MACHINE LEARNING LAB REPORT

Register No :- RA2512051010001
Name :- SANGRAM LEMBE

INDEX

```
# import libraries

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

```
student_data = pd.read_csv("/content/1a_linear_regression_dirty - 1a_linear_regression_dirty.csv")
```

```
student_data
```

	House_Size	Bedrooms	Age	Price
0	1649.0	1	22.0	245256
1	1458.0	5	3.0	169804
2	1694.0	1	16.0	215222
3	1956.0	3	29.0	212860
4	1429.0	2	9.0	201196
...	...	...	...	...
95	1060.0	1	9.0	138971
96	1588.0	5	11.0	166539
97	1578.0	2	22.0	258096
98	1501.0	2	16.0	277642
99	1429.0	2	7.0	178024

100 rows × 4 columns

```
student_data.head()
```

	House_Size	Bedrooms	Age	Price
0	1649.0	1	22.0	245256
1	1458.0	5	3.0	169804
2	1694.0	1	16.0	215222
3	1956.0	3	29.0	212860
4	1429.0	2	9.0	201196

Mean , Median , Mode

```
std_house_size_mean = student_data['House_Size'].mean()
print("student house size mean :",std_house_size_mean)
```

```
student house size mean : 1468.7878787878788
```

```
std_house_size_median = student_data['House_Size'].median()
print("student house size median :",std_house_size_median)
```

```
student house size median : 1465.0
```

```
std_house_size_mode = student_data['House_Size'].mode()
print("student house size mode :",std_house_size_mode)
```

```
student house size mode : 0    1319.0
1    1360.0
2    1429.0
3    1608.0
4    1746.0
Name: House_Size, dtype: float64
```

```
std_age_mean = student_data['Age'].mean()
print("student age mean :",std_age_mean)
std_age_median = student_data['Age'].median()
print("student age median :",std_age_median)
std_age_mode = student_data['Age'].mode()
print("student age mode :",std_age_mode[0])
```

```
student age mean : 14.797979797979798
student age median : 16.0
student age mode : 1.0
```

```
# student_data['Bedrooms'] = student_data['Bedrooms'].replace('three',3)
```

```
# Method 2
words_to_numbers = {'zero': 0, 'one': 1, 'two': 2, 'three': 3,
                    'four': 4, 'five': 5}
def bedroom_to_numeric(val):
    if isinstance(val, str) and val.lower() in words_to_numbers:
        return words_to_numbers[val.lower()]
    try:
        return int(val)
    except:
        return np.nan  # Missing or invalid value as NaN

student_data['Bedrooms'] = student_data['Bedrooms'].apply(bedroom_to_numeric)
```

Fillna --> Filling Missing Values

```
student_data.House_Size = student_data.House_Size.fillna(std_house_size_mean) #filling Missing value as mean
student_data.Age = student_data.Age.fillna(std_age_median)                    #filling Missing value as median
student_data.Bedrooms = student_data.Bedrooms.fillna(std_age_mode[0])        #filling Missing value as mode
```

```
print(student_data.isnull())  # check is there any Null value is available
```

```
    House_Size  Bedrooms    Age  Price
0        False     False  False  False
1        False     False  False  False
2        False     False  False  False
3        False     False  False  False
4        False     False  False  False
..         ...       ...    ...    ...
95       False     False  False  False
96       False     False  False  False
97       False     False  False  False
98       False     False  False  False
99       False     False  False  False

[100 rows x 4 columns]
```

```
print(student_data.iloc[10])  # pandas index from zero and without heading , it will become row no 10
```

```
House_Size      1360.0
Bedrooms           3.0
Age               21.0
Price         211614.0
Name: 10, dtype: float64
```

Split the Dataset

```
# split the dataset
from sklearn.model_selection import train_test_split

x = student_data[['Age','Bedrooms','House_Size']]
y = student_data['Price']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

```
x_train
```

	Age	Bedrooms	House_Size
55	6.0	4	1779.0
88	27.0	3	1341.0
26	14.0	1	1154.0
42	28.0	1	1465.0
69	29.0	2	1306.0
...	...	...	...
60	24.0	1	1356.0
71	23.0	4	1961.0
14	8.0	2	982.0
92	17.0	3	1289.0
51	28.0	3	1384.0

80 rows × 3 columns

```
x_test
```

	Age	Bedrooms	House_Size
83	18.0	3	1344.0
53	14.0	3	1683.0
70	21.0	1	1608.0
45	19.0	5	1284.0
44	16.0	3	1056.0
39	15.0	2	1559.0
22	19.0	2	1520.0
80	8.0	1	1434.0
10	21.0	3	1360.0
0	22.0	1	1649.0
18	1.0	1	1227.0
30	16.0	5	1319.0
73	3.0	1	1969.0
33	27.0	4	1182.0
90	2.0	4	1529.0
4	9.0	2	1429.0
76	10.0	4	1526.0
77	22.0	5	1410.0
12	20.0	4	1572.0
31	22.0	4	2055.0

```
y_train
```

	Price
55	153370
88	218100
26	227478
42	215888
69	203044
...	...
60	277153
71	206304
14	208251
92	227586
51	136004

80 rows × 1 columns

dtype: int64

```
y_test
```

	Price
83	176773
53	293600
70	232358
45	263320
44	247269
39	283065
22	155343
80	175678
10	211614
0	245256
18	110575
30	268171
73	264027
33	222103
90	168514
4	201196
76	234342
77	203548
12	142371
31	241438

dtype: int64

```
from sklearn.linear_model import LinearRegression
```

```
regressor = LinearRegression()
```

```
regressor.fit(x_train,y_train)
```

```
LinearRegression()
```

```
y_train_prediction = regressor.predict(x_train)
y_train_prediction
```

```
array([283725.59119932, 202103.97333686, 171012.2079902 , 226967.51345084,
       196525.1230752 , 202165.33226358, 274576.22312178, 247007.96611689,
       264588.24222819, 230782.05328647, 207709.3962819 , 291010.88431471,
       202994.80524644, 200050.04510414, 217081.81361999, 226773.37267989,
       228303.51760826, 309522.42705768, 166168.63519698, 179795.68301074,
       134899.90379064, 276706.91401872, 278065.87149347, 154172.80927153,
       256139.36537576, 156453.89354477, 254456.9786788 , 235829.82880287,
       170129.58964056, 124885.83762757, 200504.97261071, 315199.57961373,
       247542.1728143 , 156740.99482485, 206835.43035155, 234343.91868306,
       320995.83065716, 278842.63796889, 245501.57065181, 263311.08047089,
       186198.35777648, 221276.05999675, 288882.83795601, 213380.90035918,
       249629.29703699, 289312.60646635, 207976.98704452, 250894.18700895,
       220886.7262085 , 247295.01884241, 123706.36273912, 244998.04341777,
       253459.19205555, 250258.67407663, 233682.2719267 , 216283.63552603,
       279014.57097535, 233338.40591378, 212130.79869918, 288332.7401072 ,
       168617.54569671, 187772.81493194, 123437.23027771, 217573.55648259,
       223822.70591989, 194683.07515713, 220983.92566345, 270292.45488154,
       154087.39418803, 250765.2052512 , 283848.4370644 ,  88522.76009005,
       284362.64583043, 313280.76903143, 313685.99397108, 207269.87251283,
       315632.96749013, 138841.34771675, 193477.02758529, 209929.60904232])
```

```
prediction_y_test = regressor.predict(x_test)
prediction_y_test
```

```
array([203513.12079244, 266339.22658399, 253974.99107493, 190248.34666962,
       150652.48143648, 244461.32019375, 236896.65194432, 223160.57174962,
       206174.88643463, 261432.21719678, 185696.37224123, 196981.02540959,
       322185.9870776 , 171756.75330346, 238055.07461651, 221085.71931227,
       236741.09750317, 213172.6394106 , 244262.8145039 , 333043.38826223])
```

```
y_test
```

	Price
83	176773
53	293600
70	232358
45	263320
44	247269
39	283065
22	155343
80	175678
10	211614
0	245256
18	110575
30	268171
73	264027
33	222103
90	168514
4	201196
76	234342
77	203548
12	142371
31	241438

dtype: int64

```
print(regressor.coef_)
print(regressor.intercept_)
```

```
[  -95.17034224 -1058.65813607   184.20479181]
-39169.078827729274
```

```
regressor.predict([[1500,3,20]])
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but
  warnings.warn(
array([-181416.47075963])
```

```
 -95.17034224 * 1500 -1058.65813607*3 +  184.20479181 *20
```

```
-142247.39193200998
```

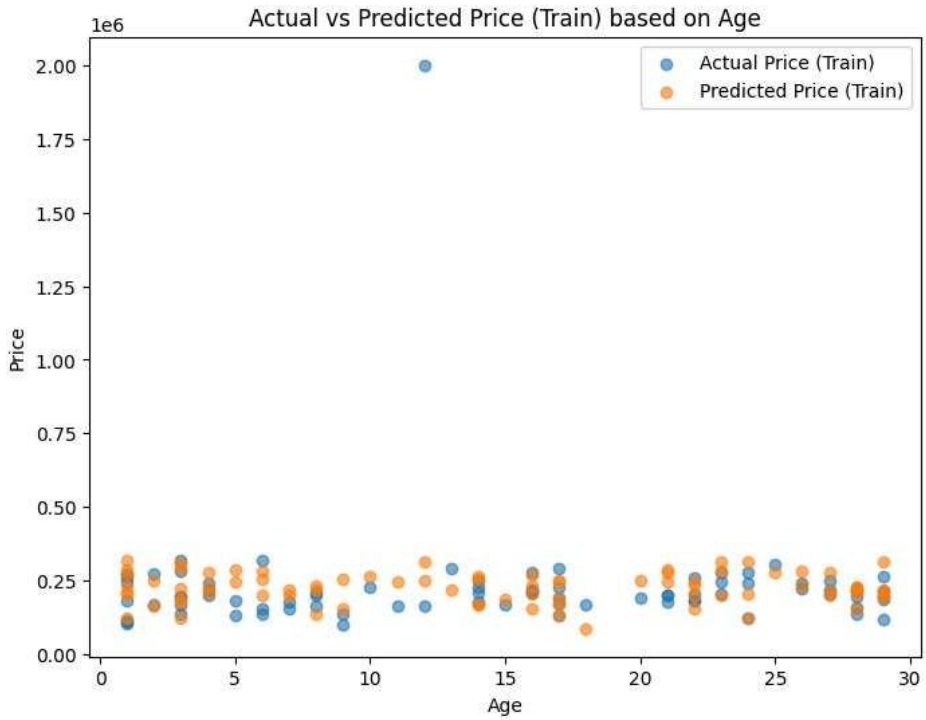
```
yy = regressor.intercept_ + x @ regressor.coef_.T   # here we taking Transpose of matrix

round(yy,2)
```

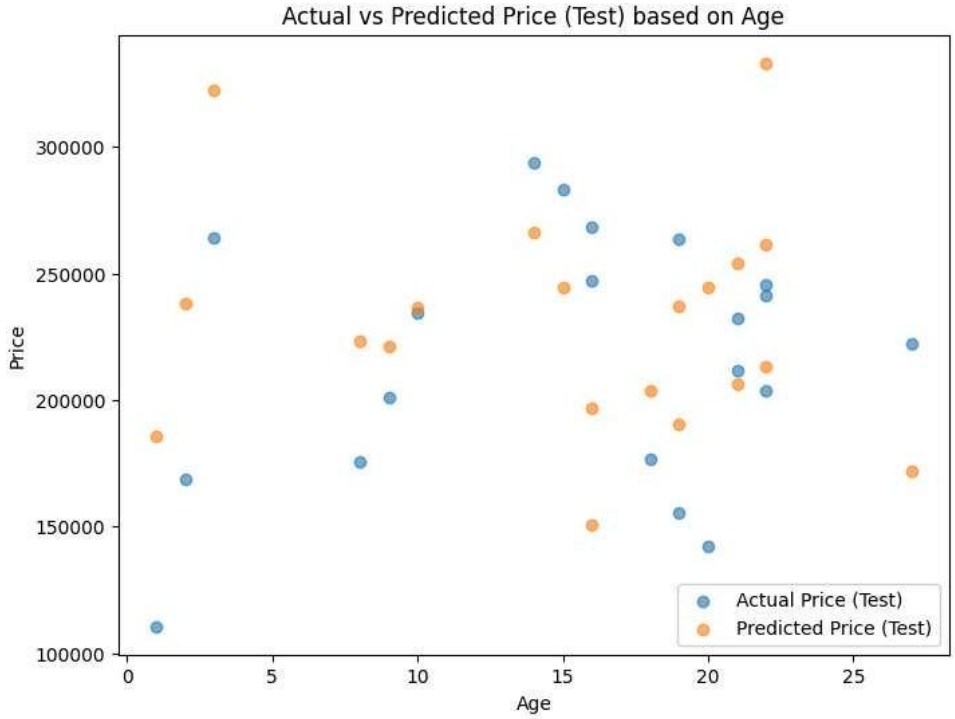
	0
0	261432.22
1	223822.71
2	270292.45
3	315199.58
4	221085.72
...	...
95	154172.81
96	247007.97
97	247295.02
98	233682.27
99	221276.06

100 rows × 1 columns

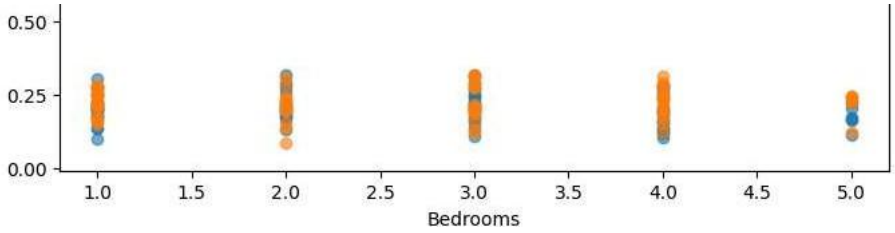
dtype: float64

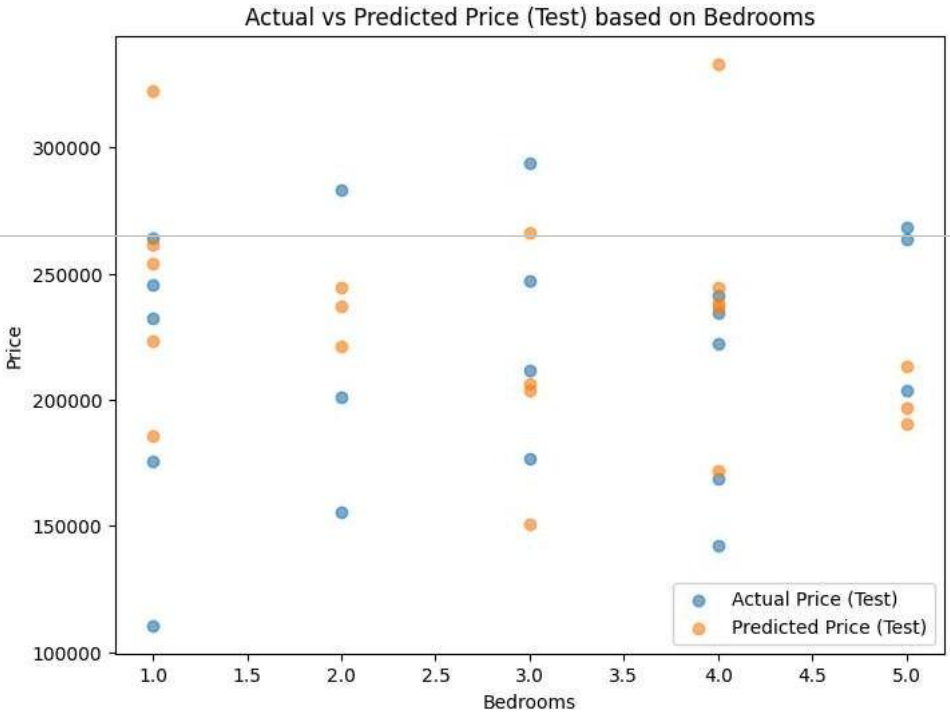
```
# Get independent variables
indep_variables = x.columns

# Plot training and testing data on different graphs
for col in indep_variables:
    plt.figure(figsize=(8, 6))
    # Plot actual vs predicted for training set
    plt.scatter(x_train[col], y_train, label='Actual Price (Train)', alpha=0.6)
    plt.scatter(x_train[col], regressor.predict(x_train), label='Predicted Price (Train)', alpha=0.6)
    plt.xlabel(col)
    plt.ylabel('Price')
    plt.title(f'Actual vs Predicted Price (Train) based on {col}')
    plt.legend()
    plt.show()

    plt.figure(figsize=(8, 6))
    # Plot actual vs predicted for testing set
    plt.scatter(x_test[col], y_test, label='Actual Price (Test)', alpha=0.6)
    plt.scatter(x_test[col], regressor.predict(x_test), label='Predicted Price (Test)', alpha=0.6)
    plt.xlabel(col)
    plt.ylabel('Price')
    plt.title(f'Actual vs Predicted Price (Test) based on {col}')
    plt.legend()
    plt.show()
```

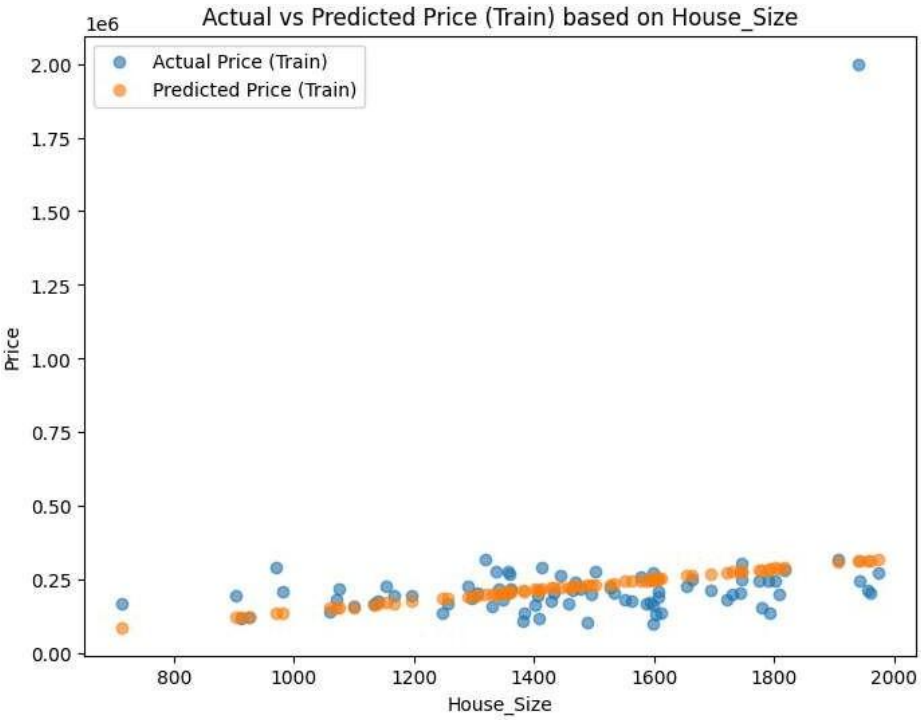

Actual vs Predicted Price (Train) based on Age
1e6
Actual Price (Train)
Predicted Price (Train)
Price
Age

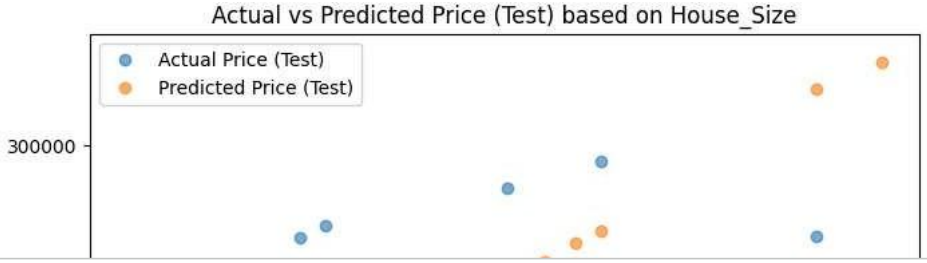

Actual vs Predicted Price (Test) based on Age
Actual Price (Test)
Predicted Price (Test)
Price
Age


Actual vs Predicted Price (Train) based on Bedrooms
1e6
Actual Price (Train)
Predicted Price (Train)
Price


0.50
0.25
0.00
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Bedrooms


Actual vs Predicted Price (Test) based on Bedrooms
300000
250000
200000
150000
100000
Price
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Bedrooms
Actual Price (Test)
Predicted Price (Test)


Actual vs Predicted Price (Train) based on House_Size
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Price
800
1000
1200
1400
1600
1800
2000
House_Size
Actual Price (Train)
Predicted Price (Train)


Actual vs Predicted Price (Test) based on House_Size
Actual Price (Test)
Predicted Price (Test)
300000

In [ ]:
```
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```


In [ ]:
```
student_scores_multi = pd.read_
```

In [ ]:
```
student_scores_multi.head()
```

Out[ ]:

	Hours_Studied	Attendance_Rate	Assignments_Completed	Exam_Score
0	7	66	8	76.46
1	4	80	5	75.50
2	8	68	9	91.11
3	5	98	9	89.95
4	7	77	6	91.40

In [ ]:
```
student_scores_multi
```

Out[ ]:

	Hours_Studied	Attendance_Rate	Assignments_Completed	Exam_Score
0	7	66	8	76.46
1	4	80	5	75.50
2	8	68	9	91.11
3	5	98	9	89.95
4	7	77	6	91.40
5	3	63	9	60.01
6	7	84	6	93.17
7	8	73	5	87.98
8	5	68	8	69.81
9	4	85	8	78.12
10	8	61	8	91.36
11	8	79	9	101.48
12	3	87	5	75.98
13	6	66	9	82.69
14	5	67	9	93.36
15	2	94	5	62.40
16	8	73	5	84.51
17	6	76	5	77.70
18	2	95	5	60.41
19	5	99	8	95.71
20	1	63	7	55.02
21	6	61	7	74.60
22	9	65	5	84.83
23	1	63	7	43.02
24	3	88	7	69.05
25	7	77	5	87.22
26	4	85	7	75.44
27	9	93	9	107.37
28	3	69	6	64.01
29	5	95	6	90.29

In [ ]:
```
# Split the complete attributes into "Input" and "Output" attributes

x  =  student_scores_multi[['Hours_Studied','Attendance_Rate','Assignments_Compl
y  =  student_scores_multi['Exam_Score']
```

In [ ]:
```
from sklearn.model_selection import train_test_split #splitting of data
```

In [ ]:
```
x_train , x_test , y_train , y_test = train_test_split(x,y, test_size = 0.2 ,
```

In [ ]:
```
x_train # to view the train , test records
```

Out[ ]:

	Hours_Studied	Attendance_Rate	Assignments_Completed
26	4	85	7
16	8	73	5
25	7	77	5
28	3	69	6
10	8	61	8
3	5	98	9
1	4	80	5
19	5	99	8
22	9	65	5
12	3	87	5
5	3	63	9
14	5	67	9
0	7	66	8
21	6	61	7
4	7	77	6
8	5	68	8
13	6	66	9
9	4	85	8
15	2	94	5
29	5	95	6
23	1	63	7
6	7	84	6
17	6	76	5
11	8	79	9

In [ ]:
```
x_test
```

Out[ ]:

	Hours_Studied	Attendance_Rate	Assignments_Completed
20	1	63	7
24	3	88	7
7	8	73	5
18	2	95	5
2	8	68	9
27	9	93	9

In [ ]:
```
y_train
```

Out[ ]:

	Exam_Score
26	75.44
16	84.51
25	87.22
28	64.01
10	91.36
3	89.95
1	75.50
19	95.71
22	84.83
12	75.98
5	60.01
14	93.36
0	76.46
21	74.60
4	91.40
8	69.81
13	82.69
9	78.12
15	62.40
29	90.29
23	43.02
6	93.17
17	77.70
11	101.48

dtype: float64

In [ ]:
```
y_test
```

Out[ ]:

	Exam_Score
20	55.02
24	69.05
7	87.98
18	60.41
2	91.11
27	107.37

dtype: float64

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

In [ ]:
```
lm = LinearRegression()
```

lm.fit(x_train, y_train) is used to train the linear regression model initialized in the previous step.

- lm: This is the LinearRegression model object.
- .fit(): This is the method used to train the model.

In [ ]:
```
lm.fit(x_train,y_train)
```

Out[ ]:

LinearRegression

LinearRegression()

In [ ]:
```
# prediction = lm.predict([[4,85,7]])
prediction = lm.predict(x_train)

prediction
```

Out[ ]:
```
array([ 77.91461739,  88.58927156,  85.45191123,  60.4984674 ,
        87.76288977,  95.66173388,  70.58198637,  94.1179132 ,
        89.30092353,  69.26390731,  63.31064818,  76.86249398,
        85.23195653,  74.48650457,  87.602159  ,  75.31867329,
        81.81913559,  80.06486517,  67.94582826,  87.39170927,
        47.8840152 ,  91.84714866,  79.28241542, 100.82882524])
```

In [ ]:
```
prediction = lm.predict(x_test)

prediction
```

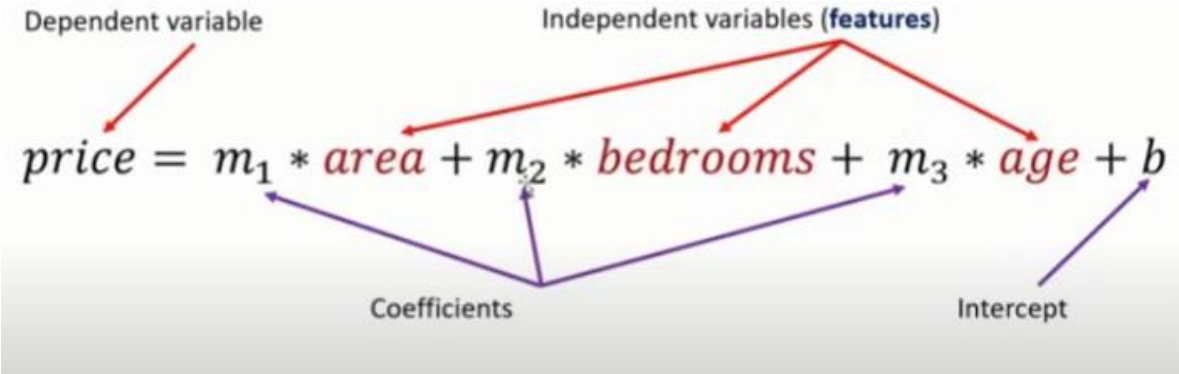
Out[ ]:
```
array([ 47.8840152 ,  74.17082996,  88.58927156,  68.55225535,
        94.15812721, 114.88187326])
```

In [ ]:
```
y_test
```

Out[ ]:

	Exam_Score
20	55.02
24	69.05
7	87.98
18	60.41
2	91.11
27	107.37

dtype: float64



In [ ]:
```
print(lm.intercept_)

#displays the y-intercept of the linear regression model (lm)
```

-10.93569486218594

In [ ]:
```
print(lm.coef_)
```

[5.56306871 0.60642709 2.15024778]

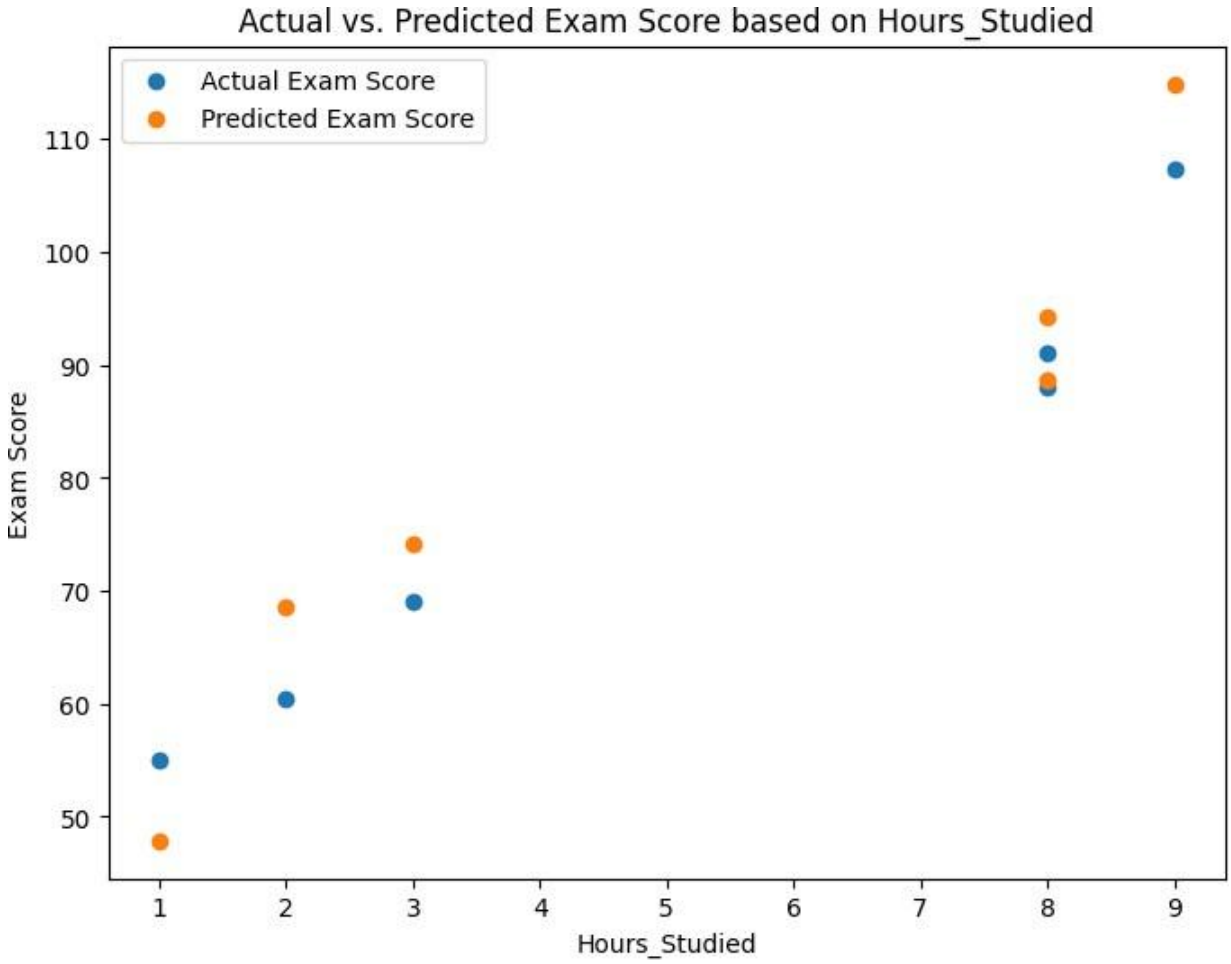
In [ ]:
```
yy = lm.intercept_ + lm.coef_ * x  # fromula : Y = Mx + c
round(yy,0)
```

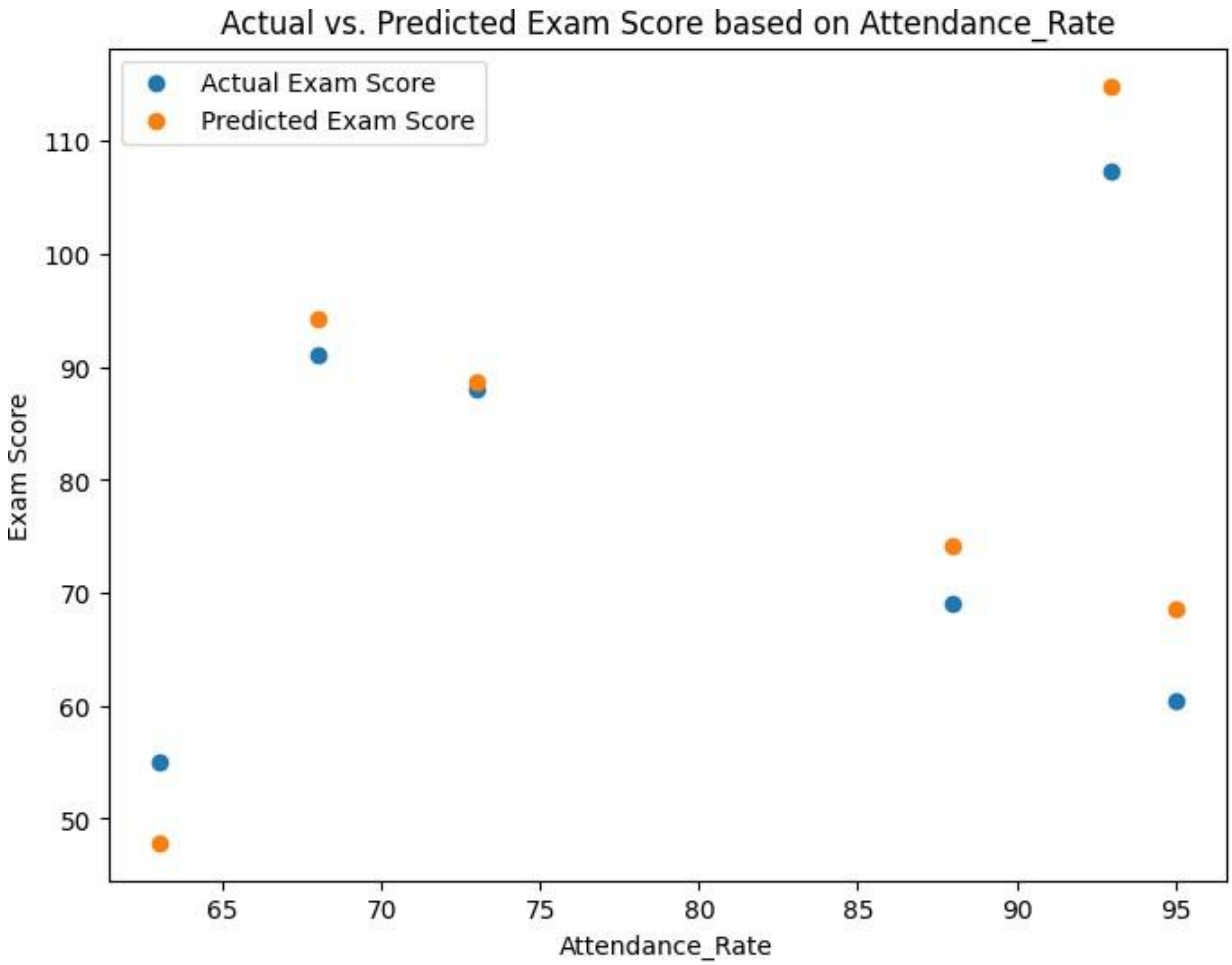
Out[ ]:

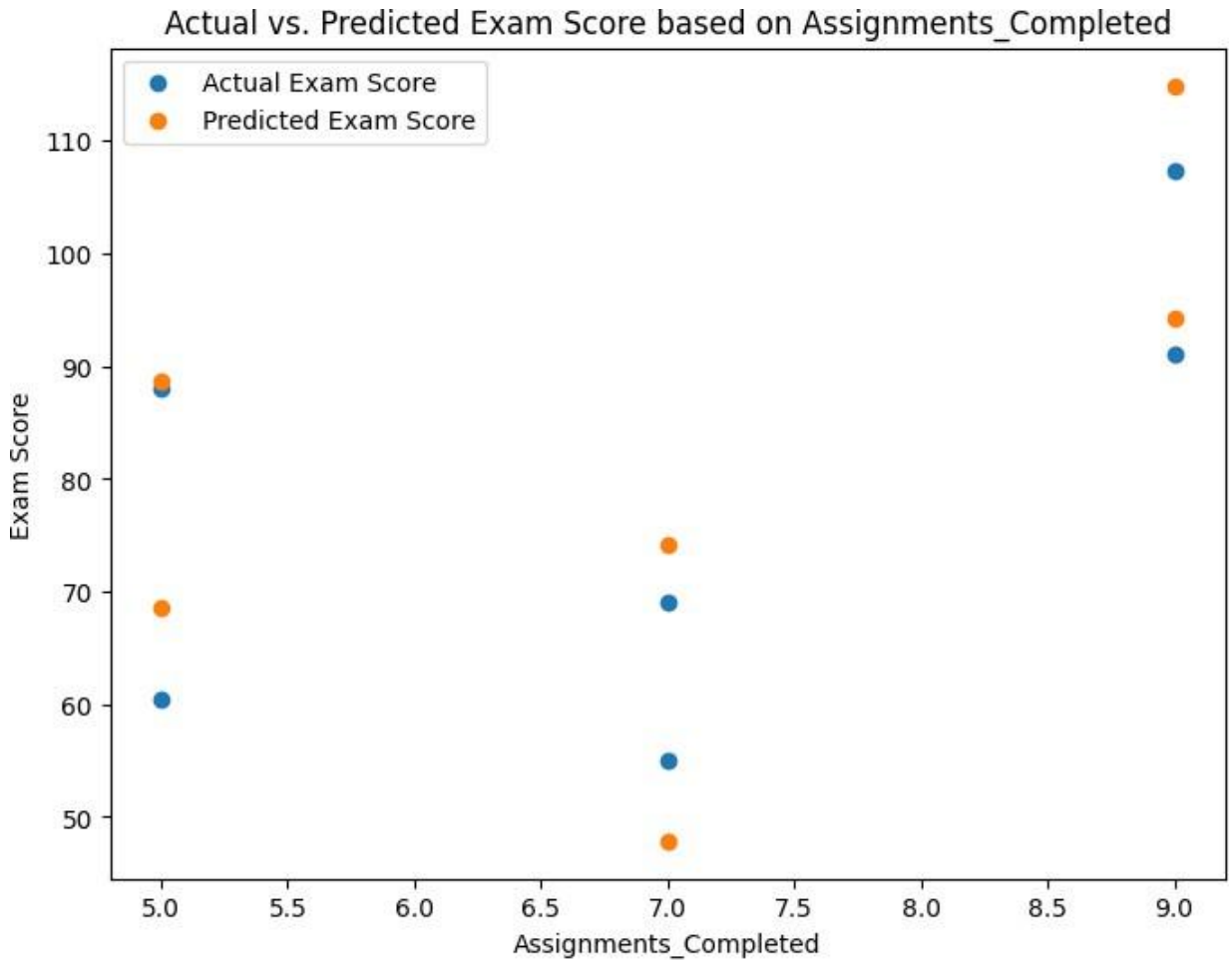
	Hours_Studied	Attendance_Rate	Assignments_Completed
0	28.0	29.0	6.0
1	11.0	38.0	-0.0
2	34.0	30.0	8.0
3	17.0	48.0	8.0
4	28.0	36.0	2.0
5	6.0	27.0	8.0
6	28.0	40.0	2.0
7	34.0	33.0	-0.0
8	17.0	30.0	6.0
9	11.0	41.0	6.0
10	34.0	26.0	6.0
11	34.0	37.0	8.0
12	6.0	42.0	-0.0
13	22.0	29.0	8.0
14	17.0	30.0	8.0
15	0.0	46.0	-0.0
16	34.0	33.0	-0.0
17	22.0	35.0	-0.0
18	0.0	47.0	-0.0
19	17.0	49.0	6.0
20	-5.0	27.0	4.0
21	22.0	26.0	4.0
22	39.0	28.0	-0.0
23	-5.0	27.0	4.0
24	6.0	42.0	4.0
25	28.0	36.0	-0.0
26	11.0	41.0	4.0
27	39.0	45.0	8.0
28	6.0	31.0	2.0
29	17.0	47.0	2.0

In [ ]:

```
# Get the names of the independent variables
independent_variables = x.columns

# Plot each independent variable against the actual and predicted exam scores
for col in independent_variables:
    plt.figure(figsize=(8, 6)) # Create a new figure for each plot
    plt.scatter(x_test[col], y_test, label='Actual Exam Score')
    plt.scatter(x_test[col], prediction, label='Predicted Exam Score')
    plt.xlabel(col)
    plt.ylabel('Exam Score')
    plt.title(f'Actual vs. Predicted Exam Score based on {col}')
    plt.legend()
    plt.show()
```




Actual vs. Predicted Exam Score based on Attendance_Rate
Actual Exam Score
Predicted Exam Score
Exam Score
Attendance_Rate
110
100
90
80
70
60
50
65
70
75
80
85
90
95



In [ ]:
```
print(y_test)
print(prediction)
```

```
20      55.02
24      69.05
7       87.98
18      60.41
2       91.11
27     107.37
Name: Exam_Score, dtype: float64
[ 47.8840152   74.17082996 88.58927156 68.55225535 94.15812721
 114.88187326]
```

In [ ]:
```
# Metrics
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
mae = mean_absolute_error(y_test,prediction)
mse = mean_squared_error(y_test,prediction)
r2 = r2_score(y_test,prediction)
```

In [ ]:
```
print(mae, mse, r2)
```

```
5.261390358336993 34.92200533660641 0.8978002104849254
```

In [ ]:
```
y_test = [55,69,87]
```

```
prediction = [47 ,74 , 88]
mean_absolute_error(y_test,prediction)
```

Out[ ]: 4.666666666666667

In [ ]:
```
y_test.describe()
```

Out[ ]:

	Exam_Score
count	6.000000
mean	78.490000
std	20.249542
min	55.020000
25%	62.570000
50%	78.515000
75%	90.327500
max	107.370000

dtype: float64

In [ ]:
```

```

```
# import libraries

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

```
student_dirty = pd.read_csv('/content/student_scores_dirty - student_scores_dirty.csv')
```

```
student_dirty.head()
```

	Student_ID	Hours_Studied	Attendance_Rate	Assignments_Completed	Exam_Score
0	1	1.0	85%	8.0	35.0
1	2	2.0	90%	NaN	45.0
2	3	NaN	NaN	7.0	50.0
3	4	4.0	75 %	5.0	NaN
4	5	200.0	80%	9.0	65.0

```
student_dirty
```

	Student_ID	Hours_Studied	Attendance_Rate	Assignments_Completed	Exam_Score
0	1	1.0	85%	8.0	35.0
1	2	2.0	90%	NaN	45.0
2	3	NaN	NaN	7.0	50.0
3	4	4.0	75 %	5.0	NaN
4	5	200.0	80%	9.0	65.0
5	6	6.0	NaN	8.0	70.0
6	7	NaN	88%	NaN	75.0
7	8	8.0	95%	10.0	NaN
8	9	9.0	65%	4.0	90.0
9	10	10.0	70%	3.0	95.0

```
std_hr_median = student_dirty.Hours_Studied.median()
```

- student_dirty['Attendance_Rate'] = student_dirty['Attendance_Rate'].str.replace('%', '').str.strip(): This line cleans the 'Attendance_Rate' column. .str.replace('%', '') removes the '%' symbol from each string in the column. .str.strip() removes any leading or trailing whitespace from each string.
- student_dirty['Attendance_Rate'] = pd.to_numeric(student_dirty['Attendance_Rate']): This line converts the cleaned 'Attendance_Rate' column to a numeric data type. This is necessary for mathematical operations like calculating the mean.
- print(student_dirty.Attendance_Rate.mean()): This line calculates the mean (average) of the now numeric 'Attendance_Rate' column and prints it to the console.

```
student_dirty['Attendance_Rate'] = student_dirty['Attendance_Rate'].str.replace('%', '').str.strip()
student_dirty['Attendance_Rate'] = pd.to_numeric(student_dirty['Attendance_Rate'])
print(student_dirty.Attendance_Rate.mean())
```

```
81.0
```

```
std_rate_mean = student_dirty.Attendance_Rate.mean()
```

```
Start coding or generate with AI.
```

```
std_assign_comp_median = student_dirty.Assignments_Completed.median()
```

```
import math

mean_Exam = math.floor(student_dirty.Exam_Score.mean())   # import math for taking value  in numbers
print(mean_Exam)
```

```
65
```

fillna function

```
student_dirty.Hours_Studied = student_dirty.Hours_Studied.fillna(std_hr_median)
student_dirty.Attendance_Rate = student_dirty.Attendance_Rate.fillna(std_rate_mean)
student_dirty.Assignments_Completed = student_dirty.Assignments_Completed.fillna(std_assign_comp_median)
student_dirty.Exam_Score = student_dirty.Exam_Score.fillna(mean_Exam)
```

```
student_dirty
```

	Student_ID	Hours_Studied	Attendance_Rate	Assignments_Completed	Exam_Score
0	1	1.0	85.0	8.0	35.0
1	2	2.0	90.0	7.5	45.0
2	3	7.0	81.0	7.0	50.0
3	4	4.0	75.0	5.0	65.0
4	5	200.0	80.0	9.0	65.0
5	6	6.0	81.0	8.0	70.0
6	7	7.0	88.0	7.5	75.0
7	8	8.0	95.0	10.0	65.0
8	9	9.0	65.0	4.0	90.0
9	10	10.0	70.0	3.0	95.0

```
# split the dataset

x = student_dirty[['Attendance_Rate', 'Hours_Studied', 'Assignments_Completed']]
y = student_dirty[['Exam_Score']]
```

```
from sklearn.model_selection import train_test_split
x_train , x_test , y_train , y_test = train_test_split(x,y,test_size = 0.2,random_state = 101)
```

```
x_train
```

	Attendance_Rate	Hours_Studied	Assignments_Completed
0	85.0	1.0	8.0
4	80.0	200.0	9.0
9	70.0	10.0	3.0
3	75.0	4.0	5.0
5	81.0	6.0	8.0
7	95.0	8.0	10.0
6	88.0	7.0	7.5
1	90.0	2.0	7.5

```
x_test
```

	Attendance_Rate	Hours_Studied	Assignments_Completed
8	65.0	9.0	4.0
2	81.0	7.0	7.0

```
y_train
```

	Exam_Score
0	35.0
4	65.0
9	95.0
3	65.0
5	70.0
7	65.0
6	75.0
1	45.0

```
y_test
```

	Exam_Score
8	90.0
2	50.0

```
from sklearn.linear_model import LinearRegression
```

```
reg  = LinearRegression()
```

```
reg.fit(x_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
prediction_y_train = reg.predict(x_train)
prediction_y_train
```

```
array([[59.20528794],
       [66.5067318 ],
       [83.60679536],
       [73.82534557],
       [59.94139334],
       [49.58420189],
       [61.41931045],
       [60.91093366]])
```

```
prediction_y_test = reg.predict(x_test)
prediction_y_test
```

```
array([[79.68231203],
       [64.4351982 ]])
```

```
y_test
```

	Exam_Score
8	90.0
2	50.0

```
print(reg.intercept_)
```

```
[104.32902604]
```

```
reg.coef_
```

```
array([[-0.11386431,  0.05612964, -4.43767522]])
```

```
reg.predict([[100,  5,  8]])
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but
  warnings.warn(
array([[57.7218419]])
```

```
-0.11386431*100 +  0.05612964*5  -4.43767522*8 + 104.32
```

```
57.71281543999999
```

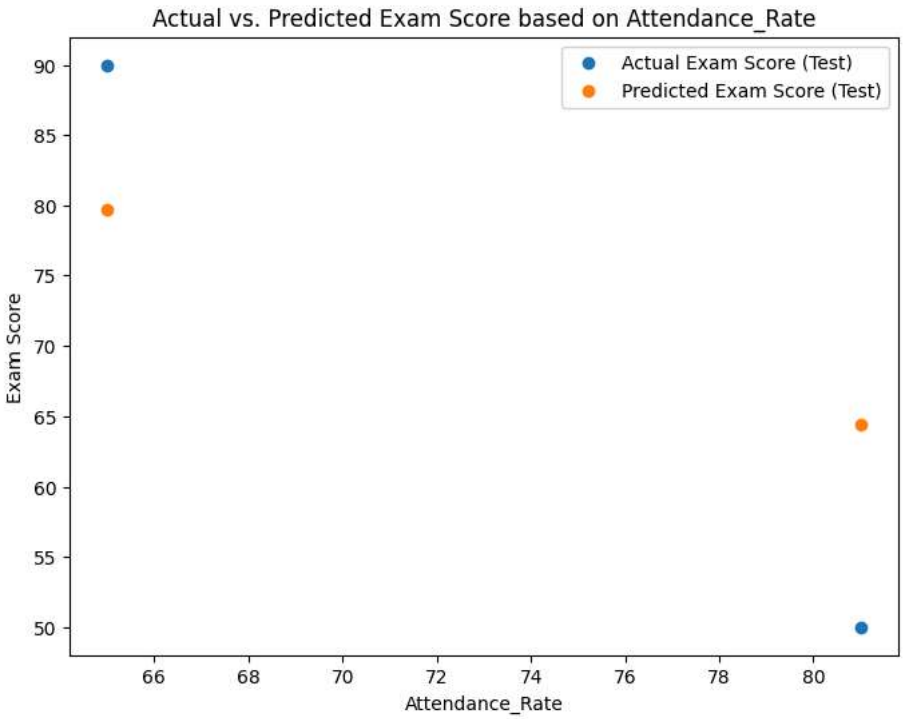
- yy = reg.intercept_ + x @ reg.coef_.T:
- reg.intercept_: This is the y-intercept of the linear regression model. It's the predicted value of the exam score when all other features (Attendance Rate, Hours Studied, and Assignments Completed) are zero.
- x: This is the DataFrame containing the features (Attendance Rate, Hours Studied, and Assignments Completed) for which you want to make predictions.
- reg.coef_.T: This is the transpose of the coefficients of the linear regression model. The coefficients represent the change in the predicted exam score for a one-unit increase in each respective feature, holding other features constant. **We use the transpose to align the dimensions for matrix multiplication.**
- @: This symbol represents matrix multiplication. The features (x) are multiplied by the transposed coefficients (reg.coef_.T). reg.intercept_ + ...: The intercept is added to the result of the matrix multiplication. This entire calculation follows the linear regression equation: y = intercept + (coefficient_1 * feature_1) + (coefficient_2 * feature_2) + ...
- yy = ...: The calculated predicted values are stored in the variable yy.
- round(yy,0): This rounds the predicted values in the yy DataFrame to 0 decimal places. This is often done to make the predictions easier to interpret, especially if the exam scores are expected to be whole numbers.

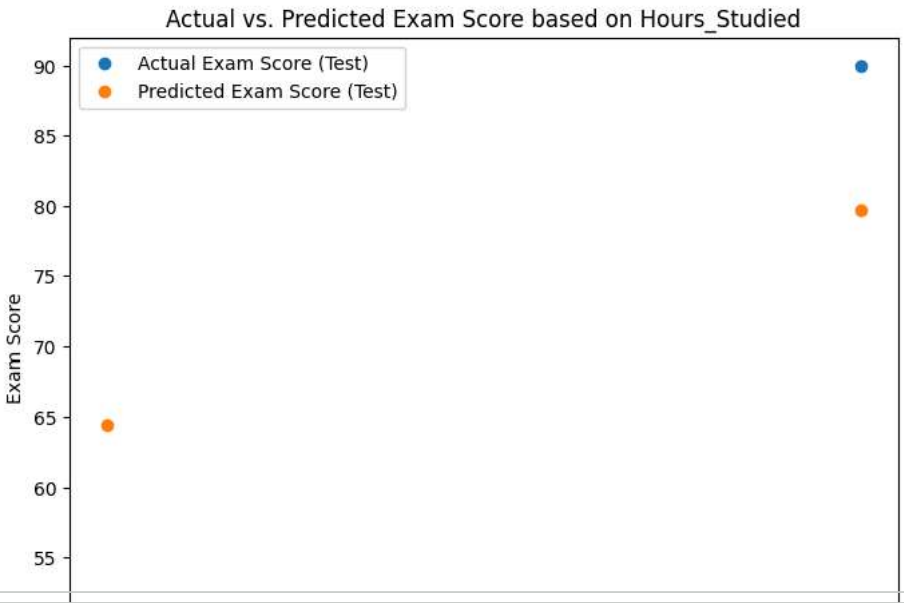
```
yy = reg.intercept_ + x @ reg.coef_.T
round(yy,0)
```

	0
0	59.0
1	61.0
2	64.0
3	74.0
4	67.0
5	60.0
6	61.0
7	50.0
8	80.0
9	84.0

```
# Get the names of the independent variables
indep_variables = x.columns

# plot each independent variable against the actual and predicted exam score

for col in indep_variables:
    plt.figure(figsize=(8,6)) # Create a new figure for each plot

    # Plot actual vs. predicted for the training set
    #plt.scatter(x_train[col], y_train, label='Actual Exam Score (Train)')
    #plt.scatter(x_train[col], reg.predict(x_train), label='Predicted Exam Score (Train)') # Use predictions for train set

    # Plot actual vs. predicted for the testing set
    plt.scatter(x_test[col], y_test, label='Actual Exam Score (Test)')
```

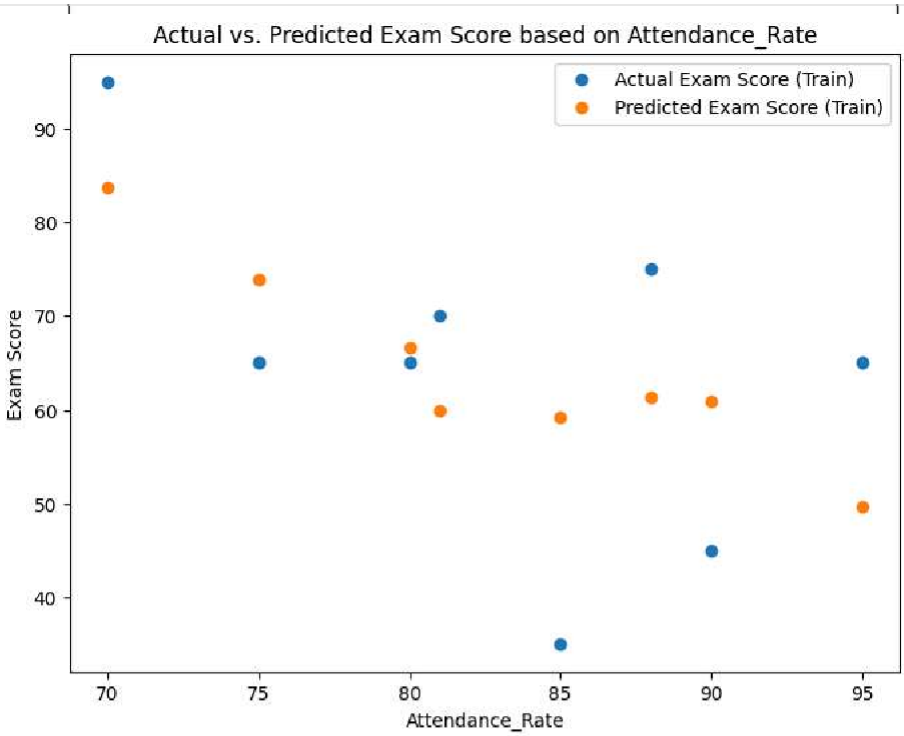
```
    plt.scatter(x_test[col], reg.predict(x_test), label='Predicted Exam Score (Test)') # Use predictions for test set


    plt.xlabel(col)
    plt.ylabel('Exam Score')
    plt.title(f'Actual vs. Predicted Exam Score based on {col}')
    plt.legend()
    plt.show()
```

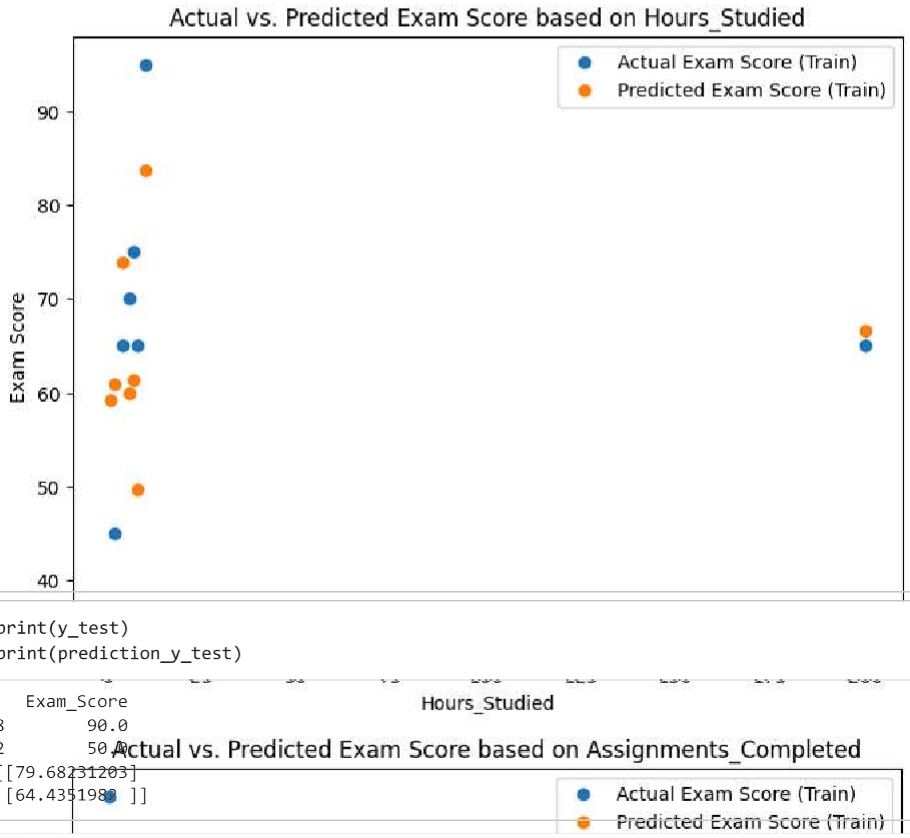





```
# Get the names of the independent variables
indep_variables = x.columns

# plot each independent variable against the actual and predicted exam score

for col in indep_variables:
    plt.figure(figsize=(8,6)) # Create a new figure for each plot

    # Plot actual vs. predicted for the training set
    plt.scatter(x_train[col], y_train, label='Actual Exam Score (Train)')
    plt.scatter(x_train[col], reg.predict(x_train), label='Predicted Exam Score (Train)') # Use predictions for train set

    # Plot actual vs. predicted for the testing set
    #plt.scatter(x_test[col], y_test, label='Actual Exam Score (Test)')
    #plt.scatter(x_test[col], reg.predict(x_test), label='Predicted Exam Score (Test)') # Use predictions for test set
```

```
    plt.xlabel(col)
    plt.ylabel('Exam Score')
    plt.title(f'Actual vs. Predicted Exam Score based on {col}')
    plt.legend()
    plt.show()
```






```
print(y_test)
print(prediction_y_test)
```

```
   Exam_Score
8        90.0
2        50.0
[[79.68231203]
 [64.43519882]]
```

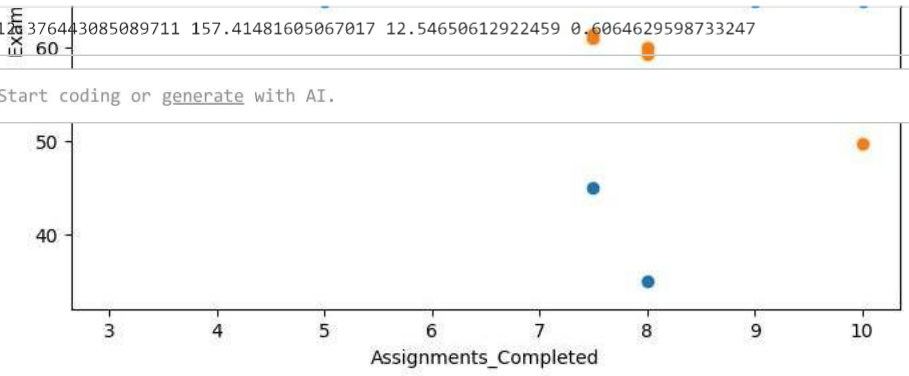


```
# Metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
mae = mean_absolute_error(y_test , prediction_y_test)
mse = mean_squared_error(y_test , prediction_y_test)
rmse = np.sqrt(mse)
r2 = r2_score(y_test , prediction_y_test)
```

```
print(mae,mse,rmse,r2)
```

```
1.1337644308508971 157.41481605067017 12.54650612922459 0.6064629598733247
```

Start coding or generate with AI.



Simple Linear Regression

```
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
student_hrs_study_exam_dataset = pd.read_csv('/content/student_scores - student_scores.csv')
```

```
student_hrs_study_exam_dataset  # to view uploaded dataset
```

	Hours_Studied	Exam_Score
0	1	35
1	2	45
2	3	50
3	4	55
4	5	65
5	6	70
6	7	75
7	8	85
8	9	90
9	10	95

```
# Split the complete attributes into "Input" and "Output" attributes

x = student_hrs_study_exam_dataset[['Hours_Studied']]  # Input
y = student_hrs_study_exam_dataset[['Exam_Score']]     # Output
```

```
from sklearn.model_selection import train_test_split  #Split the records for training and testing
```

```
x_train , x_test , y_train , y_test = train_test_split(x,y, test_size = 0.2 , random_state = 101)
```

```
x_train # to view the train , test records
```

	Hours_Studied
0	1
4	5
9	10
3	4
5	6
7	8
6	7
1	2

```
x_test
```

	Hours_Studied
8	9
2	3

```
y_train
```

	Exam_Score
0	35
4	65
9	95
3	55
5	70
7	85
6	75
1	45

```
y_test
```

	Exam_Score
8	90
2	50

Apply Linear regression

```
from sklearn.linear_model import LinearRegression
```

```
lm = LinearRegression()
```

```
lm.fit(x_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
predictions = lm.predict(x_test)
```

```
predictions
```

```
array([[89.63796477],
       [49.89236791]])
```

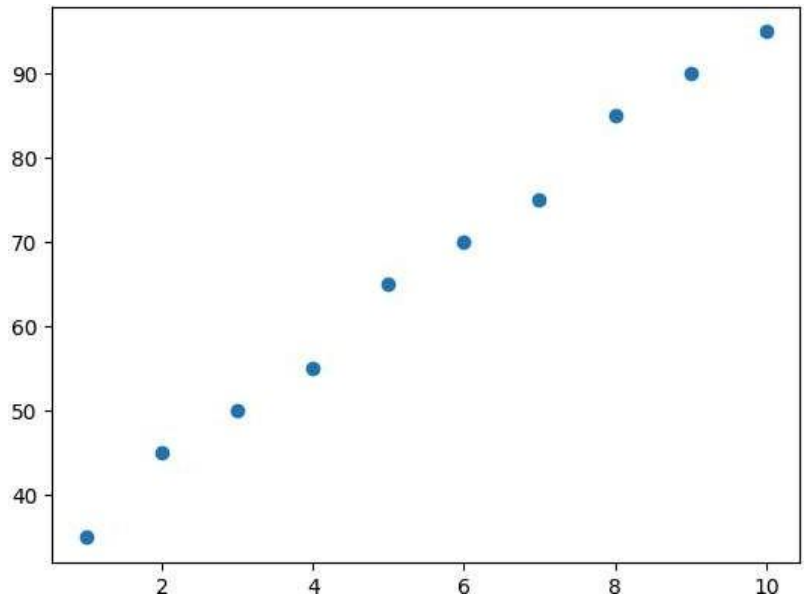
```
y_test
```

	Exam_Score
8	90
2	50

plot the Scatter plot

```
plt.scatter(x,y)
```

```
<matplotlib.collections.PathCollection at 0x7bf9735d7290>
```




```
print(lm.intercept_)

#displays the y-intercept of the linear regression model (lm)
```

```
[30.01956947]
```

```
print(lm.coef_)
#displays the coefficient of the independent variable (Hours_Studied) in your linear regression model (lm)
# In simple linear regression, this coefficient represents the slope of the regression line
```

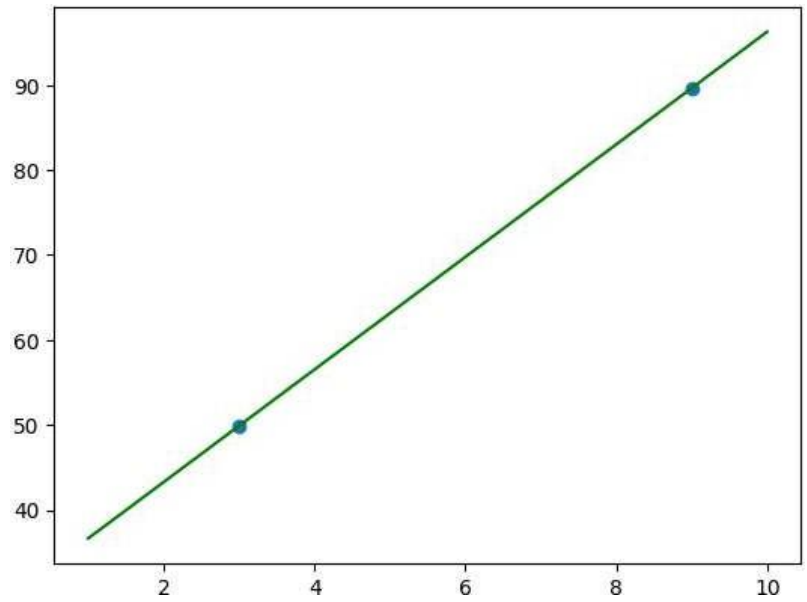
```
[[6.62426614]]
```

```
yy = lm.intercept_ + lm.coef_ * x  # fromula : Y = Mx + c
round(yy,0)


# This argument in the round() function specifies the number of decimal places to which you want to round the v
```

	Hours_Studied
0	37.0
1	43.0
2	50.0
3	57.0
4	63.0
5	70.0
6	76.0
7	83.0
8	90.0
9	96.0

```
plt.plot(x,yy,'g')
#plt.scatter(X,Y)
plt.scatter(x_test,predictions)
```

```
<matplotlib.collections.PathCollection at 0x7bf973642f10>
```




```
print(y_test)
print(predictions)
```

```
   Exam_Score
8          90
2          50
[[89.63796477]
 [49.89236791]]
```

```
# Metrics
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
mae = mean_absolute_error(y_test,predictions)
mse = mean_squared_error(y_test,predictions)
r2 = r2_score(y_test,predictions)
```

```
print(mae, mse, r2)
```

```
0.23483365949119417 0.07132708591036088 0.9998216822852241
```

```
y_true = [50,90]
y_pred = [49,89]

mean_absolute_error(y_true,y_pred)
```

```
1.0
```

```
#%%shell
#jupyter nbconvert --to html /content/LinearRegression.ipynb


#nbconvert to convert the current Jupyter notebook (/content/LinearRegression.ipynb) to an HTML file.
```

Start coding or generate with AI.

MLA_Lab_2_k_fold_validation

1. Import Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

2. Load Datasets

```
df = pd.read_csv('/content/processes2.csv')
df.head()
```

	Unnamed: 0	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	seats	max_power (in bph)	Mileage Unit	Mileage
0	0	Maruti	2014	450000	145500	Diesel	Individual	Manual	First Owner	5	74.00	kmpl	23.40
1	2	Hyundai	2010	225000	127000	Diesel	Individual	Manual	First Owner	5	90.00	kmpl	23.00
2	4	Hyundai	2017	440000	45000	Petrol	Individual	Manual	First Owner	5	81.86	kmpl	20.14

3. Data Cleaning

```
# Example: Select needed columns and drop NA rows
df = df[['selling_price','km_driven','year','seats','Engine (CC)']]
df = df.dropna()
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2095 entries, 0 to 2094
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   selling_price  2095 non-null   int64
 1   km_driven      2095 non-null   int64
 2   year           2095 non-null   int64
 3   seats          2095 non-null   int64
 4   Engine (CC)    2095 non-null   int64
dtypes: int64(5)
memory usage: 82.0 KB
None
```

4. Define Features and Target

creates the feature matrix X and target vector y

```
x = df.drop('selling_price', axis=1)
y = df['selling_price']
```

5. Setup KFold cross validation

- prepares 5-fold cross-validation (shuffles data for randomness) and an empty list to save validation scores.

```
from sklearn.model_selection import KFold
```

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
mse_scores = []
```

6. Train and Validation with KFold

cross-validation:

- Splits data into train/test sets per fold

```
for train_index, test_index in kf.split(x):
  x_train, x_test = x.iloc[train_index], x.iloc[test_index]
  y_train, y_test = y.iloc[train_index], y.iloc[test_index]
```

- Trains the linear regression model
- Makes predictions on test set

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
```

- Computes and prints MSE for each fold, and saves it in mse_scores.

```
from sklearn.metrics import mean_squared_error

mse = mean_squared_error(y_test, y_pred)
mse_scores.append(mse)
print(f"MSE for this fold: {mse}")
```

```
MSE for this fold: 15730039441.473888
```

```
Start coding or generate with AI.
```

MLA_Assignment_2_Logistic_Regression

1. Import Libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report
```

2. Load Dataset

```
logi_data = pd.read_csv('/content/2a_logistic_regression_dirty.csv')
logi_data.head()
```

	Age	BMI	Blood_Pressure	Cholesterol	Diabetes
0	44.0	19.478891	95	250	0
1	54.0	28.807859	87	155	0
2	60.0	22.253709	83	175	1
3	49.0	12.704033	83	213	1
4	36.0	25.287065	135	208	1

3. Data Cleaning

```
# Replace 'high' in Cholesterol with NaN and convert to numeric

logi_data['Cholesterol'] = pd.to_numeric(logi_data['Cholesterol'].replace('high', np.nan))
```

```
# missing values with median (better than dropping data)

logi_data['Age'].fillna(logi_data['Age'].median(), inplace=True)
logi_data['BMI'].fillna(logi_data['BMI'].median(), inplace=True)
logi_data['Cholesterol'].fillna(logi_data['Cholesterol'].median(), inplace=True)
```

```
/tmp/ipython-input-523122053.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df


  logi_data['Age'].fillna(logi_data['Age'].median(), inplace=True)
/tmp/ipython-input-523122053.py:4: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df


  logi_data['BMI'].fillna(logi_data['BMI'].median(), inplace=True)
/tmp/ipython-input-523122053.py:5: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df


  logi_data['Cholesterol'].fillna(logi_data['Cholesterol'].median(), inplace=True)
```

4. Splitting Input , Output

```
# spliting
X = logi_data[['Age', 'BMI', 'Blood_Pressure', 'Cholesterol']].astype(float)
y = logi_data['Diabetes'].astype(int)
```

5. Training, Testing Data

- StandardScaler() : standardizes features by removing the mean and scaling to unit variance
- fit_transform(X): computes the mean and standard deviation on the dataset X and then scales X so that each feature has a mean of 0 and a standard deviation of 1.

```
# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)
```

6. Logistic Regression

```
# Logistic regression model
model = LogisticRegression(max_iter=500)
model.fit(X_train, y_train)
```

```
LogisticRegression(max_iter=500)
```

7. Prediction and Accuracy

```
# Prediction
y_pred = model.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.4f}")
print("Classification Report:\n", report)
```

```
Accuracy: 0.5000
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.73      0.59        15
           1       0.50      0.27      0.35        15

    accuracy                           0.50        30
   macro avg       0.50      0.50      0.47        30
weighted avg       0.50      0.50      0.47        30
```

```
Start coding or generate with AI.
```

Machine Learning Algorithm

Confusion Matrix

Mahavir
Page No.
Date :

Name; Sangram Lembe
Reg. NO; RA2512051010001

Q.①

Prediction	Actual Value: TP	Actual Value: FP
	(50)	0
	FN	TN
	(50)	(900)

Matrics calculation;

$$\text{Accuracy} = \frac{TP + TN}{\text{Total}} = \frac{950}{1000} = 95\%$$

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{50}{50 + 0} = 1 \; (100\%)$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{50}{50 + 50} = 0.5 \; (50\%)$$

$$F_1 \text{ score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$= \frac{2 \times 1 \times 0.5}{1 + 0.5} = 0.6667 \; (66.67\%)$$

Why accuracy is misleading:

① Accuracy reflects the proportions of correct predic-tions, but in an imbalanced dataset, a model can achieve high accuracy simply by predic-ting the majority class (Non-cancer), while missing many actual cancer cases.

② This approch gives false sense of perfor-mance as it ignores minority class performance, which is critical in medical diagnosis.

Better metric suggestions

① F_1 score is hermonic mean of precision and recall, giving balanced view for both classes.

② Matthew's corelation coefficient MCC is also recomended, as it encorporates all four confusion matrix categories - TP, TN, FP, FN

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

Mahavir
Page No.
Date :

(2)

Actual ⟶

Predicted ↓		Positive	Negative
	positive	50	10
	Negative	10	30

$$\text{Accuracy} = \frac{TP + TN}{\text{Total}} = \frac{50 + 30}{100} = 0.8 \ (80\%)$$

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{50}{50 + 10} = 0.83 \ (83\%)$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{50}{60} = 0.83 \ (83\%)$$

$$F_1\text{-score} = \frac{2 \times \text{precision} \times \text{Recall}}{\text{precision} + \text{Recall}}$$

$$= \frac{2 \times 0.83 \times 0.83}{0.833 + 0.833}$$

$$= \frac{2 \times 0.694}{1.666} \approx 0.833 \ (85\%)$$

KNN_assignment

Step 1: Data Loading and Preprocessing

Import Libraries

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
import numpy as np
```

Load the Data

```
# Load your CSV file
file_path = '/content/loan_default.csv'
df = pd.read_csv(file_path)
print(df.head())  # Check the first few rows
print(df.info())  # Get a summary of the data
```

```
  Income  LoanAmount  CreditScore  Defaulted
0  11472     30483.0        420.0        0.0
1   5727         NaN        751.0        1.0
2  15732     23763.0        428.0        1.0
3   6958         NaN        390.0        1.0
4   2524     16733.0        328.0        1.0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Income       196 non-null    object
 1   LoanAmount   194 non-null    float64
 2   CreditScore  194 non-null    float64
 3   Defaulted    194 non-null    float64
dtypes: float64(3), object(1)
memory usage: 6.5+ KB
None
```

Data Cleaning

- SimpleImputer: This is a tool from the scikit-learn library designed specifically for handling missing values.
- Any remaining missing values(NaN) are filled using the median of their respective columns. The median is a good choice as it is robust to outliers.

```
# Replace 'error' with NaN
df.replace('error', np.nan, inplace=True)

# Handle missing values using Median Imputation
imputer = SimpleImputer(strategy='median')
df_imputed = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)

# .fit() : learns what value to use for each column's missing cells.
# .transform(): In DataFrame, it replaces all the missing values (like NaN or empty cells) with the medians
# .fit_transform : returns the data as a NumPy array
# pd.DataFrame(...) to convert it back into a pandas DataFrame.

print("\nDataFrame info after imputation:")
df_imputed.info()
print("\nDataFrame head after imputation:")
```

```
print(df_imputed.head())
```

```
DataFrame info after imputation:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Income       205 non-null    float64
 1   LoanAmount   205 non-null    float64
 2   CreditScore  205 non-null    float64
 3   Defaulted    205 non-null    float64
dtypes: float64(4)
memory usage: 6.5 KB

DataFrame head after imputation:
    Income  LoanAmount  CreditScore  Defaulted
0  11472.0     30483.0        420.0        0.0
1   5727.0     22971.5        751.0        1.0
2  15732.0     23763.0        428.0        1.0
3   6958.0     22971.5        390.0        1.0
4   2524.0     16733.0        328.0        1.0
```

Step 2: Data Splitting

- The dataset is split into features (X) and the target variable (y). In this case, Defaulted is the target you want to predict.
- The data is then divided into a training set (80%) and a testing set (20%). The model will learn from the training set and be evaluated on the unseen testing set.
- Feature scaling is applied using StandardScaler to standardize the feature values. This is important for both Logistic Regression and especially for KNN, which is a distance-based algorithm

```
X = df_imputed[['Income', 'LoanAmount', 'CreditScore']]
y = df_imputed['Defaulted']
```

Train Test Split

```
from sklearn.model_selection import train_test_split

# Add stratify=y
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
X_train shape: (164, 3)
X_test shape: (41, 3)
y_train shape: (164,)
y_test shape: (41,)
```

Scale the features

```
# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Step 3: Model Training and Prediction

Logistic Regression

```
log_reg = LogisticRegression(random_state=42, class_weight='balanced')
log_reg.fit(X_train_scaled, y_train)
y_pred_log_reg = log_reg.predict(X_test_scaled)
```

K-Nearest Neighbors (KNN)

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)
y_pred_knn = knn.predict(X_test_scaled)
```

Step 4: Model Evaluation and Comparison

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

```
# --- Logistic Regression Evaluation ---
print("\n--- Logistic Regression Metrics ---")
accuracy_log_reg = accuracy_score(y_test, y_pred_log_reg)
precision_log_reg = precision_score(y_test, y_pred_log_reg, average='weighted')  # Our target variable Defaulted is not strictl
recall_log_reg = recall_score(y_test, y_pred_log_reg, average='weighted')

print(f"Accuracy: {accuracy_log_reg:.4f}")
print(f"Precision: {precision_log_reg:.4f}")
print(f"Recall: {recall_log_reg:.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_log_reg, zero_division=0))


# --- KNN Evaluation ---
print("\n--- KNN Metrics ---")
accuracy_knn = accuracy_score(y_test, y_pred_knn)
precision_knn = precision_score(y_test, y_pred_knn, average='weighted')
recall_knn = recall_score(y_test, y_pred_knn, average='weighted')

print(f"Accuracy: {accuracy_knn:.4f}")
print(f"Precision: {precision_knn:.4f}")
print(f"Recall: {recall_knn:.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_knn, zero_division=0))
```

```
--- Logistic Regression Metrics ---
Accuracy: 0.4878
Precision: 0.5017
Recall: 0.4878

Classification Report:
              precision    recall  f1-score   support
```

```
              0.0       0.44      0.67      0.53        18
              1.0       0.57      0.36      0.44        22
             10.0       0.00      0.00      0.00         1

         accuracy                           0.49        41
        macro avg       0.34      0.34      0.33        41
     weighted avg       0.50      0.49      0.47        41


     --- KNN Metrics ---
     Accuracy: 0.5854
     Precision: 0.5706
     Recall: 0.5854

     Classification Report:
                   precision    recall  f1-score   support

              0.0       0.58      0.39      0.47        18
              1.0       0.59      0.77      0.67        22
             10.0       0.00      0.00      0.00         1

         accuracy                           0.59        41
        macro avg       0.39      0.39      0.38        41
     weighted avg       0.57      0.59      0.56        41

     /usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-define
       _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
     /usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-define
       _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
Start coding or generate with AI.
```

SVM_assignment

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score
```

Step 1: Load the dataset

```
df = pd.read_csv('/content/Diabates_dirty.csv')
print(df)
```

```
     Age        BMI  Blood_Pressure Cholesterol  Diabetes
0   44.0  19.478891              95         250         0
1   54.0  28.807859              87         155         0
2   60.0  22.253709              83         175         1
3   49.0  12.704033              83         213         1
4   36.0  25.287065             135         208         1
..   ...        ...             ...         ...       ...
95  47.0  29.573028             113         163         0
96  27.0  14.125587              87         209         0
97  60.0  35.160713             119         179         0
98  58.0  16.179365             162         184         1
99  20.0  32.499443             121         154         1

[100 rows x 5 columns]
```

Step 2: Data Inspection and Preprocessing

```
print("Initial Dataset Info:")
df.info()
```

```
Initial Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             99 non-null     float64
 1   BMI             99 non-null     float64
 2   Blood_Pressure  100 non-null    int64
 3   Cholesterol     100 non-null    object
 4   Diabetes        100 non-null    int64
dtypes: float64(2), int64(2), object(1)
memory usage: 4.0+ KB
```

Check for missing values

```
print("\nMissing values before cleaning:")
print(df.isnull().sum())
```

```
Missing values before cleaning:
Age               1
BMI               1
Blood_Pressure    0
Cholesterol       0
Diabetes          0
dtype: int64
```

```
print("\n--- Converting non-numeric strings to numbers ---")

for col in df.drop('Diabetes', axis=1).columns:
    # Use pd.to_numeric with errors='coerce'.
```

```
        # This will turn any string that can't be converted into a number into NaN.
        df[col] = pd.to_numeric(df[col], errors='coerce')

print("--- Converting non-numeric strings to numbers ---")
```

```
# Now, we handle the NaNs that were created and any that were there before
print("Handling missing values created from string conversion...")
for col in df.columns:
    if df[col].isnull().any():
        # Fill NaN values with the mean of the column
        mean_val = df[col].mean()
        df[col] = df[col].fillna(mean_val)
        print(f"Filled missing values in '{col}'.")
```

```
Handling missing values created from string conversion...
Filled missing values in 'Age'.
Filled missing values in 'BMI'.
Filled missing values in 'Cholesterol'.
```

Step 2: Define Features (X) and Target (y)

```
X = df.drop('Diabetes', axis=1)
y = df['Diabetes']
```

Step 3: Split the data into training and testing sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 4: Scale the features

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Step 5: Initialize and train the models

```
# Logistic Regression
log_reg = LogisticRegression(random_state=42)
log_reg.fit(X_train_scaled, y_train)
y_pred_log_reg = log_reg.predict(X_test_scaled)
```

```
# K-Nearest Neighbors (KNN)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)
y_pred_knn = knn.predict(X_test_scaled)
```

```
# Support Vector Machine (SVM)
svm = SVC(kernel='linear', random_state=42)
svm.fit(X_train_scaled, y_train)
y_pred_svm = svm.predict(X_test_scaled)
```

Step 6: Evaluate the models

```
# Logistic Regression Metrics
acc_log_reg = accuracy_score(y_test, y_pred_log_reg)
prec_log_reg = precision_score(y_test, y_pred_log_reg)
rec_log_reg = recall_score(y_test, y_pred_log_reg)
```

```
# KNN Metrics
acc_knn = accuracy_score(y_test, y_pred_knn)
prec_knn = precision_score(y_test, y_pred_knn)
rec_knn = recall_score(y_test, y_pred_knn)
```

```
# SVM Metrics
acc_svm = accuracy_score(y_test, y_pred_svm)
prec_svm = precision_score(y_test, y_pred_svm)
rec_svm = recall_score(y_test, y_pred_svm)
```

Step 7: Compare the results

```
results_df = pd.DataFrame({
    'Model': ['Logistic Regression', 'K-Nearest Neighbors (KNN)', 'Support Vector Machine (SVM)'],
    'Accuracy': [acc_log_reg, acc_knn, acc_svm],
    'Precision': [prec_log_reg, prec_knn, prec_svm],
    'Recall': [rec_log_reg, rec_knn, rec_svm]
})
```

```
print(results_df)
```

```
                         Model  Accuracy  Precision    Recall
0          Logistic Regression      0.45      0.500  0.272727
1    K-Nearest Neighbors (KNN)      0.55      0.625  0.454545
2  Support Vector Machine (SVM)      0.45      0.500  0.272727
```

Start coding or generate with AI.

Step 1: Import Necessary Libraries

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder , MinMaxScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score , confusion_matrix , classification_report
import matplotlib.pyplot as plt
import seaborn as sns
```

Step 2: Load and Preprocess the Data

```
# Load the dataset
df = pd.read_csv('/content/rainy_weather.csv')
```

```
# Convert "Date" to detetime and extract features

df['Date'] = pd.to_datetime(df['Date'],format = '%d/%m/%y')
df['Year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
df['Day'] = df['Date'].dt.day
df.drop('Date', axis = 1 , inplace = True)
```

```
# Encode categorical Features

catgorical_cols = ['Location', 'RainToday','RainTomorrow']

for col in catgorical_cols:
    if df[col].dtype == 'object':
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])
```

Step 3: Prepare Features and Target Variable

```
# Separate Feature and Target Variable
x = df.drop('RainTomorrow', axis = 1)
y = df['RainTomorrow']
```

Step 4: Scale the Features

```
# scale the Features

scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(x)
```

Step 5: Split the Data into Training and Testing Sets

```
x_train , x_test ,y_train , y_test  = train_test_split(x_scaled, y, test_size = 0.2, random_state = (42))
```

step 6 : Train the Naive Bayes Model

```
# Initialize and train the Gaussian Naive Bayes Model
model = GaussianNB()
model.fit(x_train , y_train)
```

```
GaussianNB()
```

step 7 : Evaluate the Model

```
# Make Prediction on the test data
y_pred = model.predict(x_test)
```

```
# Calculate and print the accuracy
accuracy  = accuracy_score(y_test , y_pred)
print(f"Accuracy : {accuracy}")
```

```
Accuracy : 0.73
```

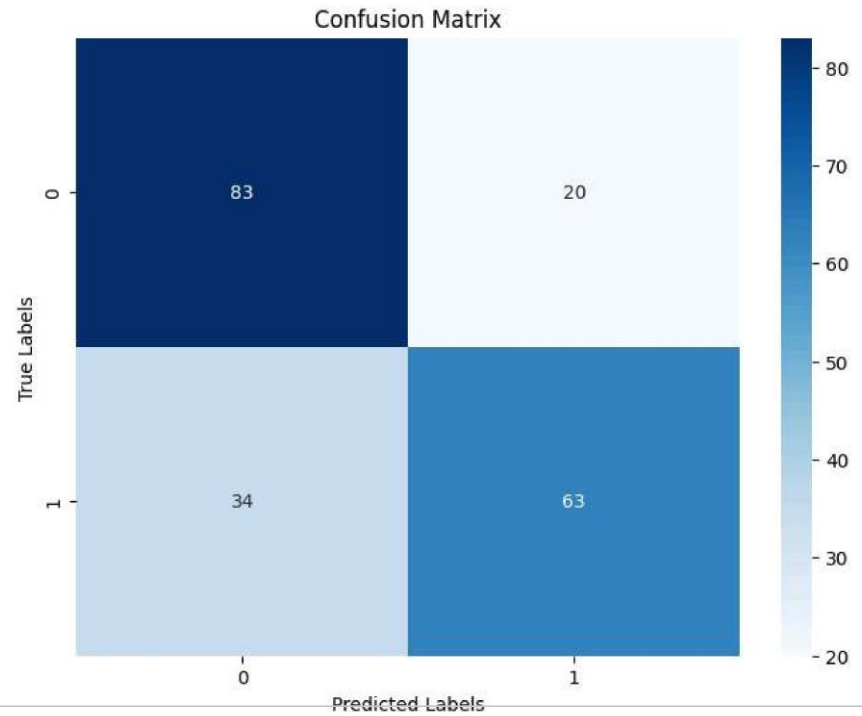
```
df.shape
```

```
(1000, 16)
```

```
# Print the classification report
print("\n Classification report")
print(classification_report(y_test , y_pred))
```

```
 Classification report
              precision    recall  f1-score   support

           0       0.71      0.81      0.75       103
           1       0.76      0.65      0.70        97

    accuracy                           0.73       200
   macro avg       0.73      0.73      0.73       200
weighted avg       0.73      0.73      0.73       200
```

```
# Genrate and Visualize the confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize= (8,6))
sns.heatmap(cm, annot = True , fmt = 'd' , cmap = 'Blues')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```



```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import pandas as pd
```

```
#
# Load the  dataset
#
df = pd.read_csv('InformationGainData.csv')
df.drop(['RID'],axis=1,inplace=True)
print(df)
```

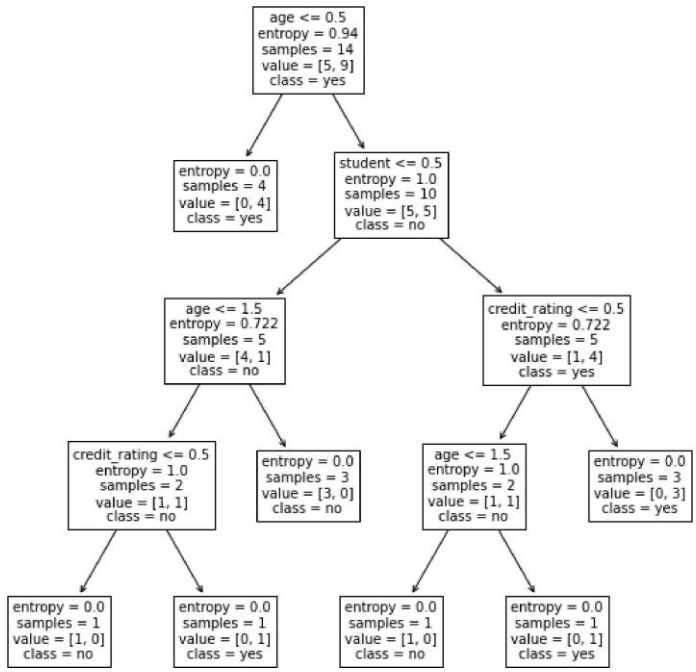
```
           age  income student credit_rating Class:buys computer
0        youth    high      no          fair                  no
1        youth    high      no     excellent                  no
2   middle_aged    high      no          fair                 yes
3       senior  medium      no          fair                 yes
4       senior     low     yes          fair                 yes
5       senior     low     yes     excellent                  no
6   middle_aged     low     yes     excellent                 yes
7        youth  medium      no          fair                  no
8        youth     low     yes          fair                 yes
9       senior  medium     yes          fair                 yes
10       youth  medium     yes     excellent                 yes
11  middle_aged  medium      no     excellent                 yes
12  middle_aged    high     yes          fair                 yes
13      senior  medium      no     excellent                  no
```

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df['age']=le.fit_transform(df['age'])
df['income']=le.fit_transform(df['income'])
df['student']=le.fit_transform(df['student'])
df['credit_rating']=le.fit_transform(df['credit_rating'])
df['Class:buys computer']=le.fit_transform(df['Class:buys computer'])
df
```

	age	income	student	credit_rating	Class:buys computer
0	2	0	0	1	0
1	2	0	0	0	0
2	0	0	0	1	1
3	1	2	0	1	1
4	1	1	1	1	1
5	1	1	1	0	0
6	0	1	1	0	1
7	2	2	0	1	0
8	2	1	1	1	1
9	1	2	1	1	1
10	2	2	1	0	1
11	0	2	0	0	1
12	0	0	1	1	1
13	1	2	0	0	0

```
X_train=X
y_train=y
feature_names=["age",   "income",   "student",  "credit_rating"]
target_names=["no","yes"]
#,  "Class:buys computer"]
```

```
clf_tree = DecisionTreeClassifier(criterion='entropy', random_state=1)
clf_tree.fit(X_train, y_train)
#
# Plot the decision tree
#
fig, ax = plt.subplots(figsize=(10, 10))
tree.plot_tree(clf_tree, feature_names=feature_names, class_names=target_names, fontsize=10)
plt.show()

#age:youth, income:low, student:yes, credit_rating:fair
#Expected output: Yes
#test data

type(X_train)
```




```
pandas.core.frame.DataFrame
```

```
#age:youth, income:low, student:yes, credit_rating:fair
#2,1,1,1

X_test=[[2,1,1,1]]
y_test = [1]
y_pred = clf_tree.predict(X_test)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but DecisionTreeCl
  warnings.warn(
```

```
from sklearn.metrics import accuracy_score
```

```
print ("Accuracy : ", accuracy_score(y_test,y_pred)*100)
```

```
Accuracy :  100.0
```

```
print(y_pred)
```

```
[1]
```

```
#age: senior, income: medium, student: no, credit_rating: excellent
#1,2,0,0
#y_test=0
X_test=[[1,2,0,0]]
y_test = [0]
```

```
y_pred = clf_tree.predict(X_test)
```

```
[0]
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but DecisionTreeCl
  warnings.warn(
```

```
print(y_pred)
```

```
[0]
```

```
import os
import pandas as pd
import numpy as np

# Visualization
import matplotlib.pyplot as plt
from sklearn.metrics import (
    accuracy_score, precision_recall_fscore_support,
    classification_report, confusion_matrix, roc_curve, auc, RocCurveDisplay
)

# Modeling
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.utils.multiclass import type_of_target
from joblib import dump
```

```
# ----------------------------
# 1) CONFIG
# ----------------------------
csv_path   = "/content/RandomForest_dataset.xlsx"     # your file
target_col = "Approved"                     # change if your target differs
model_out  = "rf_model.joblib"
```

```
# ----------------------------
# 2) LOAD DATA
# ----------------------------
df = pd.read_excel(csv_path)

print("Shape:", df.shape)
print("\nColumns:", list(df.columns))
print("\nHead:\n", df.head())

if target_col not in df.columns:
    raise ValueError(f"Target column '{target_col}' not found. Please set target_col correctly.")
```

```
Shape: (20, 7)

Columns: ['Income', 'CreditScore', 'Age', 'EmploymentYears', 'DebtToIncome', 'HasDefault', 'Approved']

Head:
    Income  CreditScore  Age  EmploymentYears  DebtToIncome  HasDefault  \
0   32000          580   24                1          0.45           0
1   45000          610   27                2          0.38           0
2   52000          640   29                3          0.33           0
3   61000          680   31                4          0.30           0
4   72000          720   35                6          0.28           0

   Approved
0         0
1         0
2         1
3         1
4         1
```

```
# ----------------------------
# 3) BASIC CLEANING
# ----------------------------
# Drop fully duplicate rows (optional)
df = df.drop_duplicates().reset_index(drop=True)

# Split features/target
y = df[target_col]
X = df.drop(columns=[target_col])

# Detect problem type
t_type = type_of_target(y)
print("\nDetected target type:", t_type)
if t_type not in ["binary", "multiclass"]:
    raise ValueError("This script is for classification (binary/multiclass). Target must be categorical (0/1, etc.).")
```

```
# Identify numeric vs categorical columns
numeric_cols = X.select_dtypes(include=[np.number]).columns.tolist()
categorical_cols = [c for c in X.columns if c not in numeric_cols]

print("\nNumeric columns:", numeric_cols)
print("Categorical columns:", categorical_cols if categorical_cols else "(none)")
```

```
Detected target type: binary

Numeric columns: ['Income', 'CreditScore', 'Age', 'EmploymentYears', 'DebtToIncome', 'HasDefault']
Categorical columns: (none)
```

```
# ------------------------------
# 4) PREPROCESSORS
# ------------------------------
numeric_preprocessor = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    # RF doesn't require scaling, but if ranges vary a lot, scaling is ok:
    # ("scaler", StandardScaler())
])

categorical_preprocessor = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore", sparse_output=False))
])

preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_preprocessor, numeric_cols),
        ("cat", categorical_preprocessor, categorical_cols)
    ],
    remainder="drop"
)
```

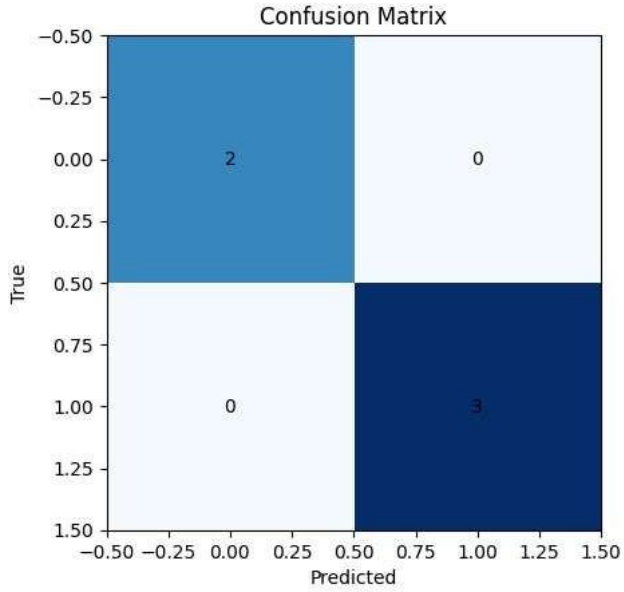
```
# ------------------------------
# 5) SPLIT
# ------------------------------
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.25,
    random_state=42,
    stratify=y
)
```

```
# ------------------------------
# 6) PIPELINE: PREPROCESS + RF
# ------------------------------
rf = RandomForestClassifier(
    n_estimators=400,
    max_depth=None,
    min_samples_leaf=2,
    max_features="sqrt",
    class_weight=None,     # set to "balanced" if your dataset is imbalanced
    random_state=42,
    n_jobs=-1,
    oob_score=False        # OOB needs bootstrap=True and no CV; we do test split + optional CV
)

model = Pipeline(steps=[
    ("preprocess", preprocessor),
    ("rf", rf)
])
```

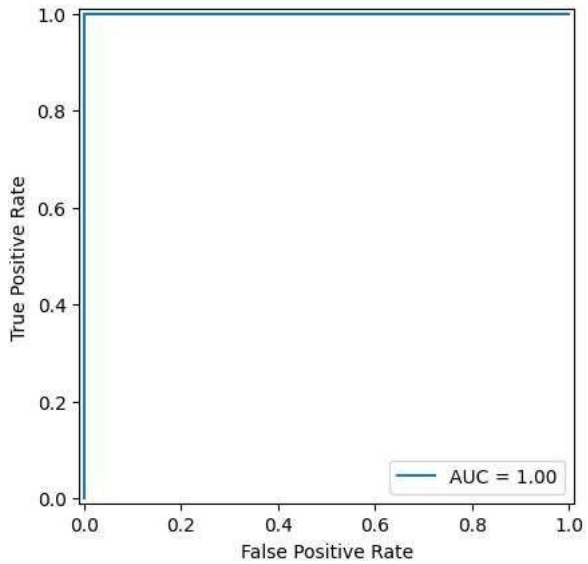
```
# ------------------------------------------
# 7) TRAIN
# ------------------------------------------
model.fit(X_train, y_train)

# ------------------------------------------
```

```
# 8) EVALUATE
# ----------------------------------------------
pred = model.predict(X_test)
print("\n=== Classification Report (Test) ===")
print(classification_report(y_test, pred, digits=3))

acc = accuracy_score(y_test, pred)
prec, rec, f1, _ = precision_recall_fscore_support(y_test, pred, average="binary" if t_type=="binary" else "weighted")

print(f"Accuracy:  {acc:.3f}")
print(f"Precision: {prec:.3f}")
print(f"Recall:    {rec:.3f}")
print(f"F1-score:  {f1:.3f}")

# Confusion matrix
cm = confusion_matrix(y_test, pred)
print("\nConfusion Matrix:\n", cm)

fig, ax = plt.subplots()
im = ax.imshow(cm, cmap="Blues")
ax.set_title("Confusion Matrix")
ax.set_xlabel("Predicted")
ax.set_ylabel("True")
for (i, j), v in np.ndenumerate(cm):
    ax.text(j, i, str(v), ha="center", va="center")
plt.show()

# ROC curve (binary only)
if t_type == "binary":
    # get positive class probability (after pipeline)
    if hasattr(model.named_steps["rf"], "predict_proba"):
        proba = model.predict_proba(X_test)[:, 1]
        fpr, tpr, _ = roc_curve(y_test, proba)
        roc_auc = auc(fpr, tpr)
        print(f"\nROC-AUC: {roc_auc:.3f}")

        RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc).plot()
        plt.show()
```

```
=== Classification Report (Test) ===
              precision    recall  f1-score   support

           0      1.000     1.000     1.000         2
           1      1.000     1.000     1.000         3

    accuracy                          1.000         5
   macro avg      1.000     1.000     1.000         5
weighted avg      1.000     1.000     1.000         5

Accuracy:  1.000
Precision: 1.000
Recall:    1.000
F1-score:  1.000

Confusion Matrix:
 [[2 0]
 [0 3]]
```



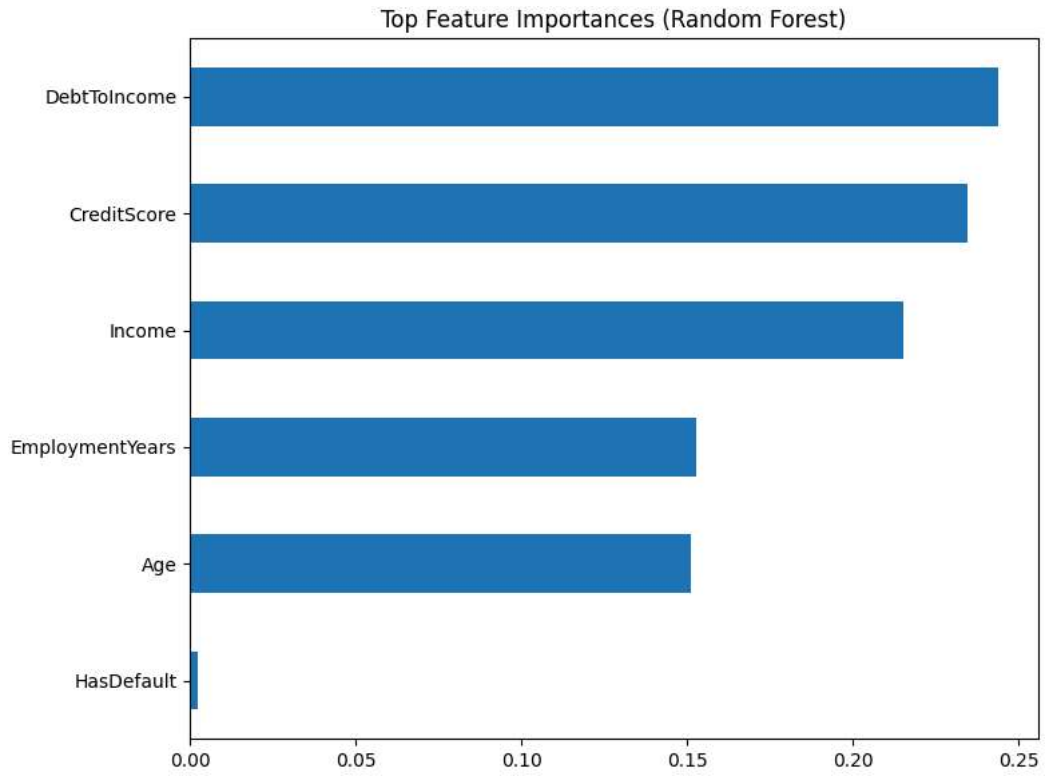

```
ROC-AUC: 1.000
```




```
# ----------------------------
# 9) FEATURE IMPORTANCE
# ----------------------------
# Get feature names after preprocessing to map importances
def get_feature_names(preprocessor, numeric_cols, categorical_cols):
    names = []
    # numeric
    names.extend(numeric_cols)
```

```
    # categorical (after one-hot)
    if categorical_cols:
        ohe = preprocessor.named_transformers_["cat"].named_steps["onehot"]
        cat_names = ohe.get_feature_names_out(categorical_cols).tolist()
        names.extend(cat_names)
    return names

# Extract trained RF and feature importances
rf_model = model.named_steps["rf"]
feat_names = get_feature_names(model.named_steps["preprocess"], numeric_cols, categorical_cols)
importances = rf_model.feature_importances_

# Combine & sort
fi = pd.Series(importances, index=feat_names).sort_values(ascending=False)
print("\n=== Top Feature Importances ===")
print(fi.head(20))

# Plot
fi.head(20).plot(kind="barh", figsize=(8,6))
plt.gca().invert_yaxis()
plt.title("Top Feature Importances (Random Forest)")
plt.tight_layout()
plt.show()
```

```
=== Top Feature Importances ===
DebtToIncome       0.243724
CreditScore        0.234589
Income             0.215167
EmploymentYears    0.152783
Age                0.151236
HasDefault         0.002500
dtype: float64
```




```
# ----------------------------
# 10) OPTIONAL: CROSS-VALIDATION SCORE
# ----------------------------
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = cross_val_score(model, X, y, cv=cv, scoring="accuracy", n_jobs=-1)
print("\nCV Accuracy (5-fold): mean = %.3f, std = %.3f, scores = %s" % (cv_scores.mean(), cv_scores.std(), np.round(cv_scores,
```

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

```
import pandas as pd
```

```
features = pd.read_csv('customers_product.csv')
```

```
features.shape
```

```
(19, 1)
```

```
kmeans = KMeans(
    n_clusters=3,
    n_init=10,
    max_iter=300,
    random_state=42
)
```

```
kmeans.fit(features)
```

```
KMeans
KMeans(n_clusters=3, n_init=10, random_state=42)
```

```
kmeans.cluster_centers_
```

```
array([[16.66666667],
       [ 7.66666667],
       [ 1.85714286]])
```

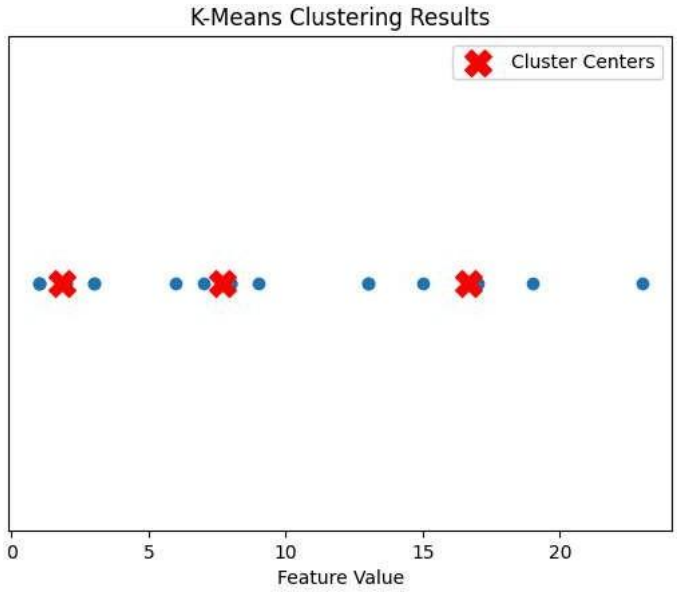
```
kmeans.n_iter_
```

```
2
```

```
features.shape
```

```
(19, 1)
```

```
import matplotlib.pyplot as plt

plt.scatter(features, [0] * len(features))
plt.scatter(kmeans.cluster_centers_, [0] * len(kmeans.cluster_centers_), c='red', marker='X', s=200, label='Cluster Centers')
plt.title('K-Means Clustering Results')
plt.xlabel('Feature Value')
plt.yticks([]) # Hide y-axis ticks for 1D visualization
plt.legend()
plt.show()
```



Start coding or generate with AI.