DML

Sr. No	List of Practical Experiments
1	Write a program to implement Simple Linear Regression
2	Write a program to implement multiple Linear Regression
3	Write a program to implement K-nearest Neighbors (K-NN)/SVM
4	Write a program to implement Naïve Bayse / DT
5	Write a program to implement K-means clustering.
6	Write a program to implement Hierarchical clustering.
7	Write a program to build ANN.
8	Write a program to build CNN.
9	
10	

Practical 1

Aim: Write a program to implement Simple Linear Regression

Code:

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

dataset = pd.read_csv("Salary_Data.csv")
X = dataset.iloc[ : , : -1].valuesy
= dataset.iloc[ : , -1].values

print(X)
print("\n\n\n
")print(y)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/4, random_state = 0)

print(X_train)
print("\n\n\n
")
print(X_test)
print("\n\n\n
")
print(y_train)
print("\n\n\n
")print(y_test)

from sklearn.linear_model import LinearRegression
linear_regression = LinearRegression()
linear_regression.fit(X_train, y_train)

y_train_pred =
linear_regression.predict(X_train)y_test_pred =
linear_regression.predict(X_test)

plt.scatter(X_train, y_train, color = "green", marker = "+", label = "Observed data")
plt.plot(X_train, y_train_pred, color = "red", label = "Predicted data")
plt.xlabel("Years of experience")
plt.ylabel("Salary")
plt.title("Years of experience v/s Salary (Training dataset)")
plt.legend()
plt.show()

plt.scatter(X_test, y_test, color = "green", marker = "+", label = "Observed
```

```
data")plt.plot(X_test, y_test_pred, color = "red", label = "Predicted data")
plt.xlabel("Years of experience")
plt.ylabel("Salary")
plt.title("Years of experience v/s Salary (Testing dataset)")
```

```
plt.legend()
plt.show()
```

Output:

```
[[ 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]
```

```
[[ 5.3]
 [ 7.9]
 [ 2.9]
 [ 5.1]
 [ 3.2]
 [ 4.5]
 [ 8.2]
 [ 6.8]
 [ 1.3]
 [10.5]
 [ 3. ]
 [ 2.2]
 [ 5.9]
 [ 6. ]
```

Practical 2

Aim: Write a program to implement multiple Linear Regression

Code:

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

dataset = pd.read_csv('/content/50_Startups-
2.csv')x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

print(x)

print(y)

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import
OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [3])],
remainder='passthrough')
x = np.array(ct.fit_transform(x))

print(x)

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)

from sklearn.linear_model import LinearRegression
regressor  = LinearRegression()
regressor.fit(x_train, y_train)

y_pred =
regressor.predict(x_test)
```

```
np.set_printoptions(precision=2)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

OUTPUT:-

```
[[165349.2 136897.8 471784.1 'New York']
 [162597.7 151377.59 443898.53 'California']
 [153441.51 101145.55 407934.54 'Florida']
 [144372.41 118671.85 383199.62 'New York']
 [142107.34 91391.77 366168.42 'Florida']
 [131876.9 99814.71 362861.36 'New York']
 [134615.46 147198.87 127716.82 'California']
 [130298.13 145530.06 323876.68 'Florida']
 [120542.52 148718.95 311613.29 'New York']
 [123334.88 108679.17 304981.62 'California']
 [101913.08 110594.11 229160.95 'Florida']
 [100671.96 91790.61 249744.55 'California']
 [93863.75 127320.38 249839.44 'Florida']
 [91992.39 135495.07 252664.93 'California']
 [119943.24 156547.42 256512.92 'Florida']
 [114523.61 122616.84 261776.23 'New York']
 [78013.11 121597.55 264346.06 'California']
 [94657.16 145077.58 282574.31 'New York']
 [91749.16 114175.79 294919.57 'Florida']
 [86419.7 153514.11 0.0 'New York']
 [76253.86 113867.3 298664.47 'California']
 [78389.47 153773.43 299737.29 'New York']
 [73994.56 122782.75 303319.26 'Florida']
```

```
[192261.83 191792.06 191050.39 182901.99 166187.94 156991.12 156122.51
 155752.6  152211.77 149759.96 146121.95 144259.4  141585.52 134307.35
 132602.65 129917.04 126992.93 125370.37 124266.9  122776.86 118474.03
 111313.02 110352.25 108733.99 108552.04 107404.34 105733.54 105008.31
 103282.38 101004.64  99937.59  97483.56  97427.84  96778.92  96712.8
  96479.51  90708.19  89949.14  81229.06  81005.76  78239.91  77798.83
  71498.49  69758.98  65200.33  64926.08  49490.75  42559.73  35673.41
  14681.4 ]
```

```
[[0.0 0.0 1.0 165349.2 136897.8 471784.1]
 [1.0 0.0 0.0 162597.7 151377.59 443898.53]
 [0.0 1.0 0.0 153441.51 101145.55 407934.54]
 [0.0 0.0 1.0 144372.41 118671.85 383199.62]
 [0.0 1.0 0.0 142107.34 91391.77 366168.42]
 [0.0 0.0 1.0 131876.9 99814.71 362861.36]
 [1.0 0.0 0.0 134615.46 147198.87 127716.82]
 [0.0 1.0 0.0 130298.13 145530.06 323876.68]
 [0.0 0.0 1.0 120542.52 148718.95 311613.29]
 [1.0 0.0 0.0 123334.88 108679.17 304981.62]
 [0.0 1.0 0.0 101913.08 110594.11 229160.95]
 [1.0 0.0 0.0 100671.96 91790.61 249744.55]
 [0.0 1.0 0.0 93863.75 127320.38 249839.44]
 [1.0 0.0 0.0 91992.39 135495.07 252664.93]
 [0.0 1.0 0.0 119943.24 156547.42 256512.92]
 [0.0 0.0 1.0 114523.61 122616.84 261776.23]
 [1.0 0.0 0.0 78013.11 121597.55 264346.06]
 [0.0 0.0 1.0 94657.16 145077.58 282574.31]
```

```
[[103015.2  103282.38]
 [132582.28 144259.4 ]
 [132447.74 146121.95]
 [ 71976.1   77798.83]
 [178537.48 191050.39]
 [116161.24 105008.31]
 [ 67851.69  81229.06]
 [ 98791.73  97483.56]
 [113969.44 110352.25]
 [167921.07 166187.94]]
```

Practical 3

Aim: Write a program to implement K-nearest Neighbors (K-NN)/SVM

Code:

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

dataset =
pd.read_csv('/content/Social_Network_Ads.csv')x =
dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

print(x)

print(y)

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=0)

print(x_train)

print(y_train)

print(x_test)

print(y_test)

from sklearn.preprocessing import
StandardScalersc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)

print(x_train)

print(x_test)

from sklearn.svm import SVC
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(x_train, y_train)

print(classifier.predict(sc.transform([[30,200000]])))
```

OUTPUT:-

```
y_pred = classifier.predict(x_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
from sklearn.metrics import confusion_matrix,
accuracy_scorecm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test,
y_pred)
```

Output:

```
[[     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]
```

```
[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 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 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]
```

```
[[     44  39000]
 [     32 120000]
 [     38  50000]
 [     32 135000]
 [     52  21000]
 [     53 104000]
 [     39  42000]
 [     38  61000]
 [     36  50000]
 [     36  63000]
 [     35  25000]
 [     35  50000]
 [     42  73000]
 [     47  49000]
```

```
[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]
```

```
[[    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]
```

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

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
 [-0.21060859 -0.19087153]
 [-0.30964085 -1.29261101]
 [-0.30964085 -0.5677824 ]
 [ 0.38358493  0.09905991]
 [ 0.8787462  -0.59677555]
 [ 2.06713324 -1.17663843]
 [ 1.07681071 -0.13288524]
```

```
[[-0.80480212  0.50496393]
 [-0.01254409 -0.5677824 ]
 [-0.30964085  0.1570462 ]
 [-0.80480212  0.27301877]
 [-0.30964085 -0.5677824 ]
 [-1.10189888 -1.43757673]
 [-0.70576986 -1.58254245]
 [-0.21060859  2.15757314]
 [-1.99318916 -0.04590581]
 [ 0.8787462  -0.77073441]
 [-0.80480212 -0.59677555]
```

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

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[66  2]
 [ 8 24]]
```

```
0.9
```

Practical 4

Aim: Write a program to implement Naïve Bayse / DT

Code:

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

dataset =
pd.read_csv('/content/Social_Network_Ads.csv')x =
dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

print(x)

print(y)

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=0)

print(x_train)

print(y_train)

print(x_test)

print(y_test)

from sklearn.preprocessing import
StandardScalersc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)

print(x_train)

print(x_test)

from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
classifier.fit(x_train, y_train)

print(classifier.predict(sc.transform([[40, 200000]])))

y_pred = classifier.predict(x_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
```

y_test.reshape(len(y_test),1)),1))from sklearn.metrics import confusion_matrix,

accuracy_score

```
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

Output:

```
[[     19  19000]
 [     35  20000]
 [     26  43000]
 [     27  57000]
 [     19  76000]
 [     27  58000]
 [     27  84000]
 [     32 150000]
 [     25  33000]
 [     35  65000]
 [     26  80000]
 [     26  52000]
```

```
[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 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 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]
```

```
[[     44  39000]
 [     32 120000]
 [     38  50000]
 [     32 135000]
 [     52  21000]
 [     53 104000]
 [     39  42000]
 [     38  61000]
 [     36  50000]
 [     36  63000]
 [     35  25000]
 [     35  50000]
 [     42  73000]
 [     47  49000]
 [     59  29000]
```

```
[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]
```

```
[[    30  87000]
 [    38  50000]
 [    35  75000]
 [    30  79000]
 [    35  50000]
 [    27  20000]
 [    31  15000]
 [    36 144000]
 [    18  68000]
 [    47  43000]
```

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

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
 [-0.21060859 -0.19087153]
 [-0.30964085 -1.29261101]
 [-0.30964085 -0.5677824 ]
```

```
[[-0.80480212  0.50496393]
 [-0.01254409 -0.5677824 ]
 [-0.30964085  0.1570462 ]
 [-0.80480212  0.27301877]
 [-0.30964085 -0.5677824 ]
 [-1.10189888 -1.43757673]
 [-0.70576986 -1.58254245]
 [-0.21060859  2.15757314]
 [-1.99318916 -0.04590581]
 [ 0.8787462  -0.77073441]
 [-0.80480212 -0.59677555]
 [-1.00286662 -0.42281668]
```

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[64  4]
 [ 3 29]]

0.93
```

Practical 5

Aim: Write a program to implement K-means clustering.

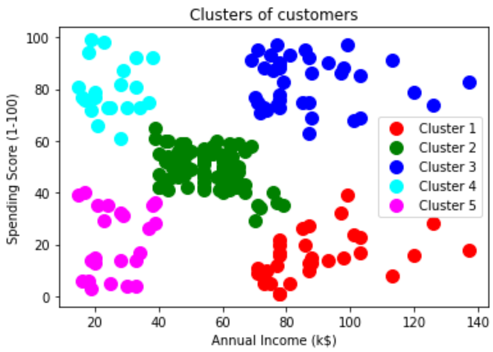
Code:

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

dataset = pd.read_csv('/content/Mall_Customers.csv')
X = dataset.iloc[:, [3,4]].values

print(X)

from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
y_hc = hc.fit_predict(X)

print(y_hc)

plt.scatter(X[y_hc==0,0], X[y_hc==0,1], s=100, c='red', label='Cluster 1')
plt.scatter(X[y_hc==1,0], X[y_hc==1,1], s=100, c='green', label='Cluster 2')
plt.scatter(X[y_hc==2,0], X[y_hc==2,1], s=100, c='blue', label='Cluster 3')
plt.scatter(X[y_hc==3,0], X[y_hc==3,1], s=100, c='cyan', label='Cluster 4')
plt.scatter(X[y_hc==4,0], X[y_hc==4,1], s=100, c='magenta', label='Cluster 5')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

Output:

```
[[ 15  39]
 [ 15  81]
 [ 16   6]
 [ 16  77]
 [ 17  40]
 [ 17  76]
 [ 18   6]
 [ 18  94]
 [ 19   3]
 [ 19  72]
```

```
[4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4
 3 4 3 4 3 4 1 4 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 2 1 2 1 2 0 2 0 2 1 2 0 2 0 2 0 2 0 2 1 2 0 2 1 2
 0 2 0 2 0 2 0 2 0 2 0 2 1 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]
```



Practical 6

Aim: Write a program to implement Hierarchical clustering.

Code:

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

dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values

from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
print(y_kmeans)

plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label
='Centroids')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
```

```
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

Output:

```
[2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2
 3 2 3 2 3 2 0 2 3 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 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 0 0 0 0 0 0 0 0 0 4 1 4 0 4 1 4 1 4 0 4 1 4 1 4 1 4 1 4 0 4 1 4 1 4
 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1
 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4]
```

Practical 7

Aim: Write a program to build ANN.

Code:

```
import numpy as np
import pandas as pd
import tensorflow as tf

dataset = pd.read_csv('Churn_Modelling.csv')
X = dataset.iloc[:, 3:-1].values
y = dataset.iloc[:, -1].values

print(X)

print(y)

from sklearn.preprocessing import
LabelEncoderle = LabelEncoder()
X[:, 2] = le.fit_transform(X[:, 2])

print(X)

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import
OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])],
remainder='passthrough')
```

```
X = np.array(ct.fit_transform(X))

print(X)

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)

from sklearn.preprocessing import
StandardScalersc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

ann = tf.keras.models.Sequential()

ann.add(tf.keras.layers.Dense(units=6, activation='relu'))

ann.add(tf.keras.layers.Dense(units=6, activation='relu'))

ann.add(tf.keras.layers.Dense(units=1,

activation='sigmoid'))

ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =

['accuracy'])ann.fit(X_train, y_train, batch_size = 32, epochs = 100)

print(ann.predict(sc.transform([[1, 0, 0, 600, 1, 40, 3, 60000, 2, 1, 1, 50000]])) >0.5)

y_pred = ann.predict(X_test)
y_pred = (y_pred > 0.5)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))

from sklearn.metrics import confusion_matrix,
accuracy_scorecm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test,
y_pred)
```

Output:

```
[[619 'France' 'Female' ... 1 1 101348.88]
 [608 'Spain' 'Female' ... 0 1 112542.58]
 [502 'France' 'Female' ... 1 0 113931.57]
 ...
 [709 'France' 'Female' ... 0 1 42085.58]
 [772 'Germany' 'Male' ... 1 0 92888.52]
 [792 'France' 'Female' ... 1 0 38190.78]]
```

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

```
[[619 'France' 0 ... 1 1 101348.88]
 [608 'Spain' 0 ... 0 1 112542.58]
 [502 'France' 0 ... 1 0 113931.57]
 ...
 [709 'France' 0 ... 0 1 42085.58]
 [772 'Germany' 1 ... 1 0 92888.52]
 [792 'France' 0 ... 1 0 38190.78]]
```

```
[[1.0 0.0 0.0 ... 1 1 101348.88]
 [0.0 0.0 1.0 ... 0 1 112542.58]
 [1.0 0.0 0.0 ... 1 0 113931.57]
 ...
 [1.0 0.0 0.0 ... 0 1 42085.58]
 [0.0 1.0 0.0 ... 1 0 92888.52]
 [1.0 0.0 0.0 ... 1 0 38190.78]]
```

```
Epoch 1/100
250/250 [==============================] - 1s 1ms/step - loss: 0.5750 - accuracy: 0.7490
Epoch 2/100
250/250 [==============================] - 0s 1ms/step - loss: 0.4712 - accuracy: 0.7960
Epoch 3/100
250/250 [==============================] - 0s 2ms/step - loss: 0.4428 - accuracy: 0.7986
Epoch 4/100
250/250 [==============================] - 0s 2ms/step - loss: 0.4296 - accuracy: 0.8075
Epoch 5/100
250/250 [==============================] - 0s 2ms/step - loss: 0.4212 - accuracy: 0.8149
Epoch 6/100
250/250 [==============================] - 0s 2ms/step - loss: 0.4138 - accuracy: 0.8220
Epoch 7/100
```

```
y_pred = ann.predict(X_test)
y_pred = (y_pred > 0.5)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

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

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[1499   96]
 [ 186  219]]
```

```
0.859
```

Practical 8

Aim: Write a program to build CNN.

Code:

```
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2,
zoom_range=0.2,horizontal_flip=True)
training_set =
train_datagen.flow_from_directory('/content/drive/MyDrive/small_dataset/training_set',
target_size=(64,64), batch_size=32, class_mode='binary')

train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2,
zoom_range=0.2,horizontal_flip=True)
test_set =
train_datagen.flow_from_directory('/content/drive/MyDrive/small_dataset/test_set',
target_size=(64,64), batch_size=32, class_mode='binary')

cnn = tf.keras.models.Sequential()

cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu',
input_shape=[64,64,3]))

cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

cnn.add(tf.keras.layers.Flatten())
```

```
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))

cnn.add(tf.keras.layers.Dense(units=1,

activation='sigmoid'))
```

cnn.compile(optimizer='adam', loss='binary_crossentropy',

metrics=['accuracy'])cnn.fit(x=training_set, validation_data=test_set,

epochs=25)

import numpy as np
from keras.preprocessing import image
test_image=image.load_img('/content/drive/MyDrive/small_dataset/single_prediction/cat_or_dog_
1
.jpg', target_size=(64,64))
test_image=image.img_to_array(test_image)
test_image=np.expand_dims(test_image,
axis=0)result=cnn.predict(test_image)
training_set.class_indices
if result[0][0]==1:
prediction='dog'
else:
prediction='cat'

print(prediction)

Output:

```
Epoch 1/25
1/1 [==============================] - 1s 897ms/step - loss: 0.7015 - accuracy: 0.5000 - val_loss: 0.7160 - val_accu
racy: 0.5000
Epoch 2/25
1/1 [==============================] - 0s 227ms/step - loss: 0.6286 - accuracy: 0.9000 - val_loss: 0.7793 - val_accu
racy: 0.5000
Epoch 3/25
1/1 [==============================] - 0s 224ms/step - loss: 0.6135 - accuracy: 0.5000 - val_loss: 0.7770 - val_accu
racy: 0.5000
Epoch 4/25
1/1 [==============================] - 0s 230ms/step - loss: 0.6256 - accuracy: 0.4000 - val_loss: 0.7575 - val_accu
racy: 0.4000
Epoch 5/25
1/1 [==============================] - 0s 211ms/step - loss: 0.5565 - accuracy: 0.9000 - val_loss: 0.7845 - val_accu
racy: 0.4000
```

```
print(prediction)
```

```
dog
```