BANARSIDAS CHANDIWALA INSTITUTE OF INFORMATION TECHNOLOGY

Affiliated To Guru Gobind Singh Indraprastha University

SECTOR 16-C, DWARAKA, NEW DELHI

PRACTICAL FILE

SUBJECT: ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

SUBMITTED BY:	**SUBMITTED TO:**
Tanish Sharma	DR. ANU TANEJA
70311104422	ASST. PROFESSOR

S.NO.	Topic	Sign
1.	**Write a program to solve the missing data problem.**	
2.	**Write a program to implement simple linear regression on employee salary data.**	
3.	**Write a program to predict the price of a house using multiple linear regression.**	
4.	**Write a program to perform Logistic Regression on Social Network Ads.**	
5.	**Write a program to perform K-Means Clustering Mall Customers.**	
6.	**WAP to implement DFS Algorithm of AI in Python.**	
7.	**WAP to implement BFS Algorithm of AI in Python.**	
8.	**WAP to implement Backpropagation Algorithm in Python.**	
9.	**WAP to build a recommendation system in Python.**	

Q1. Write a program to solve the missing data problem.

```
import numpy as np
import pandas as pd
data=pd.read_csv('Data.csv')
data

x=data.iloc[:,:-1].values
x

y=data.iloc[:,-1].values
y

data.isnull()

data2=data.dropna(how='any')
data2

data3=data.dropna(how='all')
data3

data4=data.fillna(method='ffill')
data4

data5=data.fillna(method='bfill')
data5

from sklearn.impute import SimpleImputer
si=SimpleImputer(missing_values=np.NaN, strategy='mean')
si.fit(x[:,1:3])
si.transform(x[:,1:3])
x
y
```

1 to 10 of 10 entries Filter

Country	Age	Salary	Purchased
France	44	72000	No
Spain	27	48000	Yes
Germany	30	54000	No
Spain	38	61000	No
Germany	40		Yes
France	35	58000	Yes
Spain		52000	No
France	48	79000	Yes
Germany	50	83000	No
France	37	67000	Yes

Dataset

```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 nan]
 ['France' 35.0 58000.0]
 ['Spain' nan 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

independent variable

	Country	Age	Salary	Purchased
0	False	False	False	False
1	False	False	False	False
2	False	False	False	False
3	False	False	False	False
4	False	False	True	False
5	False	False	False	False
6	False	True	False	False
7	False	False	False	False
8	False	False	False	False
9	False	False	False	False

isnull()

```
['No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes']
```

Dependent variable

```
data.isnull()
```

	Country	Age	Salary	Purchased
0	False	False	False	False
1	False	False	False	False
2	False	False	False	False
3	False	False	False	False
4	False	False	True	False
5	False	False	False	False
6	False	True	False	False
7	False	False	False	False
8	False	False	False	False
9	False	False	False	False

```
data.dropna(how='any')
```

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
5	France	35.0	58000.0	Yes
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

```
data.dropna(how='all')
```

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

```
data.fillna(method='ffill')
```

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	61000.0	Yes
5	France	35.0	58000.0	Yes
6	Spain	35.0	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

```
data.fillna(method='bfill')
```

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	58000.0	Yes
5	France	35.0	58000.0	Yes
6	Spain	48.0	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

```
si.transform(x[:,1:3])
```

```
array([[4.40000000e+01, 7.20000000e+04],
       [2.70000000e+01, 4.80000000e+04],
       [3.00000000e+01, 5.40000000e+04],
       [3.80000000e+01, 6.10000000e+04],
       [4.00000000e+01, 6.37777778e+04],
       [3.50000000e+01, 5.80000000e+04],
       [3.87777778e+01, 5.20000000e+04],
       [4.80000000e+01, 7.90000000e+04],
       [5.00000000e+01, 8.30000000e+04],
       [3.70000000e+01, 6.70000000e+04]])
```

```
x
```

```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, nan],
       ['France', 35.0, 58000.0],
       ['Spain', nan, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

```
y
```

```
array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'],
      dtype=object)
```

Q2. Write a program to implement simple linear regression on employee salary data.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
dataset=pd.read_csv('Salary_Data.csv')
dataset x=dataset.iloc[:,:-1].values
x
y=dataset.iloc[:,-1].values
y

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=5)

print(x_train)
print(y_train)
print(x_test)
print(y_test)
from sklearn.linear_model import LinearRegression
reg=LinearRegression()
reg.fit(x_train, y_train)
y_pred=reg.predict(x_test)
y_pred
plt.scatter(x_test, y_test, color='red')
plt.plot(x_test, y_pred, color='blue')
plt.title('Salary vs Experience')
plt.xlabel('Experience (Years)')
plt.ylabel('Salary')
print(y_test-y_pred)
intercept=reg.intercept_
intercept slope=reg.coef_ slope
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
mean_absolute_error(y_test, y_pred)
mean_squared_error(y_test, y_pred)
r2_score(y_test, y_pred)
```

Salary_Data.csv

1 to 25 of 30 entries Filter

YearsExperience	Salary
1.1	39343
1.3	46205
1.5	37731
2	43525
2.2	39891
2.9	56642
3	60150
3.2	54445
3.2	64445
3.7	57189
3.9	63218
4	55794
4	56957
4.1	57081
4.5	61111
4.9	67938
5.1	66029
5.3	83088
5.9	81363
6	93940
6.8	91738
7.1	98273
7.9	101302
8.2	113812
8.7	109431

```
dataset.isnull().sum()
```

```
YearsExperience    0
Salary             0
dtype: int64
```

```
independent variable [[ 1.1]
 [ 1.3]
 [ 1.5]
 [ 2. ]
 [ 2.2]
 [ 2.9]
 [ 3. ]
 [ 3.2]
 [ 3.2]
 [ 3.7]
 [ 3.9]
 [ 4. ]
 [ 4. ]
 [ 4.1]
 [ 4.5]
 [ 4.9]
 [ 5.1]
 [ 5.3]
 [ 5.9]
 [ 6. ]
 [ 6.8]
 [ 7.1]
 [ 7.9]
 [ 8.2]
 [ 8.7]
 [ 9. ]
 [ 9.5]
 [ 9.6]
 [10.3]
 [10.5]]
```

```
dependent variable [ 39343  46205  37731  43525  39891  56642  60150  54445  64445  57189
  63218  55794  56957  57081  61111  67938  66029  83088  81363  93940
  91738  98273 101302 113812 109431 105582 116969 112635 122391 121872]
```

```
print(x_train)
```

```
[[ 1.5]
 [ 4.1]
 [ 9.5]
 [ 7.1]
 [10.3]
 [ 1.1]
 [ 5.3]
 [ 2.9]
 [ 1.3]
 [ 9.6]
 [ 4. ]
 [ 6.8]
 [ 6. ]
 [ 8.7]
 [ 3.2]
 [ 2.2]
 [ 3.2]
 [ 3.7]
 [ 5.1]
 [ 7.9]
 [ 3. ]
 [ 4.9]
 [ 4.5]
 [ 2. ]]
```

```
print(y_train)
```

```
[ 37731  57081 116969  98273 122391  39343  83088  56642  46205 112635
  56957  91738  93940 109431  54445  39891  64445  57189  66029 101302
  60150  67938  61111  43525]
```

```
print(x_test)
```

```
[[ 4. ]
 [10.5]
 [ 8.2]
 [ 9. ]
 [ 5.9]
 [ 3.9]]
```

```
print(y_test)
```

```
[ 55794 121872 113812 105582  81363  63218]
```

```
y_pred
```

```
array([ 63822.10276786, 125176.91866803, 103466.75304182, 111018.11499876,
        81756.5874156 ,  62878.18252324])
```

```
print(y_test-y_pred)
```

```
[-8028.10276786 -3304.91866803 10345.24695818 -5436.11499876
  -393.5874156    339.81747676]
```

```
intercept
```

```
26065.292983138308
```

```
slope
```

```
array([9439.20244618])
```

```
mean_absolute_error(y_test, y_pred)
```

```
4641.2980475332
```

```
mean_squared_error(y_test, y_pred)
```

```
35369798.221735574
```

```
Text(0, 0.5, 'Salary')
```


```
r2_score(y_test, y_pred)
```

```
0.9439628569611376
```

Q3. Write a program to predict the price of a house using multiple linear regression.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
dataset=pd.read_csv('HousePrice.csv')
dataset
dataset.shape
dataset.isnull()
dataset=dataset.dropna(how='all')
dataset
dataset.shape
dataset.isnull()
dataset=dataset.fillna(method='ffill')
dataset
x=dataset.iloc[:,:-1].values
x
y=dataset.iloc[:,-1].values
y
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=0)

print(x_train)
print(y_train)
print(x_test)
print(y_test)

from  sklearn.linear_model  import  LinearRegression
reg=LinearRegression()
reg.fit(x_train, y_train)
y_pred=reg.predict(x_test)
y_pred
print(y_test-y_pred)
intercept=reg.intercept_
intercept
slope=reg.coef_
slope
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
mean_absolute_error(y_test, y_pred)
mean_squared_error(y_test, y_pred)
r2_score(y_test, y_pred)
```

HousePrice.csv

1 to 21 of 21 entries Filter

Area	Bedrooms	Construct	Price
2600	2	10	500000
4000	4	30	100000
8000	6	2	800000
3000		40	500000
10000	2	1	6000000
1000	3	5	300000
500	2	8	2500000
2000	2	10	450000
8000	2	2	500000
500	3	6	2000000

Show 25 per page

```
dataset.isnull()
```

	Area	Bedrooms	Construct	Price
0	False	False	False	False
1	False	False	False	False
2	False	False	False	False
3	False	True	False	False
4	False	False	False	False
5	False	False	False	False
6	False	False	False	False
7	False	False	False	False
8	False	False	False	False
9	False	False	False	False
10	True	True	True	True
11	True	True	True	True
12	True	True	True	True
13	True	True	True	True
14	True	True	True	True
15	True	True	True	True
16	True	True	True	True
17	True	True	True	True

```
new_dataset.isnull()
```

	Area	Bedrooms	Construct	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
5	False	False	False	False
6	False	False	False	False
7	False	False	False	False
8	False	False	False	False
9	False	False	False	False
10	False	False	False	False
11	False	False	False	False
12	False	False	False	False
13	False	False	False	False
14	False	False	False	False
15	False	False	False	False
16	False	False	False	False
17	False	False	False	False

```
x
```

```
array([[2.6e+03, 2.0e+00, 1.0e+01],
       [4.0e+03, 4.0e+00, 3.0e+01],
       [8.0e+03, 6.0e+00, 2.0e+00],
       [3.0e+03, 6.0e+00, 4.0e+01],
       [1.0e+04, 2.0e+00, 1.0e+00],
       [1.0e+03, 3.0e+00, 5.0e+00],
       [5.0e+02, 2.0e+00, 8.0e+00],
       [2.0e+03, 2.0e+00, 1.0e+01],
       [8.0e+03, 2.0e+00, 2.0e+00],
       [5.0e+02, 3.0e+00, 6.0e+00],
       [5.0e+02, 3.0e+00, 6.0e+00],
       [5.0e+02, 3.0e+00, 6.0e+00],
       [5.0e+02, 3.0e+00, 6.0e+00],
       [5.0e+02, 3.0e+00, 6.0e+00],
       [5.0e+02, 3.0e+00, 6.0e+00],
       [5.0e+02, 3.0e+00, 6.0e+00],
       [5.0e+02, 3.0e+00, 6.0e+00],
       [5.0e+02, 3.0e+00, 6.0e+00],
       [5.0e+02, 3.0e+00, 6.0e+00],
       [5.0e+02, 3.0e+00, 6.0e+00],
       [5.0e+02, 3.0e+00, 6.0e+00]])
```

```
y
```

```
0       500000.0
1       100000.0
2       800000.0
3       500000.0
4      6000000.0
5       300000.0
6      2500000.0
7       450000.0
8       500000.0
9      2000000.0
10     2000000.0
11     2000000.0
12     2000000.0
13     2000000.0
14     2000000.0
15     2000000.0
16     2000000.0
17     2000000.0
18     2000000.0
19     2000000.0
20     2000000.0
Name: Price, dtype: float64
```

```
print('x_train',x_train)
```

```
x_train [[5.0e+02 3.0e+00 6.0e+00]
 [5.0e+02 3.0e+00 6.0e+00]
 [5.0e+02 3.0e+00 6.0e+00]
 [5.0e+02 2.0e+00 8.0e+00]
 [5.0e+02 3.0e+00 6.0e+00]
 [1.0e+04 2.0e+00 1.0e+00]
 [8.0e+03 6.0e+00 2.0e+00]
 [1.0e+03 3.0e+00 5.0e+00]
 [5.0e+02 3.0e+00 6.0e+00]
 [5.0e+02 3.0e+00 6.0e+00]
 [2.0e+03 2.0e+00 1.0e+01]
 [5.0e+02 3.0e+00 6.0e+00]
 [3.0e+03 6.0e+00 4.0e+01]
 [2.6e+03 2.0e+00 1.0e+01]
 [5.0e+02 3.0e+00 6.0e+00]
 [5.0e+02 3.0e+00 6.0e+00]]
```

```
print('y_train',y_train)
```

```
y_train 10    2000000.0
14    2000000.0
18    2000000.0
6     2500000.0
19    2000000.0
4     6000000.0
2      800000.0
5      300000.0
16    2000000.0
9     2000000.0
7      450000.0
17    2000000.0
3      500000.0
0      500000.0
15    2000000.0
12    2000000.0
Name: Price, dtype: float64
```

```
print('x_test',x_test)
```

```
x_test [[8.e+03 2.e+00 2.e+00]
 [5.e+02 3.e+00 6.e+00]
 [5.e+02 3.e+00 6.e+00]
 [4.e+03 4.e+00 3.e+01]
 [5.e+02 3.e+00 6.e+00]]
```

```
print('y_test',y_test)
```

```
y_test 8      500000.0
13    2000000.0
20    2000000.0
1      100000.0
11    2000000.0
Name: Price, dtype: float64
```

```
y_predict
```

```
array([3936913.24996648, 1566610.13081333, 1566610.13081333,
       1447125.75072891, 1566610.13081333])
```

```
intercept
```

```
3076719.499604637
```

```
slope
```

```
array([ 2.38632913e+02, -5.05727611e+05, -1.87071652e+04])
```

```
mean_absolute_error(y_test, y_predict)
```

```
1216841.721651081
```

```
mean_squared_error(y_test, y_predict)
```

```
2838120162442.605
```

Q4. Write a program to perform Logistic Regression on Social Network Ads.

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

dataset=pd.read_csv('Social_Network_Ads.csv')
data
data.shape
data.isnull().sum()
x=data.iloc[:,2:4].values
x
y=data.iloc[:,4].values
y
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=0)
x_train
x_test
y_train
y_test
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
x_train=ss.fit_transform(x_train)
x_train
x_test=ss.transform(x_test)


from sklearn.linear_model
import LogisticRegression
lr=LogisticRegression()
lr.fit(x_train, y_train)
y_pred=lr.predict(x_test)
y_pred

comp=pd.DataFrame({"Actual":y_test, "Predicted":y_pred})

comp from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test, y_pred) cm import seaborn as sns sns.heatmap(cm,annot=True)
```

Social_Network_Ads.csv

1 to 25 of 400 entries

User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0
15606274	Female	26	52000	0
15746139	Male	20	86000	0
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1
15617482	Male	45	26000	1
15704583	Male	46	28000	1
15621083	Female	48	29000	1
15649487	Male	45	22000	1
15736760	Female	47	49000	1
15714658	Male	48	41000	1
15599081	Female	45	22000	1
15705113	Male	46	23000	1

Show 25 per page 1 2 10 16

```
print(x)
```

```
[[    19  19000]
 [    35  20000]
 [    26  43000]
 [    27  57000]
 [    19  76000]
 [    27  58000]
 [    27  84000]
 [    32 150000]
 [    25  33000]
 [    35  65000]
 [    26  80000]
 [    26  52000]
 [    20  86000]
 [    32  18000]
 [    18  82000]
 [    29  80000]
 [    47  25000]
 [    45  26000]
 [    46  28000]
 [    48  29000]
 [    45  22000]
 [    47  49000]
 [    48  41000]
 [    45  22000]
 [    46  23000]
 [    47  20000]
 [    49  28000]
 [    47  30000]
 [    29  43000]
 [    31  18000]
```

```
dataset.isnull().sum()
```

```
User ID            0
Gender             0
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

```
dataset.shape
```

```
(400, 5)
```

```
print(y)
```

```
[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 1
 1 1 0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 1 1 0 1 0 1 0 1 0 0 1 1 0 1 0 0 1
 1 0 1 1 0 1 1 0 0 1 0 0 1 1 1 1 1 0 1 1 1 1 0 1 1 0 1 0 1 0 1 1 1 1 0 0 0
 1 1 0 1 1 1 1 1 0 0 0 1 1 0 0 1 0 1 0 1 1 0 1 0 1 1 0 1 1 0 0 0 1 1 0 1 0
 0 1 0 1 0 0 1 1 0 0 1 1 0 1 1 0 0 1 0 1 0 1 1 1 0 1 0 1 1 1 0 1 1 1 1 0 1
 1 1 0 1 0 1 0 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 0 1]
```

```
print("x_train = ",x_train

x_train =  [[    58 144000
 [    59  83000]
 [    24  55000]
 [    26  35000]
 [    58  38000]
 [    42  80000]
 [    40  75000]
 [    59 130000]
 [    46  41000]
 [    41  60000]
 [    42  64000]
 [    37 146000]
 [    23  48000]
 [    25  33000]
 [    24  84000]
 [    27  96000]
 [    23  63000]
 [    48  33000]
 [    48  90000]
 [    42 104000]
 [    44  39000]
 [    32 120000]
 [    38  50000]
 [    32 135000]
 [    52  21000]
 [    53 104000]
```

```
print("x_test = ",x_test)

x_test =  [[    30  87000]
 [    38  50000]
 [    35  75000]
 [    30  79000]
 [    35  50000]
 [    27  20000]
 [    31  15000]
 [    36 144000]
 [    18  68000]
 [    47  43000]
 [    30  49000]
 [    28  55000]
 [    37  55000]
 [    39  77000]
 [    20  86000]
 [    32 117000]
 [    37  77000]
 [    19  85000]
 [    55 130000]
 [    35  22000]
 [    35  47000]
 [    47 144000]
 [    41  51000]
 [    47 105000]
 [    23  28000]
 [    49 141000]
 [    28  87000]
 [    29  80000]
 [    37  62000]
 [    32  86000]
```

```
print("y_train = ",y_train)

y_train =  [1 0 0 0 1 1 0 1 1 0 0 1 0 0 0 1 0 1 1 1 0 1 0 1 1 1 0 0 0 0 0 0 1 1 1 0 1
 0 0 1 0 1 0 1 0 0 1 1 1 1 0 1 0 1 0 0 1 0 0 1 0 0 0 0 0 1 1 1 1 0 0 0 1 0
 1 0 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1 0 1 1 1 0 0 1 1 0 0 1 1 0 1 0 0 1 1 0
 1 1 1 0 0 0 0 0 1 0 0 1 1 1 1 1 0 1 1 0 1 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0
 0 0 1 0 1 1 0 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 1 1 1 0
 0 0 0 0 0 1 1 1 1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 1
 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0 1 0 1
 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0 1 0 1 0 0 1
 0 0 1 0 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 0 0 0]
```

```
print("y_test = ",y_test)

y_test =  [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 1 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1
 0 0 0 0 1 1]
```

```
x_train = ob.fit_transform(x_train)
x_train
```

```
array([[ 1.92295008e+00,  2.14601566e+00],
       [ 2.02016082e+00,  3.78719297e-01],
       [-1.38221530e+00, -4.32498705e-01],
       [-1.18779381e+00, -1.01194013e+00],
       [ 1.92295008e+00, -9.25023920e-01],
       [ 3.67578135e-01,  2.91803083e-01],
       [ 1.73156642e-01,  1.46942725e-01],
       [ 2.02016082e+00,  1.74040666e+00],
       [ 7.56421121e-01, -8.38107706e-01],
       [ 2.70367388e-01, -2.87638347e-01],
       [ 3.67578135e-01, -1.71750061e-01],
       [-1.18475597e-01,  2.20395980e+00],
       [-1.47942605e+00, -6.35303205e-01],
       [-1.28500455e+00, -1.06988428e+00],
       [-1.38221530e+00,  4.07691369e-01],
       [-1.09058306e+00,  7.55356227e-01],
       [-1.47942605e+00, -2.00722133e-01],
       [ 9.50842613e-01, -1.06988428e+00],
       [ 9.50842613e-01,  5.81523798e-01],
       [ 3.67578135e-01,  9.87132798e-01],
       [ 5.61999628e-01, -8.96051849e-01],
       [-6.04529329e-01,  1.45068594e+00],
       [-2.12648508e-02, -5.77359062e-01],
```

```
x_test = ob.transform(x_test)
x_test
```

```
array([[3.00e+01, 8.70e+04],
       [3.80e+01, 5.00e+04],
       [3.50e+01, 7.50e+04],
       [3.00e+01, 7.90e+04],
       [3.50e+01, 5.00e+04],
       [2.70e+01, 2.00e+04],
       [3.10e+01, 1.50e+04],
       [3.60e+01, 1.44e+05],
       [1.80e+01, 6.80e+04],
       [4.70e+01, 4.30e+04],
       [3.00e+01, 4.90e+04],
       [2.80e+01, 5.50e+04],
       [3.70e+01, 5.50e+04],
       [3.90e+01, 7.70e+04],
       [2.00e+01, 8.60e+04],
       [3.20e+01, 1.17e+05],
       [3.70e+01, 7.70e+04],
       [1.90e+01, 8.50e+04],
       [5.50e+01, 1.30e+05],
       [3.50e+01, 2.20e+04],
       [3.50e+01, 4.70e+04],
       [4.70e+01, 1.44e+05],
       [4.10e+01, 5.10e+04],
       [4.70e+01, 1.05e+05],
       [2.30e+01, 2.80e+04],
       [4.90e+01, 1.41e+05],
       [2.80e+01, 8.70e+04],
       [2.90e+01, 8.00e+04],
       [3.70e+01, 6.20e+04],
```

```
ob = LogisticRegression()
ob.fit(x_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
print(y_predict)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1]
```

```
comp
```

	Actual:	predicted:
0	0	1
1	0	1
2	0	1
3	0	1
4	0	1
...	...	...
75	0	1
76	0	1
77	0	1
78	1	1
79	1	1

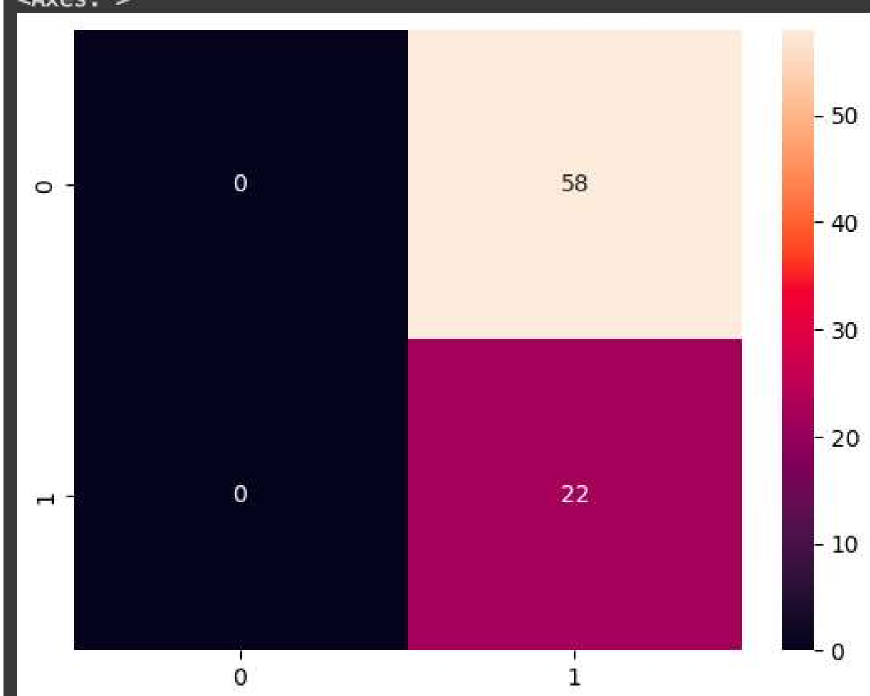
80 rows × 2 columns

```
cm
```

```
array([[ 0, 58],
       [ 0, 22]])
```

```
sns.heatmap(cm, annot = True)
```

```
<Axes: >
```



Q5. Write a program to perform K-Means Clustering Mall Customers.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
dataset=pd.read_csv('Mall_Customers.csv')
dataset

x=dataset.iloc[:, [3,4]].values
x

from sklearn.cluster import KMeans
wcss=[]
for i in range(1,11):
km=KMeans(n_clusters=i)
km.fit(x)        #train on data
wcss.append(km.inertia_)
wcss

plt.plot(range(1,11), wcss)
plt.title('Elbow Test')
plt.xlabel('No of Clusters')
plt.ylabel('WCSS')

km=KMeans(n_clusters=5) km.fit(x)
y=km.fit_predict(x)
y

plt.scatter(x[y==0,0], x[y==0,1], c='red')
plt.scatter(x[y==1,0], x[y==1,1], c='blue')
plt.scatter(x[y==2,0], x[y==2,1], c='green')
plt.scatter(x[y==3,0], x[y==3,1], c='purple')
plt.scatter(x[y==4,0], x[y==4,1], c='orange')
plt.xlabel('Annual Income (in Thousands)')
plt.ylabel('Spending Score')
```

Mall_Customers.csv

1 to 25 of 200 entries

CustomerID	Genre	Age	Annual Income (k$)	Spending
0001	Male	19	15	39
0002	Male	21	15	81
0003	Female	20	16	6
0004	Female	23	16	77
0005	Female	31	17	40
0006	Female	22	17	76
0007	Female	35	18	6
0008	Female	23	18	94
0009	Male	64	19	3
0010	Female	30	19	72
0011	Male	67	19	14
0012	Female	35	19	99
0013	Female	58	20	15
0014	Female	24	20	77
0015	Male	37	20	13
0016	Male	22	20	79
0017	Female	35	21	35
0018	Male	20	21	66
0019	Male	52	23	29
0020	Female	35	23	98
0021	Male	35	24	35
0022	Male	25	24	73
0023	Female	46	25	5
0024	Male	31	25	73
0025	Female	54	28	14

x

```
array([[ 15,  39],
       [ 15,  81],
       [ 16,   6],
       [ 16,  77],
       [ 17,  40],
       [ 17,  76],
       [ 18,   6],
       [ 18,  94],
       [ 19,   3],
       [ 19,  72],
       [ 19,  14],
       [ 19,  99],
       [ 20,  15],
       [ 20,  77],
       [ 20,  13],
       [ 20,  79],
       [ 21,  35],
       [ 21,  66],
       [ 23,  29],
       [ 23,  98],
       [ 24,  35],
       [ 24,  73],
       [ 25,   5],
```

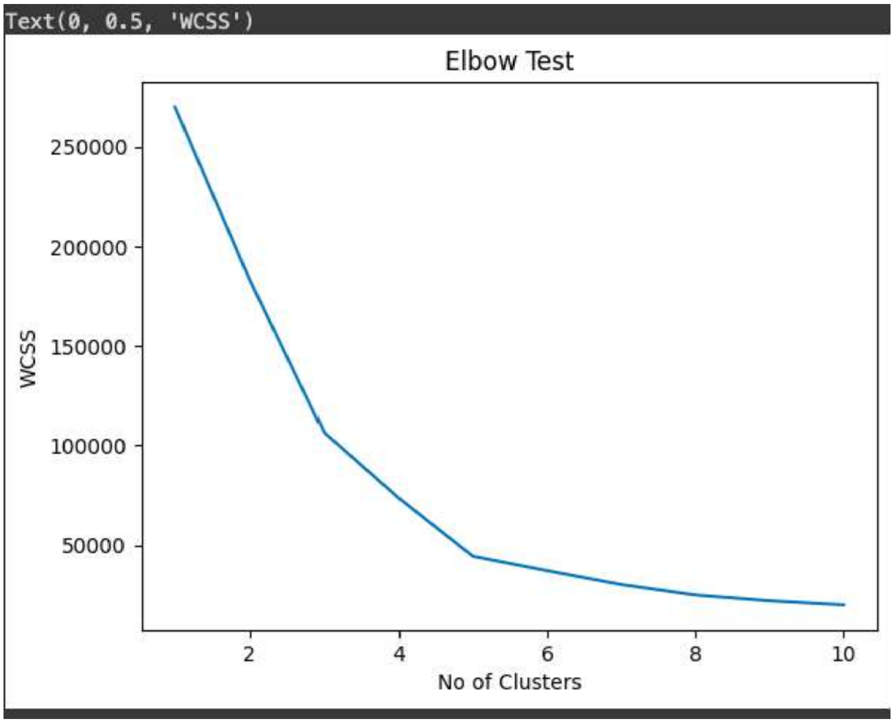
```
[269981.28,
 183653.32894736843,
 106348.37306211122,
 73679.78903948836,
 44448.4554479337,
 37233.814510710006,
 30273.394312070042,
 25018.781613414067,
 22131.92051101073,
 20065.930434100777]
```

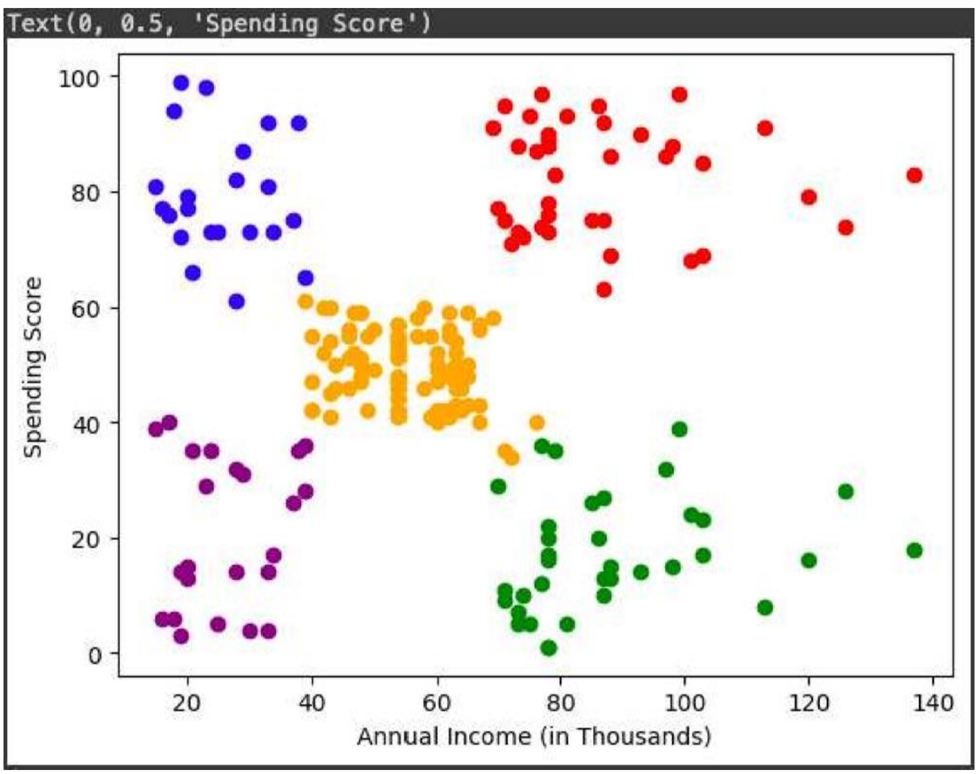
WCSS

```
KMeans
KMeans(n_clusters=5)
```

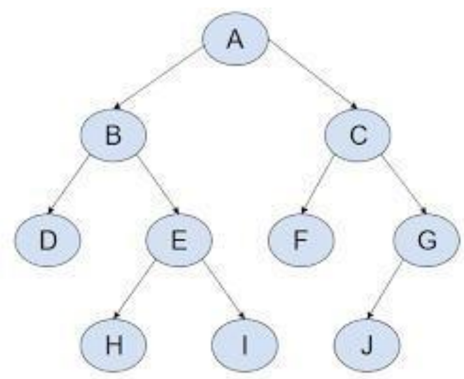
```
array([3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1,
       3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 4,
       3, 1, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 2, 0, 4, 0, 2, 0, 2, 0,
       4, 0, 2, 0, 2, 0, 2, 0, 2, 0, 4, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0,
       2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0,
       2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0,
       2, 0], dtype=int32)
```

y=km.fit_predict(x)


Text(0, 0.5, 'WCSS')
Elbow Test
WCSS
250000
200000
150000
100000
50000
2
4
6
8
10
No of Clusters


Text(0, 0.5, 'Spending Score')
Spending Score
100
80
60
40
20
0
20
40
60
80
100
120
140
Annual Income (in Thousands)

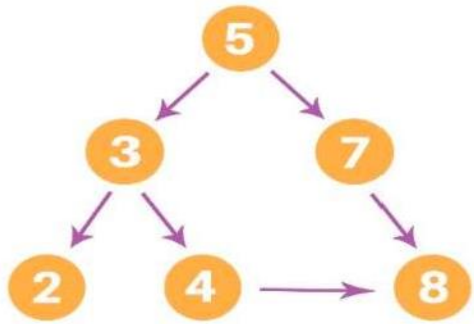
6. WAP to implement DFS Algorithm of AI in Python.




```
def dfs(graph,start,goal,stack,visited):
    stack.append(start)
    visited.append(start)
    print('The path traversed is:')
    while stack:
        element=stack.pop()
        print(element,end=" ")
        if(element==goal):
            break
        for neighbor in graph[element]:
            if neighbor not in visited:
                stack.append(neighbor)
                visited.append(neighbor)

graph={ 'A':['C','B'],
'B':['E','D'],
'C':['G','F'],
'D':[],
'E':['I','H'],
'F':[],
'G':['J'],
'H':[],
'I':[],
'J':[]

}

start='B'
goal='F'
visited=[]
stack=[]

dfs(graph,start,goal,visited,stack)
```

```
The path traversed is:
B D E H I
```

7. WAP to implement BFS Algorithm of AI in Python.



```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

visited = [] # List for visited nodes.
queue = []     #Initialize a queue

def bfs(visited, graph, node): #function for BFS
  visited.append(node)
  queue.append(node)

  while queue:          # Creating loop to visit each node
    m = queue.pop(0)
    print (m, end = " ")

    for neighbour in graph[m]:
```

```
        visited.append(neighbour)

        queue.append(neighbour)


# Driver Code

print("Following is the Breadth-First Search")

bfs(visited, graph, '5')    # function calling
```

```
Following is the Breadth-First Search
5 3 7 2 4 8
```

8. WAP to implement Backpropagation Algorithm in Python.

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0) #maximum of X array longitudinally
y = y/100

#Sigmoid Function
def sigmoid (x):
    return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)

#Variable initialization
epoch=5

 #Setting training iterations
lr=0.1

#Setting learning rate
inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer

#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))

#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
    #Forward Propogation
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+bout
    output = sigmoid(outinp)

 #Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)
    hiddengrad  =  derivatives_sigmoid(hlayer_act)

#how much hidden layer wts contributed to error
    d_hiddenlayer = EH * hiddengrad
            wout += hlayer_act.T.dot(d_output) *lr
```

```
 # dotproduct of nextlayererror and currentlayerop
    wh += X.T.dot(d_hiddenlayer) *lr
    print ("-----------Epoch-", i+1, "Starts ----------")
    print("Input: \n" + str(X))
    print("Actual Output: \n" + str(y))
    print("Predicted Output: \n" ,output)
    print ("-----------Epoch-", i+1, "Ends --------- \n")
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

```
————-Epoch- 1 Starts————-
Input:
[[0.66666667 1. ]
[0.33333333 0.55555556]
[1. 0.66666667]]
Actual Output:
[[0.92]
[0.86]
[0.89]]
Predicted Output:
[[0.81951208]
[0.8007242 ]
[0.82485744]]
————-Epoch- 1 Ends————-

...

————-Epoch- 5 Ends————-

Input:
[[0.66666667 1. ]
[0.33333333 0.55555556]
[1. 0.66666667]]
Actual Output:
[[0.92]
[0.86]
[0.89]]
Predicted Output:
[[0.8227362 ]
[0.80389106]
[0.82806747]]
```

9. WAP to build a recommendation system in Python.

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel

df = pd.read_csv('netflix_titles.csv')# Replace NaN with an empty string
df['description'] = df['description'].fillna('')

# Create a TfidfVectorizer and Remove stopwords
tfidf = TfidfVectorizer(stop_words='english')# Fit and transform the data to a tfidf matrix
tfidf_matrix = tfidf.fit_transform(df['description'])# Print the shape of the tfidf_matrix
tfidf_matrix.shape

# Compute the cosine similarity between each movie description
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
indices = pd.Series(df.index, index=df['title']).drop_duplicates()
def get_recommendations(title, cosine_sim=cosine_sim, num_recommend = 10):
    idx = indices[title]# Get the pairwsie similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    top_similar = sim_scores[1:num_recommend+1]# Get the movie indices
    movie_indices = [i[0] for i in top_similar]

# Return the top 10 most similar movies
    return df['title'].iloc[movie_indices]
get_recommendations('Power Rangers Zeo', num_recommend = 20)
```

```
7771                                    Power Rangers RPM
7773                                Power Rangers Samurai
7763                           Power Rangers Dino Thunder
8183             The Adventures of Sharkboy and Lavagirl
7765                            Power Rangers Jungle Fury
7781         Power Rangers Super Samurai: Trickster Treat
719                               Power Rangers Dino Fury
3946                                            Possessed
7764                               Power Rangers in Space
7780      Power Rangers Super Samurai: Stuck on Christmas
1179                         Mighty Morphin Power Rangers
2690                                               Code 8
7770                   Power Rangers Operation Overdrive
8559                                      The Witch Files
3452                                        Peaky Blinders
7617                          NOVA: The Impossible Flight
4744                               SWORDGAI The Animation
7777                            Power Rangers Super Megaforce
3986                                               The OA
7776    Power Rangers Samurai: Party Monsters (Hallowe...
```