is : ")
accuracy_score(y_test, pred)
```

```
Testing Accuracy is :
0.9990836358509851
```

```
# h) Generate confusion matrix, precision and recall based on TP, FP, FN, TN.

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_a
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train, y_train)

def PrintStats(cmat, y_test, pred):
    tpos = cmat[0][0]
    fneg = cmat[1][1]
    fpos = cmat[0][1]
    tneg = cmat[1][0]

lr.fit(X_train, y_train.values.ravel())
pred = lr.predict(X_test)
cmat = confusion_matrix(y_test, pred)
skplt.metrics.plot_confusion_matrix(y_test, pred)
print (classification_report(y_test, pred))
PrintStats(cmat, y_test, pred)
```

```
C:\Users\Dell\AppData\Roaming\Python\Python39\site-packages\sklearn\linear_model\_log
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\Dell\AppData\Roaming\Python\Python39\site-packages\sklearn\linear_model\_log
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56659
           1       0.71      0.69      0.70        87

    accuracy                           1.00     56746
   macro avg       0.85      0.84      0.85     56746
weighted avg       1.00      1.00      1.00     56746
```



--------XXXX--------