Machine leraning & Deep learning Journal

M.Sc Part II Computer Science

Mithun Parab 509

October 11, 2023

R.J. College of Arts, Science & Commerce
Machine learning & Deep learning
Seat number: 509

RAMNIRANJAN JHUNJHUNWALA COLLEGE

Ghatkopar (W), Mumbai-400 086

CERTIFICATE

M.Sc Part II
Computer Science
2023-2024

This is to certify that **Mithun Sahdev Parab** of M.Sc Prat II (Sem-III) Computer Science, Seat No **509** of satisfactorily completed the practicals of **MACHINE LEARNING AND DEEP LEARNING(PAPER I)** during the academic year **2023 - 2024** as specified by the **MUMBAI UNIVERSITY**.

No. of Experiments completed **9** out of **9**

Sign of Incharge:
Date: October 11, 2023 **Seat Number: 509**

Sign of Examiner:
Date: October 11, 2023 **Course Co-ordinator**

Contents

Link for GitHub

1 Practical 01: Implement Simple Linear Regression

1.1 Importing the libraries

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

1.2 Importing the dataset

[3]:
```
dataset = pd.read_csv('/content/drive/MyDrive/MSC CS/SEM 3/1. Machine Learning &␣
↪Deep Learning/Practicals/1. Simple Linear Regression/Salary_Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

[4]:
```
print(X)
```

```
[[ 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]]
```

```
[5]: print(y)
```

```
[ 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.]
```

1.3 Splitting the dataset into the Training set and Test set

```
[6]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3,␣
 ↪random_state = 0)
```

```
[7]: print(X_train)
```

```
[[ 2.9]
 [ 5.1]
 [ 3.2]
 [ 4.5]
 [ 8.2]
 [ 6.8]
 [ 1.3]
 [10.5]
 [ 3. ]
 [ 2.2]
 [ 5.9]
 [ 6. ]
 [ 3.7]
 [ 3.2]
 [ 9. ]
 [ 2. ]
 [ 1.1]
 [ 7.1]
 [ 4.9]
 [ 4. ]]
```

```
[8]: print(X_test)
```

```
[[ 1.5]
 [10.3]
 [ 4.1]
 [ 3.9]
 [ 9.5]
 [ 8.7]
 [ 9.6]
 [ 4. ]
 [ 5.3]
 [ 7.9]]
```

[9]:
```python
print(y_train)
```

```
[ 56642.  66029.  64445.  61111. 113812.  91738.  46205. 121872.  60150.
  39891.  81363.  93940.  57189.  54445. 105582.  43525.  39343.  98273.
  67938.  56957.]
```

[10]:
```python
print(y_test)
```

```
[ 37731. 122391.  57081.  63218. 116969. 109431. 112635.  55794.  83088.
 101302.]
```

1.4 Training the Simple Linear Regression model on the Training set

[11]:
```python
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

[11]:
```
LinearRegression()
```

1.5 Predicting the Test set results

[12]:
```python
y_pred = regressor.predict(X_test)
```

1.6 Visualising the Training set results

[13]:
```python
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

1.7 Visualising the Test set results

[14]:
```
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_test, regressor.predict(X_test), color = 'blue')
plt.title('Salary vs Experience (Test set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

2 Practical 02: Implement Multiple Linear Regression

2.1 Importing the libraries

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

2.2 Importing the dataset

[3]:
```
dataset = pd.read_csv('/content/drive/MyDrive/MSC CS/SEM 3/1. Machine Learning &␣
↪Deep Learning/Practicals/2. Multiple Linear Regression/50_Startups-2.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

[4]:
```
print(X)
```

```
[[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']
[67532.53 105751.03 304768.73 'Florida']
[77044.01 99281.34 140574.81 'New York']
[64664.71 139553.16 137962.62 'California']
[75328.87 144135.98 134050.07 'Florida']
[72107.6 127864.55 353183.81 'New York']
[66051.52 182645.56 118148.2 'Florida']
[65605.48 153032.06 107138.38 'New York']
[61994.48 115641.28 91131.24 'Florida']
[61136.38 152701.92 88218.23 'New York']
[63408.86 129219.61 46085.25 'California']
[55493.95 103057.49 214634.81 'Florida']
[46426.07 157693.92 210797.67 'California']
[46014.02 85047.44 205517.64 'New York']
[28663.76 127056.21 201126.82 'Florida']
[44069.95 51283.14 197029.42 'California']
[20229.59 65947.93 185265.1 'New York']
[38558.51 82982.09 174999.3 'California']
[28754.33 118546.05 172795.67 'California']
[27892.92 84710.77 164470.71 'Florida']
[23640.93 96189.63 148001.11 'California']
[15505.73 127382.3 35534.17 'New York']
[22177.74 154806.14 28334.72 'California']
[1000.23 124153.04 1903.93 'New York']
[1315.46 115816.21 297114.46 'Florida']
[0.0 135426.92 0.0 'California']
[542.05 51743.15 0.0 'New York']
[0.0 116983.8 45173.06 'California']]
```

[5]:
```
print(y)
```

```
[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 ]
```

2.3 Encoding categorical data

[6]:
```
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))
```

[7]:
```
print(X)
```

```
[[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]
 [0.0 1.0 0.0 91749.16 114175.79 294919.57]
 [0.0 0.0 1.0 86419.7 153514.11 0.0]
 [1.0 0.0 0.0 76253.86 113867.3 298664.47]
 [0.0 0.0 1.0 78389.47 153773.43 299737.29]
 [0.0 1.0 0.0 73994.56 122782.75 303319.26]
 [0.0 1.0 0.0 67532.53 105751.03 304768.73]
 [0.0 0.0 1.0 77044.01 99281.34 140574.81]
 [1.0 0.0 0.0 64664.71 139553.16 137962.62]
 [0.0 1.0 0.0 75328.87 144135.98 134050.07]
```

```
 [0.0 0.0 1.0 72107.6 127864.55 353183.81]
 [0.0 1.0 0.0 66051.52 182645.56 118148.2]
 [0.0 0.0 1.0 65605.48 153032.06 107138.38]
 [0.0 1.0 0.0 61994.48 115641.28 91131.24]
 [0.0 0.0 1.0 61136.38 152701.92 88218.23]
 [1.0 0.0 0.0 63408.86 129219.61 46085.25]
 [0.0 1.0 0.0 55493.95 103057.49 214634.81]
 [1.0 0.0 0.0 46426.07 157693.92 210797.67]
 [0.0 0.0 1.0 46014.02 85047.44 205517.64]
 [0.0 1.0 0.0 28663.76 127056.21 201126.82]
 [1.0 0.0 0.0 44069.95 51283.14 197029.42]
 [0.0 0.0 1.0 20229.59 65947.93 185265.1]
 [1.0 0.0 0.0 38558.51 82982.09 174999.3]
 [1.0 0.0 0.0 28754.33 118546.05 172795.67]
 [0.0 1.0 0.0 27892.92 84710.77 164470.71]
 [1.0 0.0 0.0 23640.93 96189.63 148001.11]
 [0.0 0.0 1.0 15505.73 127382.3 35534.17]
 [1.0 0.0 0.0 22177.74 154806.14 28334.72]
 [0.0 0.0 1.0 1000.23 124153.04 1903.93]
 [0.0 1.0 0.0 1315.46 115816.21 297114.46]
 [1.0 0.0 0.0 0.0 135426.92 0.0]
 [0.0 0.0 1.0 542.05 51743.15 0.0]
 [1.0 0.0 0.0 0.0 116983.8 45173.06]]
```

2.4 Splitting the dataset into the Training set and Test set

[8]:
```python
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)
```

2.5 Training the Multiple Linear Regression model on the Training set

[9]:
```python
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

[9]:
```
LinearRegression()
```

2.6 Predicting the Test set results

[10]:
```python
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))
```

```
[[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]]
```

3 Practical 03: Implement Logistic Regeression for classification of handwritten digits (MNIST dataset)

[2]:
```
import pandas as pd
```

#Data Collection

[3]:
```
iris_data = pd.read_csv('/content/drive/MyDrive/MSC CS/SEM 3/1. Machine Learning␣
↪& Deep Learning/Practicals/3. Logistic Regression/Iris.csv')
```

[4]:
```
#print(iris_data)
#iris_data.sample(5)
iris_data.head()
```

[4]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

#Data Cleaning

[5]:
```
iris_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

#Label Encoding

[6]:
```
# Iris-setosa = 0 , Iris-versicolor = 1, Iris-virginica = 2

from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()

iris_data['Species'] = encoder.fit_transform(iris_data['Species'])
```

[7]:
```
iris_data.head(150)
```

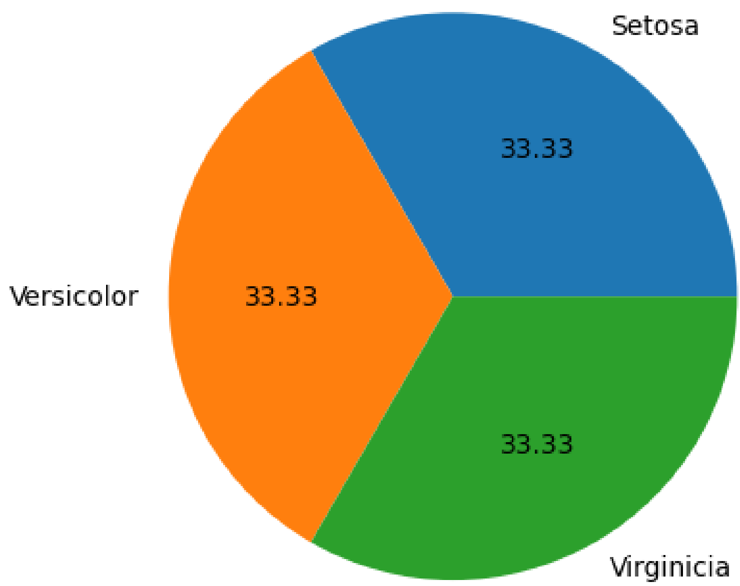
[7]:
```
      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species
0      1            5.1           3.5            1.4           0.2        0
1      2            4.9           3.0            1.4           0.2        0
2      3            4.7           3.2            1.3           0.2        0
3      4            4.6           3.1            1.5           0.2        0
4      5            5.0           3.6            1.4           0.2        0
..   ...            ...           ...            ...           ...      ...
145  146            6.7           3.0            5.2           2.3        2
146  147            6.3           2.5            5.0           1.9        2
147  148            6.5           3.0            5.2           2.0        2
148  149            6.2           3.4            5.4           2.3        2
149  150            5.9           3.0            5.1           1.8        2

[150 rows x 6 columns]
```

#**Data Analysis**

[8]:
```
import matplotlib.pyplot as plt

plt.pie(iris_data['Species'].value_counts(),labels = ['Setosa', 'Versicolor',␣
 ↪'Virginicia'], autopct = '%0.2f')
plt.show()
```



#Create Dependent & Independent Variable

[9]:
```
x = iris_data.drop('Species',axis = 1)
y = iris_data['Species']
```

[10]:
```
print(x)
```

```
      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
0      1            5.1           3.5            1.4           0.2
1      2            4.9           3.0            1.4           0.2
2      3            4.7           3.2            1.3           0.2
3      4            4.6           3.1            1.5           0.2
4      5            5.0           3.6            1.4           0.2
..   ...            ...           ...            ...           ...
145  146            6.7           3.0            5.2           2.3
146  147            6.3           2.5            5.0           1.9
147  148            6.5           3.0            5.2           2.0
148  149            6.2           3.4            5.4           2.3
149  150            5.9           3.0            5.1           1.8

[150 rows x 5 columns]
```

[11]:
```
print(y)
```

```
0      0
```

```
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: Species, Length: 150, dtype: int64
```

#Split data into Train and Test dataset

[12]:
```
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 = 2)
```

#Train the model

[13]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(max_iter = 1000)
model.fit(x_train,y_train)
LogisticRegression(max_iter=1000)
```

[13]:
```
LogisticRegression(max_iter=1000)
```

#Predict train data

[14]:
```
pred_train = model.predict(x_train)
```

#Check accuracy on train data

[15]:
```
from sklearn.metrics import confusion_matrix,accuracy_score
accuracy_score(y_train,pred_train)
```

[15]:
```
1.0
```

#Predict Test data

[16]:
```
pred_test = model.predict(x_test)
```

#Check accuracy and print Confusion matrix

[17]:
```
accuracy_score(y_test, pred_test)
```

[17]:
```
1.0
```

[20]:
```
confusion_matrix(y_test,pred_test)
```

```
[20]: array([[14,  0,  0],
             [ 0,  8,  0],
             [ 0,  0,  8]])
```

4 Practical 04: Implement SVM classifier

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

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

```
[ ]: 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)
```

```
[[    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]
 [    49  65000]
 [    45 131000]
 [    31  89000]
 [    46  82000]
 [    47  51000]
 [    26  15000]
 [    60 102000]
 [    38 112000]
 [    40 107000]
 [    42  53000]
 [    35  59000]
 [    48  41000]
```

```
[     48 134000]
[     38 113000]
[     29 148000]
[     26  15000]
[     60  42000]
[     24  19000]
[     42 149000]
[     46  96000]
[     28  59000]
[     39  96000]
[     28  89000]
[     41  72000]
[     45  26000]
[     33  69000]
[     20  82000]
[     31  74000]
[     42  80000]
[     35  72000]
[     33 149000]
[     40  71000]
[     51 146000]
[     46  79000]
[     35  75000]
[     38  51000]
[     36  75000]
[     37  78000]
[     38  61000]
[     60 108000]
[     20  82000]
[     57  74000]
[     42  65000]
[     26  80000]
[     46 117000]
[     35  61000]
[     21  68000]
[     28  44000]
[     41  87000]
[     37  33000]
[     27  90000]
[     39  42000]
[     28 123000]
[     31 118000]
[     25  87000]
[     35  71000]
[     37  70000]
[     35  39000]
[     47  23000]
[     35 147000]
```

```
[     48 138000]
[     26  86000]
[     25  79000]
[     52 138000]
[     51  23000]
[     35  60000]
[     33 113000]
[     30 107000]
[     48  33000]
[     41  80000]
[     48  96000]
[     31  18000]
[     31  71000]
[     43 129000]
[     59  76000]
[     18  44000]
[     36 118000]
[     42  90000]
[     47  30000]
[     26  43000]
[     40  78000]
[     46  59000]
[     59  42000]
[     46  74000]
[     35  91000]
[     28  59000]
[     40  57000]
[     59 143000]
[     57  26000]
[     52  38000]
[     47 113000]
[     53 143000]
[     35  27000]
[     58 101000]
[     45  45000]
[     23  82000]
[     46  23000]
[     42  65000]
[     28  84000]
[     38  59000]
[     26  84000]
[     29  28000]
[     37  71000]
[     22  55000]
[     48  35000]
[     49  28000]
[     38  65000]
[     27  17000]
```

```
[     46  28000]
[     48 141000]
[     26  17000]
[     35  97000]
[     39  59000]
[     24  27000]
[     32  18000]
[     46  88000]
[     35  58000]
[     56  60000]
[     47  34000]
[     40  72000]
[     32 100000]
[     19  21000]
[     25  90000]
[     35  88000]
[     28  32000]
[     50  20000]
[     40  59000]
[     50  44000]
[     35  72000]
[     40 142000]
[     46  32000]
[     39  71000]
[     20  74000]
[     29  75000]
[     31  76000]
[     47  25000]
[     40  61000]
[     34 112000]
[     38  80000]
[     42  75000]
[     47  47000]
[     39  75000]
[     19  25000]
[     37  80000]
[     36  60000]
[     41  52000]
[     36 125000]
[     48  29000]
[     36 126000]
[     51 134000]
[     27  57000]
[     38  71000]
[     39  61000]
[     22  27000]
[     33  60000]
[     48  74000]
```

```
 [     58   23000]
 [     53   72000]
 [     32  117000]
 [     54   70000]
 [     30   80000]
 [     58   95000]
 [     26   52000]
 [     45   79000]
 [     24   55000]
 [     40   75000]
 [     33   28000]
 [     44  139000]
 [     22   18000]
 [     33   51000]
 [     43  133000]
 [     24   32000]
 [     46   22000]
 [     35   55000]
 [     54  104000]
 [     48  119000]
 [     35   53000]
 [     37  144000]
 [     23   66000]
 [     37  137000]
 [     31   58000]
 [     33   41000]
 [     45   22000]
 [     30   15000]
 [     19   19000]
 [     49   74000]
 [     39  122000]
 [     35   73000]
 [     39   71000]
 [     24   23000]
 [     41   72000]
 [     29   83000]
 [     54   26000]
 [     35   44000]
 [     37   75000]
 [     29   47000]
 [     31   68000]
 [     42   54000]
 [     30  135000]
 [     52  114000]
 [     50   36000]
 [     56  133000]
 [     29   61000]
 [     30   89000]
```

```
[    26  16000]
[    33  31000]
[    41  72000]
[    36  33000]
[    55 125000]
[    48 131000]
[    41  71000]
[    30  62000]
[    37  72000]
[    41  63000]
[    58  47000]
[    30 116000]
[    20  49000]
[    37  74000]
[    41  59000]
[    49  89000]
[    28  79000]
[    53  82000]
[    40  57000]
[    60  34000]
[    35 108000]
[    21  72000]
[    38  71000]
[    39 106000]
[    37  57000]
[    26  72000]
[    35  23000]
[    54 108000]
[    30  17000]
[    39 134000]
[    29  43000]
[    33  43000]
[    35  38000]
[    41  45000]
[    41  72000]
[    39 134000]
[    27 137000]
[    21  16000]
[    26  32000]
[    31  66000]
[    39  73000]
[    41  79000]
[    47  50000]
[    41  30000]
[    37  93000]
[    60  46000]
[    25  22000]
[    28  37000]
```

```
 [    38  55000]
 [    36  54000]
 [    20  36000]
 [    56 104000]
 [    40  57000]
 [    42 108000]
 [    20  23000]
 [    40  65000]
 [    47  20000]
 [    18  86000]
 [    35  79000]
 [    57  33000]
 [    34  72000]
 [    49  39000]
 [    27  31000]
 [    19  70000]
 [    39  79000]
 [    26  81000]
 [    25  80000]
 [    28  85000]
 [    55  39000]
 [    50  88000]
 [    49  88000]
 [    52 150000]
 [    35  65000]
 [    42  54000]
 [    34  43000]
 [    37  52000]
 [    48  30000]
 [    29  43000]
 [    36  52000]
 [    27  54000]
 [    26 118000]]
```

[ ]:
```
print(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]
 [     21   88000]
 [     37   79000]
 [     57   60000]
 [     37   53000]
 [     24   58000]
 [     18   52000]
 [     22   81000]
 [     34   43000]
 [     31   34000]
 [     49   36000]
 [     27   88000]
 [     41   52000]
 [     27   84000]
 [     35   20000]
 [     43  112000]
 [     27   58000]
 [     37   80000]
 [     52   90000]
 [     26   30000]
 [     49   86000]
 [     57  122000]
 [     34   25000]
 [     35   57000]
 [     34  115000]
 [     59   88000]
 [     45   32000]
 [     29   83000]
 [     26   80000]
 [     49   28000]
 [     23   20000]
```

```
[    32  18000]
[    60  42000]
[    19  76000]
[    36  99000]
[    19  26000]
[    60  83000]
[    24  89000]
[    27  58000]
[    40  47000]
[    42  70000]
[    32 150000]
[    35  77000]
[    22  63000]
[    45  22000]
[    27  89000]
[    18  82000]
[    42  79000]
[    40  60000]
[    53  34000]
[    47 107000]
[    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]]
```

[ ]:
```
print(y_train)
```

```
[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]
```

[ ]:
```python
print(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 1 0 0 0 1 1 0 1 1 0 0 1 0 0 0 1 0 1 1 1]
```

[ ]:
```python
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)
```

[ ]:
```python
print(x_train)
```

```
[[ 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]
 [ 0.68068169  1.78066227]
 [-0.70576986  0.56295021]
 [ 0.77971394  0.35999821]
 [ 0.8787462  -0.53878926]
 [-1.20093113 -1.58254245]
 [ 2.1661655   0.93986109]
 [-0.01254409  1.22979253]
 [ 0.18552042  1.08482681]
 [ 0.38358493 -0.48080297]
 [-0.30964085 -0.30684411]
 [ 0.97777845 -0.8287207 ]
 [ 0.97777845  1.8676417 ]
 [-0.01254409  1.25878567]
 [-0.90383437  2.27354572]
 [-1.20093113 -1.58254245]
```

```
[ 2.1661655  -0.79972756]
[-1.39899564 -1.46656987]
[ 0.38358493  2.30253886]
[ 0.77971394  0.76590222]
[-1.00286662 -0.30684411]
[ 0.08648817  0.76590222]
[-1.00286662  0.56295021]
[ 0.28455268  0.07006676]
[ 0.68068169 -1.26361786]
[-0.50770535 -0.01691267]
[-1.79512465  0.35999821]
[-0.70576986  0.12805305]
[ 0.38358493  0.30201192]
[-0.30964085  0.07006676]
[-0.50770535  2.30253886]
[ 0.18552042  0.04107362]
[ 1.27487521  2.21555943]
[ 0.77971394  0.27301877]
[-0.30964085  0.1570462 ]
[-0.01254409 -0.53878926]
[-0.21060859  0.1570462 ]
[-0.11157634  0.24402563]
[-0.01254409 -0.24885782]
[ 2.1661655   1.11381995]
[-1.79512465  0.35999821]
[ 1.86906873  0.12805305]
[ 0.38358493 -0.13288524]
[-1.20093113  0.30201192]
[ 0.77971394  1.37475825]
[-0.30964085 -0.24885782]
[-1.6960924  -0.04590581]
[-1.00286662 -0.74174127]
[ 0.28455268  0.50496393]
[-0.11157634 -1.06066585]
[-1.10189888  0.59194336]
[ 0.08648817 -0.79972756]
[-1.00286662  1.54871711]
[-0.70576986  1.40375139]
[-1.29996338  0.50496393]
[-0.30964085  0.04107362]
[-0.11157634  0.01208048]
[-0.30964085 -0.88670699]
[ 0.8787462  -1.3505973 ]
[-0.30964085  2.24455257]
[ 0.97777845  1.98361427]
[-1.20093113  0.47597078]
[-1.29996338  0.27301877]
[ 1.37390747  1.98361427]
```

```
[ 1.27487521 -1.3505973 ]
[-0.30964085 -0.27785096]
[-0.50770535  1.25878567]
[-0.80480212  1.08482681]
[ 0.97777845 -1.06066585]
[ 0.28455268  0.30201192]
[ 0.97777845  0.76590222]
[-0.70576986 -1.49556302]
[-0.70576986  0.04107362]
[ 0.48261718  1.72267598]
[ 2.06713324  0.18603934]
[-1.99318916 -0.74174127]
[-0.21060859  1.40375139]
[ 0.38358493  0.59194336]
[ 0.8787462  -1.14764529]
[-1.20093113 -0.77073441]
[ 0.18552042  0.24402563]
[ 0.77971394 -0.30684411]
[ 2.06713324 -0.79972756]
[ 0.77971394  0.12805305]
[-0.30964085  0.6209365 ]
[-1.00286662 -0.30684411]
[ 0.18552042 -0.3648304 ]
[ 2.06713324  2.12857999]
[ 1.86906873 -1.26361786]
[ 1.37390747 -0.91570013]
[ 0.8787462   1.25878567]
[ 1.47293972  2.12857999]
[-0.30964085 -1.23462472]
[ 1.96810099  0.91086794]
[ 0.68068169 -0.71274813]
[-1.49802789  0.35999821]
[ 0.77971394 -1.3505973 ]
[ 0.38358493 -0.13288524]
[-1.00286662  0.41798449]
[-0.01254409 -0.30684411]
[-1.20093113  0.41798449]
[-0.90383437 -1.20563157]
[-0.11157634  0.04107362]
[-1.59706014 -0.42281668]
[ 0.97777845 -1.00267957]
[ 1.07681071 -1.20563157]
[-0.01254409 -0.13288524]
[-1.10189888 -1.52455616]
[ 0.77971394 -1.20563157]
[ 0.97777845  2.07059371]
[-1.20093113 -1.52455616]
[-0.30964085  0.79489537]
```

```
[ 0.08648817 -0.30684411]
[-1.39899564 -1.23462472]
[-0.60673761 -1.49556302]
[ 0.77971394  0.53395707]
[-0.30964085 -0.33583725]
[ 1.77003648 -0.27785096]
[ 0.8787462  -1.03167271]
[ 0.18552042  0.07006676]
[-0.60673761  0.8818748 ]
[-1.89415691 -1.40858358]
[-1.29996338  0.59194336]
[-0.30964085  0.53395707]
[-1.00286662 -1.089659  ]
[ 1.17584296 -1.43757673]
[ 0.18552042 -0.30684411]
[ 1.17584296 -0.74174127]
[-0.30964085  0.07006676]
[ 0.18552042  2.09958685]
[ 0.77971394 -1.089659  ]
[ 0.08648817  0.04107362]
[-1.79512465  0.12805305]
[-0.90383437  0.1570462 ]
[-0.70576986  0.18603934]
[ 0.8787462  -1.29261101]
[ 0.18552042 -0.24885782]
[-0.4086731   1.22979253]
[-0.01254409  0.30201192]
[ 0.38358493  0.1570462 ]
[ 0.8787462  -0.65476184]
[ 0.08648817  0.1570462 ]
[-1.89415691 -1.29261101]
[-0.11157634  0.30201192]
[-0.21060859 -0.27785096]
[ 0.28455268 -0.50979612]
[-0.21060859  1.6067034 ]
[ 0.97777845 -1.17663843]
[-0.21060859  1.63569655]
[ 1.27487521  1.8676417 ]
[-1.10189888 -0.3648304 ]
[-0.01254409  0.04107362]
[ 0.08648817 -0.24885782]
[-1.59706014 -1.23462472]
[-0.50770535 -0.27785096]
[ 0.97777845  0.12805305]
[ 1.96810099 -1.3505973 ]
[ 1.47293972  0.07006676]
[-0.60673761  1.37475825]
[ 1.57197197  0.01208048]
```

```
[-0.80480212  0.30201192]
[ 1.96810099  0.73690908]
[-1.20093113 -0.50979612]
[ 0.68068169  0.27301877]
[-1.39899564 -0.42281668]
[ 0.18552042  0.1570462 ]
[-0.50770535 -1.20563157]
[ 0.58164944  2.01260742]
[-1.59706014 -1.49556302]
[-0.50770535 -0.53878926]
[ 0.48261718  1.83864855]
[-1.39899564 -1.089659  ]
[ 0.77971394 -1.37959044]
[-0.30964085 -0.42281668]
[ 1.57197197  0.99784738]
[ 0.97777845  1.43274454]
[-0.30964085 -0.48080297]
[-0.11157634  2.15757314]
[-1.49802789 -0.1038921 ]
[-0.11157634  1.95462113]
[-0.70576986 -0.33583725]
[-0.50770535 -0.8287207 ]
[ 0.68068169 -1.37959044]
[-0.80480212 -1.58254245]
[-1.89415691 -1.46656987]
[ 1.07681071  0.12805305]
[ 0.08648817  1.51972397]
[-0.30964085  0.09905991]
[ 0.08648817  0.04107362]
[-1.39899564 -1.3505973 ]
[ 0.28455268  0.07006676]
[-0.90383437  0.38899135]
[ 1.57197197 -1.26361786]
[-0.30964085 -0.74174127]
[-0.11157634  0.1570462 ]
[-0.90383437 -0.65476184]
[-0.70576986 -0.04590581]
[ 0.38358493 -0.45180983]
[-0.80480212  1.89663484]
[ 1.37390747  1.28777882]
[ 1.17584296 -0.97368642]
[ 1.77003648  1.83864855]
[-0.90383437 -0.24885782]
[-0.80480212  0.56295021]
[-1.20093113 -1.5535493 ]
[-0.50770535 -1.11865214]
[ 0.28455268  0.07006676]
[-0.21060859 -1.06066585]
```

```
[ 1.67100423  1.6067034 ]
[ 0.97777845  1.78066227]
[ 0.28455268  0.04107362]
[-0.80480212 -0.21986468]
[-0.11157634  0.07006676]
[ 0.28455268 -0.19087153]
[ 1.96810099 -0.65476184]
[-0.80480212  1.3457651 ]
[-1.79512465 -0.59677555]
[-0.11157634  0.12805305]
[ 0.28455268 -0.30684411]
[ 1.07681071  0.56295021]
[-1.00286662  0.27301877]
[ 1.47293972  0.35999821]
[ 0.18552042 -0.3648304 ]
[ 2.1661655  -1.03167271]
[-0.30964085  1.11381995]
[-1.6960924   0.07006676]
[-0.01254409  0.04107362]
[ 0.08648817  1.05583366]
[-0.11157634 -0.3648304 ]
[-1.20093113  0.07006676]
[-0.30964085 -1.3505973 ]
[ 1.57197197  1.11381995]
[-0.80480212 -1.52455616]
[ 0.08648817  1.8676417 ]
[-0.90383437 -0.77073441]
[-0.50770535 -0.77073441]
[-0.30964085 -0.91570013]
[ 0.28455268 -0.71274813]
[ 0.28455268  0.07006676]
[ 0.08648817  1.8676417 ]
[-1.10189888  1.95462113]
[-1.6960924  -1.5535493 ]
[-1.20093113 -1.089659  ]
[-0.70576986 -0.1038921 ]
[ 0.08648817  0.09905991]
[ 0.28455268  0.27301877]
[ 0.8787462  -0.5677824 ]
[ 0.28455268 -1.14764529]
[-0.11157634  0.67892279]
[ 2.1661655  -0.68375498]
[-1.29996338 -1.37959044]
[-1.00286662 -0.94469328]
[-0.01254409 -0.42281668]
[-0.21060859 -0.45180983]
[-1.79512465 -0.97368642]
[ 1.77003648  0.99784738]
```

```
 [ 0.18552042 -0.3648304 ]
 [ 0.38358493  1.11381995]
 [-1.79512465 -1.3505973 ]
 [ 0.18552042 -0.13288524]
 [ 0.8787462  -1.43757673]
 [-1.99318916  0.47597078]
 [-0.30964085  0.27301877]
 [ 1.86906873 -1.06066585]
 [-0.4086731   0.07006676]
 [ 1.07681071 -0.88670699]
 [-1.10189888 -1.11865214]
 [-1.89415691  0.01208048]
 [ 0.08648817  0.27301877]
 [-1.20093113  0.33100506]
 [-1.29996338  0.30201192]
 [-1.00286662  0.44697764]
 [ 1.67100423 -0.88670699]
 [ 1.17584296  0.53395707]
 [ 1.07681071  0.53395707]
 [ 1.37390747  2.331532  ]
 [-0.30964085 -0.13288524]
 [ 0.38358493 -0.45180983]
 [-0.4086731  -0.77073441]
 [-0.11157634 -0.50979612]
 [ 0.97777845 -1.14764529]
 [-0.90383437 -0.77073441]
 [-0.21060859 -0.50979612]
 [-1.10189888 -0.45180983]
 [-1.20093113  1.40375139]]
```

[ ]:
```
print(x_test)
```

```
[[-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]
 [-0.11157634 -0.42281668]
 [ 0.08648817  0.21503249]
 [-1.79512465  0.47597078]
 [-0.60673761  1.37475825]
```

```
[-0.11157634  0.21503249]
[-1.89415691  0.44697764]
[ 1.67100423  1.75166912]
[-0.30964085 -1.37959044]
[-0.30964085 -0.65476184]
[ 0.8787462   2.15757314]
[ 0.28455268 -0.53878926]
[ 0.8787462   1.02684052]
[-1.49802789 -1.20563157]
[ 1.07681071  2.07059371]
[-1.00286662  0.50496393]
[-0.90383437  0.30201192]
[-0.11157634 -0.21986468]
[-0.60673761  0.47597078]
[-1.6960924   0.53395707]
[-0.11157634  0.27301877]
[ 1.86906873 -0.27785096]
[-0.11157634 -0.48080297]
[-1.39899564 -0.33583725]
[-1.99318916 -0.50979612]
[-1.59706014  0.33100506]
[-0.4086731  -0.77073441]
[-0.70576986 -1.03167271]
[ 1.07681071 -0.97368642]
[-1.10189888  0.53395707]
[ 0.28455268 -0.50979612]
[-1.10189888  0.41798449]
[-0.30964085 -1.43757673]
[ 0.48261718  1.22979253]
[-1.10189888 -0.33583725]
[-0.11157634  0.30201192]
[ 1.37390747  0.59194336]
[-1.20093113 -1.14764529]
[ 1.07681071  0.47597078]
[ 1.86906873  1.51972397]
[-0.4086731  -1.29261101]
[-0.30964085 -0.3648304 ]
[-0.4086731   1.31677196]
[ 2.06713324  0.53395707]
[ 0.68068169 -1.089659  ]
[-0.90383437  0.38899135]
[-1.20093113  0.30201192]
[ 1.07681071 -1.20563157]
[-1.49802789 -1.43757673]
[-0.60673761 -1.49556302]
[ 2.1661655  -0.79972756]
[-1.89415691  0.18603934]
[-0.21060859  0.85288166]
```

```
[-1.89415691 -1.26361786]
[ 2.1661655   0.38899135]
[-1.39899564  0.56295021]
[-1.10189888 -0.33583725]
[ 0.18552042 -0.65476184]
[ 0.38358493  0.01208048]
[-0.60673761  2.331532  ]
[-0.30964085  0.21503249]
[-1.59706014 -0.19087153]
[ 0.68068169 -1.37959044]
[-1.10189888  0.56295021]
[-1.99318916  0.35999821]
[ 0.38358493  0.27301877]
[ 0.18552042 -0.27785096]
[ 1.47293972 -1.03167271]
[ 0.8787462   1.08482681]
[ 1.96810099  2.15757314]
[ 2.06713324  0.38899135]
[-1.39899564 -0.42281668]
[-1.20093113 -1.00267957]
[ 1.96810099 -0.91570013]
[ 0.38358493  0.30201192]
[ 0.18552042  0.1570462 ]
[ 2.06713324  1.75166912]
[ 0.77971394 -0.8287207 ]
[ 0.28455268 -0.27785096]
[ 0.38358493 -0.16187839]
[-0.11157634  2.21555943]
[-1.49802789 -0.62576869]
[-1.29996338 -1.06066585]
[-1.39899564  0.41798449]
[-1.10189888  0.76590222]
[-1.49802789 -0.19087153]
[ 0.97777845 -1.06066585]
[ 0.97777845  0.59194336]
[ 0.38358493  0.99784738]]
```

[ ]:
```python
from sklearn.svm import SVC
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(x_train, y_train)
```

[ ]:
```
SVC(kernel='linear', random_state=0)
```

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

```
[1]
```

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

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

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

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

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

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

[ ]:
```
0.9
```

5 Practical 05: Implement KNN classifier

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

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

[ ]:
```
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)
```

```
[[    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]
[     49   65000]
[     45  131000]
[     31   89000]
[     46   82000]
[     47   51000]
[     26   15000]
[     60  102000]
[     38  112000]
[     40  107000]
[     42   53000]
[     35   59000]
[     48   41000]
[     48  134000]
[     38  113000]
[     29  148000]
[     26   15000]
[     60   42000]
[     24   19000]
[     42  149000]
[     46   96000]
[     28   59000]
[     39   96000]
[     28   89000]
[     41   72000]
[     45   26000]
[     33   69000]
[     20   82000]
[     31   74000]
[     42   80000]
[     35   72000]
[     33  149000]
[     40   71000]
[     51  146000]
[     46   79000]
[     35   75000]
[     38   51000]
[     36   75000]
[     37   78000]
[     38   61000]
[     60  108000]
[     20   82000]
[     57   74000]
[     42   65000]
[     26   80000]
[     46  117000]
[     35   61000]
```

```
[     21   68000]
[     28   44000]
[     41   87000]
[     37   33000]
[     27   90000]
[     39   42000]
[     28  123000]
[     31  118000]
[     25   87000]
[     35   71000]
[     37   70000]
[     35   39000]
[     47   23000]
[     35  147000]
[     48  138000]
[     26   86000]
[     25   79000]
[     52  138000]
[     51   23000]
[     35   60000]
[     33  113000]
[     30  107000]
[     48   33000]
[     41   80000]
[     48   96000]
[     31   18000]
[     31   71000]
[     43  129000]
[     59   76000]
[     18   44000]
[     36  118000]
[     42   90000]
[     47   30000]
[     26   43000]
[     40   78000]
[     46   59000]
[     59   42000]
[     46   74000]
[     35   91000]
[     28   59000]
[     40   57000]
[     59  143000]
[     57   26000]
[     52   38000]
[     47  113000]
[     53  143000]
[     35   27000]
[     58  101000]
```

```
[     45  45000]
[     23  82000]
[     46  23000]
[     42  65000]
[     28  84000]
[     38  59000]
[     26  84000]
[     29  28000]
[     37  71000]
[     22  55000]
[     48  35000]
[     49  28000]
[     38  65000]
[     27  17000]
[     46  28000]
[     48 141000]
[     26  17000]
[     35  97000]
[     39  59000]
[     24  27000]
[     32  18000]
[     46  88000]
[     35  58000]
[     56  60000]
[     47  34000]
[     40  72000]
[     32 100000]
[     19  21000]
[     25  90000]
[     35  88000]
[     28  32000]
[     50  20000]
[     40  59000]
[     50  44000]
[     35  72000]
[     40 142000]
[     46  32000]
[     39  71000]
[     20  74000]
[     29  75000]
[     31  76000]
[     47  25000]
[     40  61000]
[     34 112000]
[     38  80000]
[     42  75000]
[     47  47000]
[     39  75000]
```

```
[     19  25000]
[     37  80000]
[     36  60000]
[     41  52000]
[     36 125000]
[     48  29000]
[     36 126000]
[     51 134000]
[     27  57000]
[     38  71000]
[     39  61000]
[     22  27000]
[     33  60000]
[     48  74000]
[     58  23000]
[     53  72000]
[     32 117000]
[     54  70000]
[     30  80000]
[     58  95000]
[     26  52000]
[     45  79000]
[     24  55000]
[     40  75000]
[     33  28000]
[     44 139000]
[     22  18000]
[     33  51000]
[     43 133000]
[     24  32000]
[     46  22000]
[     35  55000]
[     54 104000]
[     48 119000]
[     35  53000]
[     37 144000]
[     23  66000]
[     37 137000]
[     31  58000]
[     33  41000]
[     45  22000]
[     30  15000]
[     19  19000]
[     49  74000]
[     39 122000]
[     35  73000]
[     39  71000]
[     24  23000]
```

```
[     41   72000]
[     29   83000]
[     54   26000]
[     35   44000]
[     37   75000]
[     29   47000]
[     31   68000]
[     42   54000]
[     30  135000]
[     52  114000]
[     50   36000]
[     56  133000]
[     29   61000]
[     30   89000]
[     26   16000]
[     33   31000]
[     41   72000]
[     36   33000]
[     55  125000]
[     48  131000]
[     41   71000]
[     30   62000]
[     37   72000]
[     41   63000]
[     58   47000]
[     30  116000]
[     20   49000]
[     37   74000]
[     41   59000]
[     49   89000]
[     28   79000]
[     53   82000]
[     40   57000]
[     60   34000]
[     35  108000]
[     21   72000]
[     38   71000]
[     39  106000]
[     37   57000]
[     26   72000]
[     35   23000]
[     54  108000]
[     30   17000]
[     39  134000]
[     29   43000]
[     33   43000]
[     35   38000]
[     41   45000]
```

```
[     41  72000]
[     39 134000]
[     27 137000]
[     21  16000]
[     26  32000]
[     31  66000]
[     39  73000]
[     41  79000]
[     47  50000]
[     41  30000]
[     37  93000]
[     60  46000]
[     25  22000]
[     28  37000]
[     38  55000]
[     36  54000]
[     20  36000]
[     56 104000]
[     40  57000]
[     42 108000]
[     20  23000]
[     40  65000]
[     47  20000]
[     18  86000]
[     35  79000]
[     57  33000]
[     34  72000]
[     49  39000]
[     27  31000]
[     19  70000]
[     39  79000]
[     26  81000]
[     25  80000]
[     28  85000]
[     55  39000]
[     50  88000]
[     49  88000]
[     52 150000]
[     35  65000]
[     42  54000]
[     34  43000]
[     37  52000]
[     48  30000]
[     29  43000]
[     36  52000]
[     27  54000]
[     26 118000]]
```

[ ]:
```
print(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]
 [    21  88000]
 [    37  79000]
 [    57  60000]
 [    37  53000]
 [    24  58000]
 [    18  52000]
 [    22  81000]
 [    34  43000]
 [    31  34000]
 [    49  36000]
 [    27  88000]
 [    41  52000]
 [    27  84000]
 [    35  20000]
 [    43 112000]
 [    27  58000]
```

```
[     37   80000]
[     52   90000]
[     26   30000]
[     49   86000]
[     57  122000]
[     34   25000]
[     35   57000]
[     34  115000]
[     59   88000]
[     45   32000]
[     29   83000]
[     26   80000]
[     49   28000]
[     23   20000]
[     32   18000]
[     60   42000]
[     19   76000]
[     36   99000]
[     19   26000]
[     60   83000]
[     24   89000]
[     27   58000]
[     40   47000]
[     42   70000]
[     32  150000]
[     35   77000]
[     22   63000]
[     45   22000]
[     27   89000]
[     18   82000]
[     42   79000]
[     40   60000]
[     53   34000]
[     47  107000]
[     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]]
```

[ ]:
```python
print(y_train)
```

```
[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]
```

[ ]:
```python
print(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 1 0 0 0 1 1 0 1 1 0 0 1 0 0 0 1 0 1 1 1]
```

[ ]:
```python
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
x_test = sc.transform(x_test)
```

[ ]:
```python
print(X_train)
```

```
[[ 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]
 [ 0.68068169  1.78066227]
```

```
[-0.70576986  0.56295021]
[ 0.77971394  0.35999821]
[ 0.8787462  -0.53878926]
[-1.20093113 -1.58254245]
[ 2.1661655   0.93986109]
[-0.01254409  1.22979253]
[ 0.18552042  1.08482681]
[ 0.38358493 -0.48080297]
[-0.30964085 -0.30684411]
[ 0.97777845 -0.8287207 ]
[ 0.97777845  1.8676417 ]
[-0.01254409  1.25878567]
[-0.90383437  2.27354572]
[-1.20093113 -1.58254245]
[ 2.1661655  -0.79972756]
[-1.39899564 -1.46656987]
[ 0.38358493  2.30253886]
[ 0.77971394  0.76590222]
[-1.00286662 -0.30684411]
[ 0.08648817  0.76590222]
[-1.00286662  0.56295021]
[ 0.28455268  0.07006676]
[ 0.68068169 -1.26361786]
[-0.50770535 -0.01691267]
[-1.79512465  0.35999821]
[-0.70576986  0.12805305]
[ 0.38358493  0.30201192]
[-0.30964085  0.07006676]
[-0.50770535  2.30253886]
[ 0.18552042  0.04107362]
[ 1.27487521  2.21555943]
[ 0.77971394  0.27301877]
[-0.30964085  0.1570462 ]
[-0.01254409 -0.53878926]
[-0.21060859  0.1570462 ]
[-0.11157634  0.24402563]
[-0.01254409 -0.24885782]
[ 2.1661655   1.11381995]
[-1.79512465  0.35999821]
[ 1.86906873  0.12805305]
[ 0.38358493 -0.13288524]
[-1.20093113  0.30201192]
[ 0.77971394  1.37475825]
[-0.30964085 -0.24885782]
[-1.6960924  -0.04590581]
[-1.00286662 -0.74174127]
[ 0.28455268  0.50496393]
[-0.11157634 -1.06066585]
```

```
[-1.10189888  0.59194336]
[ 0.08648817 -0.79972756]
[-1.00286662  1.54871711]
[-0.70576986  1.40375139]
[-1.29996338  0.50496393]
[-0.30964085  0.04107362]
[-0.11157634  0.01208048]
[-0.30964085 -0.88670699]
[ 0.8787462  -1.3505973 ]
[-0.30964085  2.24455257]
[ 0.97777845  1.98361427]
[-1.20093113  0.47597078]
[-1.29996338  0.27301877]
[ 1.37390747  1.98361427]
[ 1.27487521 -1.3505973 ]
[-0.30964085 -0.27785096]
[-0.50770535  1.25878567]
[-0.80480212  1.08482681]
[ 0.97777845 -1.06066585]
[ 0.28455268  0.30201192]
[ 0.97777845  0.76590222]
[-0.70576986 -1.49556302]
[-0.70576986  0.04107362]
[ 0.48261718  1.72267598]
[ 2.06713324  0.18603934]
[-1.99318916 -0.74174127]
[-0.21060859  1.40375139]
[ 0.38358493  0.59194336]
[ 0.8787462  -1.14764529]
[-1.20093113 -0.77073441]
[ 0.18552042  0.24402563]
[ 0.77971394 -0.30684411]
[ 2.06713324 -0.79972756]
[ 0.77971394  0.12805305]
[-0.30964085  0.6209365 ]
[-1.00286662 -0.30684411]
[ 0.18552042 -0.3648304 ]
[ 2.06713324  2.12857999]
[ 1.86906873 -1.26361786]
[ 1.37390747 -0.91570013]
[ 0.8787462   1.25878567]
[ 1.47293972  2.12857999]
[-0.30964085 -1.23462472]
[ 1.96810099  0.91086794]
[ 0.68068169 -0.71274813]
[-1.49802789  0.35999821]
[ 0.77971394 -1.3505973 ]
[ 0.38358493 -0.13288524]
```

```
[-1.00286662  0.41798449]
[-0.01254409 -0.30684411]
[-1.20093113  0.41798449]
[-0.90383437 -1.20563157]
[-0.11157634  0.04107362]
[-1.59706014 -0.42281668]
[ 0.97777845 -1.00267957]
[ 1.07681071 -1.20563157]
[-0.01254409 -0.13288524]
[-1.10189888 -1.52455616]
[ 0.77971394 -1.20563157]
[ 0.97777845  2.07059371]
[-1.20093113 -1.52455616]
[-0.30964085  0.79489537]
[ 0.08648817 -0.30684411]
[-1.39899564 -1.23462472]
[-0.60673761 -1.49556302]
[ 0.77971394  0.53395707]
[-0.30964085 -0.33583725]
[ 1.77003648 -0.27785096]
[ 0.8787462  -1.03167271]
[ 0.18552042  0.07006676]
[-0.60673761  0.8818748 ]
[-1.89415691 -1.40858358]
[-1.29996338  0.59194336]
[-0.30964085  0.53395707]
[-1.00286662 -1.089659  ]
[ 1.17584296 -1.43757673]
[ 0.18552042 -0.30684411]
[ 1.17584296 -0.74174127]
[-0.30964085  0.07006676]
[ 0.18552042  2.09958685]
[ 0.77971394 -1.089659  ]
[ 0.08648817  0.04107362]
[-1.79512465  0.12805305]
[-0.90383437  0.1570462 ]
[-0.70576986  0.18603934]
[ 0.8787462  -1.29261101]
[ 0.18552042 -0.24885782]
[-0.4086731   1.22979253]
[-0.01254409  0.30201192]
[ 0.38358493  0.1570462 ]
[ 0.8787462  -0.65476184]
[ 0.08648817  0.1570462 ]
[-1.89415691 -1.29261101]
[-0.11157634  0.30201192]
[-0.21060859 -0.27785096]
[ 0.28455268 -0.50979612]
```

```
[-0.21060859  1.6067034 ]
[ 0.97777845 -1.17663843]
[-0.21060859  1.63569655]
[ 1.27487521  1.8676417 ]
[-1.10189888 -0.3648304 ]
[-0.01254409  0.04107362]
[ 0.08648817 -0.24885782]
[-1.59706014 -1.23462472]
[-0.50770535 -0.27785096]
[ 0.97777845  0.12805305]
[ 1.96810099 -1.3505973 ]
[ 1.47293972  0.07006676]
[-0.60673761  1.37475825]
[ 1.57197197  0.01208048]
[-0.80480212  0.30201192]
[ 1.96810099  0.73690908]
[-1.20093113 -0.50979612]
[ 0.68068169  0.27301877]
[-1.39899564 -0.42281668]
[ 0.18552042  0.1570462 ]
[-0.50770535 -1.20563157]
[ 0.58164944  2.01260742]
[-1.59706014 -1.49556302]
[-0.50770535 -0.53878926]
[ 0.48261718  1.83864855]
[-1.39899564 -1.089659  ]
[ 0.77971394 -1.37959044]
[-0.30964085 -0.42281668]
[ 1.57197197  0.99784738]
[ 0.97777845  1.43274454]
[-0.30964085 -0.48080297]
[-0.11157634  2.15757314]
[-1.49802789 -0.1038921 ]
[-0.11157634  1.95462113]
[-0.70576986 -0.33583725]
[-0.50770535 -0.8287207 ]
[ 0.68068169 -1.37959044]
[-0.80480212 -1.58254245]
[-1.89415691 -1.46656987]
[ 1.07681071  0.12805305]
[ 0.08648817  1.51972397]
[-0.30964085  0.09905991]
[ 0.08648817  0.04107362]
[-1.39899564 -1.3505973 ]
[ 0.28455268  0.07006676]
[-0.90383437  0.38899135]
[ 1.57197197 -1.26361786]
[-0.30964085 -0.74174127]
```

```
[-0.11157634  0.1570462 ]
[-0.90383437 -0.65476184]
[-0.70576986 -0.04590581]
[ 0.38358493 -0.45180983]
[-0.80480212  1.89663484]
[ 1.37390747  1.28777882]
[ 1.17584296 -0.97368642]
[ 1.77003648  1.83864855]
[-0.90383437 -0.24885782]
[-0.80480212  0.56295021]
[-1.20093113 -1.5535493 ]
[-0.50770535 -1.11865214]
[ 0.28455268  0.07006676]
[-0.21060859 -1.06066585]
[ 1.67100423  1.6067034 ]
[ 0.97777845  1.78066227]
[ 0.28455268  0.04107362]
[-0.80480212 -0.21986468]
[-0.11157634  0.07006676]
[ 0.28455268 -0.19087153]
[ 1.96810099 -0.65476184]
[-0.80480212  1.3457651 ]
[-1.79512465 -0.59677555]
[-0.11157634  0.12805305]
[ 0.28455268 -0.30684411]
[ 1.07681071  0.56295021]
[-1.00286662  0.27301877]
[ 1.47293972  0.35999821]
[ 0.18552042 -0.3648304 ]
[ 2.1661655  -1.03167271]
[-0.30964085  1.11381995]
[-1.6960924   0.07006676]
[-0.01254409  0.04107362]
[ 0.08648817  1.05583366]
[-0.11157634 -0.3648304 ]
[-1.20093113  0.07006676]
[-0.30964085 -1.3505973 ]
[ 1.57197197  1.11381995]
[-0.80480212 -1.52455616]
[ 0.08648817  1.8676417 ]
[-0.90383437 -0.77073441]
[-0.50770535 -0.77073441]
[-0.30964085 -0.91570013]
[ 0.28455268 -0.71274813]
[ 0.28455268  0.07006676]
[ 0.08648817  1.8676417 ]
[-1.10189888  1.95462113]
[-1.6960924  -1.5535493 ]
```

```
 [-1.20093113 -1.089659  ]
 [-0.70576986 -0.1038921 ]
 [ 0.08648817  0.09905991]
 [ 0.28455268  0.27301877]
 [ 0.8787462  -0.5677824 ]
 [ 0.28455268 -1.14764529]
 [-0.11157634  0.67892279]
 [ 2.1661655  -0.68375498]
 [-1.29996338 -1.37959044]
 [-1.00286662 -0.94469328]
 [-0.01254409 -0.42281668]
 [-0.21060859 -0.45180983]
 [-1.79512465 -0.97368642]
 [ 1.77003648  0.99784738]
 [ 0.18552042 -0.3648304 ]
 [ 0.38358493  1.11381995]
 [-1.79512465 -1.3505973 ]
 [ 0.18552042 -0.13288524]
 [ 0.8787462  -1.43757673]
 [-1.99318916  0.47597078]
 [-0.30964085  0.27301877]
 [ 1.86906873 -1.06066585]
 [-0.4086731   0.07006676]
 [ 1.07681071 -0.88670699]
 [-1.10189888 -1.11865214]
 [-1.89415691  0.01208048]
 [ 0.08648817  0.27301877]
 [-1.20093113  0.33100506]
 [-1.29996338  0.30201192]
 [-1.00286662  0.44697764]
 [ 1.67100423 -0.88670699]
 [ 1.17584296  0.53395707]
 [ 1.07681071  0.53395707]
 [ 1.37390747  2.331532  ]
 [-0.30964085 -0.13288524]
 [ 0.38358493 -0.45180983]
 [-0.4086731  -0.77073441]
 [-0.11157634 -0.50979612]
 [ 0.97777845 -1.14764529]
 [-0.90383437 -0.77073441]
 [-0.21060859 -0.50979612]
 [-1.10189888 -0.45180983]
 [-1.20093113  1.40375139]]
```

[ ]:
```
print(x_test)
```

```
[[-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]
[-0.11157634 -0.42281668]
[ 0.08648817  0.21503249]
[-1.79512465  0.47597078]
[-0.60673761  1.37475825]
[-0.11157634  0.21503249]
[-1.89415691  0.44697764]
[ 1.67100423  1.75166912]
[-0.30964085 -1.37959044]
[-0.30964085 -0.65476184]
[ 0.8787462   2.15757314]
[ 0.28455268 -0.53878926]
[ 0.8787462   1.02684052]
[-1.49802789 -1.20563157]
[ 1.07681071  2.07059371]
[-1.00286662  0.50496393]
[-0.90383437  0.30201192]
[-0.11157634 -0.21986468]
[-0.60673761  0.47597078]
[-1.6960924   0.53395707]
[-0.11157634  0.27301877]
[ 1.86906873 -0.27785096]
[-0.11157634 -0.48080297]
[-1.39899564 -0.33583725]
[-1.99318916 -0.50979612]
[-1.59706014  0.33100506]
[-0.4086731  -0.77073441]
[-0.70576986 -1.03167271]
[ 1.07681071 -0.97368642]
[-1.10189888  0.53395707]
[ 0.28455268 -0.50979612]
[-1.10189888  0.41798449]
[-0.30964085 -1.43757673]
[ 0.48261718  1.22979253]
[-1.10189888 -0.33583725]
[-0.11157634  0.30201192]
[ 1.37390747  0.59194336]
[-1.20093113 -1.14764529]
[ 1.07681071  0.47597078]
```

```
[ 1.86906873  1.51972397]
[-0.4086731  -1.29261101]
[-0.30964085 -0.3648304 ]
[-0.4086731   1.31677196]
[ 2.06713324  0.53395707]
[ 0.68068169 -1.089659  ]
[-0.90383437  0.38899135]
[-1.20093113  0.30201192]
[ 1.07681071 -1.20563157]
[-1.49802789 -1.43757673]
[-0.60673761 -1.49556302]
[ 2.1661655  -0.79972756]
[-1.89415691  0.18603934]
[-0.21060859  0.85288166]
[-1.89415691 -1.26361786]
[ 2.1661655   0.38899135]
[-1.39899564  0.56295021]
[-1.10189888 -0.33583725]
[ 0.18552042 -0.65476184]
[ 0.38358493  0.01208048]
[-0.60673761  2.331532  ]
[-0.30964085  0.21503249]
[-1.59706014 -0.19087153]
[ 0.68068169 -1.37959044]
[-1.10189888  0.56295021]
[-1.99318916  0.35999821]
[ 0.38358493  0.27301877]
[ 0.18552042 -0.27785096]
[ 1.47293972 -1.03167271]
[ 0.8787462   1.08482681]
[ 1.96810099  2.15757314]
[ 2.06713324  0.38899135]
[-1.39899564 -0.42281668]
[-1.20093113 -1.00267957]
[ 1.96810099 -0.91570013]
[ 0.38358493  0.30201192]
[ 0.18552042  0.1570462 ]
[ 2.06713324  1.75166912]
[ 0.77971394 -0.8287207 ]
[ 0.28455268 -0.27785096]
[ 0.38358493 -0.16187839]
[-0.11157634  2.21555943]
[-1.49802789 -0.62576869]
[-1.29996338 -1.06066585]
[-1.39899564  0.41798449]
[-1.10189888  0.76590222]
[-1.49802789 -0.19087153]
[ 0.97777845 -1.06066585]
```

```
 [ 0.97777845  0.59194336]
 [ 0.38358493  0.99784738]]
```

[ ]:
```python
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p=2)
classifier.fit(X_train,y_train)
```

[ ]:
```
KNeighborsClassifier()
```

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

```
[1]
```

[ ]:
```python
y_pred = classifier.predict(x_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),y_test.
 ↪reshape(len(y_test),1)),1))
```

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

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

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

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

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

[ ]: 0.93

6 Practical 06: Implement K-means Clsutering

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[3]:
```
data = pd.read_csv('/content/drive/MyDrive/MSC CS/SEM 3/1. Machine Learning &␣
 ↪Deep Learning/Practicals/8. K-Means Clustering/Mall_Customers.csv')
x=data.iloc[:,[3,4]].values
```

[4]:
```
print(x)
```

```
[[ 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]
[ 25  73]
[ 28  14]
[ 28  82]
[ 28  32]
[ 28  61]
[ 29  31]
[ 29  87]
[ 30   4]
[ 30  73]
[ 33   4]
[ 33  92]
[ 33  14]
[ 33  81]
[ 34  17]
[ 34  73]
[ 37  26]
[ 37  75]
[ 38  35]
[ 38  92]
[ 39  36]
[ 39  61]
[ 39  28]
[ 39  65]
[ 40  55]
[ 40  47]
[ 40  42]
[ 40  42]
[ 42  52]
[ 42  60]
[ 43  54]
```

```
[ 43  60]
[ 43  45]
[ 43  41]
[ 44  50]
[ 44  46]
[ 46  51]
[ 46  46]
[ 46  56]
[ 46  55]
[ 47  52]
[ 47  59]
[ 48  51]
[ 48  59]
[ 48  50]
[ 48  48]
[ 48  59]
[ 48  47]
[ 49  55]
[ 49  42]
[ 50  49]
[ 50  56]
[ 54  47]
[ 54  54]
[ 54  53]
[ 54  48]
[ 54  52]
[ 54  42]
[ 54  51]
[ 54  55]
[ 54  41]
[ 54  44]
[ 54  57]
[ 54  46]
[ 57  58]
[ 57  55]
[ 58  60]
[ 58  46]
[ 59  55]
[ 59  41]
[ 60  49]
[ 60  40]
[ 60  42]
[ 60  52]
[ 60  47]
[ 60  50]
[ 61  42]
[ 61  49]
[ 62  41]
```

```
[ 62  48]
[ 62  59]
[ 62  55]
[ 62  56]
[ 62  42]
[ 63  50]
[ 63  46]
[ 63  43]
[ 63  48]
[ 63  52]
[ 63  54]
[ 64  42]
[ 64  46]
[ 65  48]
[ 65  50]
[ 65  43]
[ 65  59]
[ 67  43]
[ 67  57]
[ 67  56]
[ 67  40]
[ 69  58]
[ 69  91]
[ 70  29]
[ 70  77]
[ 71  35]
[ 71  95]
[ 71  11]
[ 71  75]
[ 71   9]
[ 71  75]
[ 72  34]
[ 72  71]
[ 73   5]
[ 73  88]
[ 73   7]
[ 73  73]
[ 74  10]
[ 74  72]
[ 75   5]
[ 75  93]
[ 76  40]
[ 76  87]
[ 77  12]
[ 77  97]
[ 77  36]
[ 77  74]
[ 78  22]
```

```
[ 78  90]
[ 78  17]
[ 78  88]
[ 78  20]
[ 78  76]
[ 78  16]
[ 78  89]
[ 78   1]
[ 78  78]
[ 78   1]
[ 78  73]
[ 79  35]
[ 79  83]
[ 81   5]
[ 81  93]
[ 85  26]
[ 85  75]
[ 86  20]
[ 86  95]
[ 87  27]
[ 87  63]
[ 87  13]
[ 87  75]
[ 87  10]
[ 87  92]
[ 88  13]
[ 88  86]
[ 88  15]
[ 88  69]
[ 93  14]
[ 93  90]
[ 97  32]
[ 97  86]
[ 98  15]
[ 98  88]
[ 99  39]
[ 99  97]
[101  24]
[101  68]
[103  17]
[103  85]
[103  23]
[103  69]
[113   8]
[113  91]
[120  16]
[120  79]
[126  28]
```

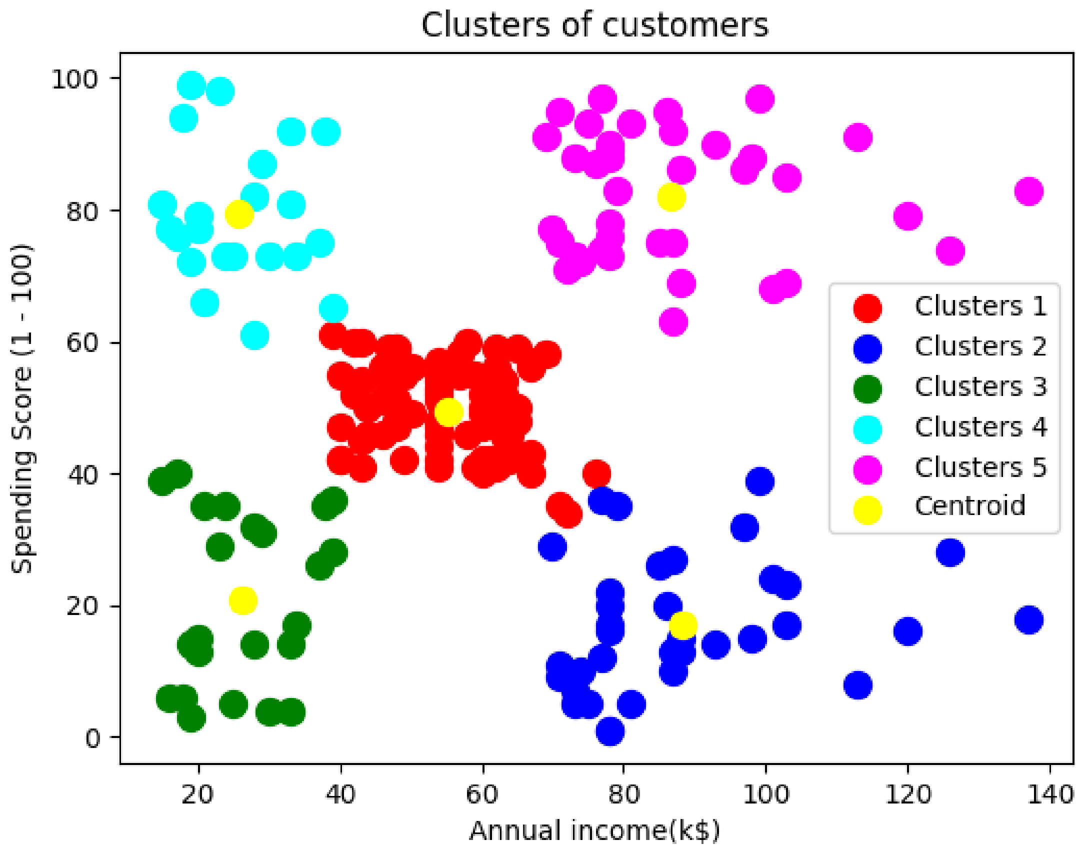
```
 [126  74]
 [137  18]
 [137  83]]
```

[5]:
```python
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)
```

```
[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 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 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 1 4]
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

[6]:
```python
plt.scatter(x[y_kmeans == 0,0],x[y_kmeans == 0,1], s=100, c = 'red', label =␣
 ↪'Clusters 1')
plt.scatter(x[y_kmeans == 1,0],x[y_kmeans == 1,1], s=100, c = 'blue', label =␣
 ↪'Clusters 2')
plt.scatter(x[y_kmeans == 2,0],x[y_kmeans == 2,1], s=100, c = 'green', label =␣
 ↪'Clusters 3')
plt.scatter(x[y_kmeans == 3,0],x[y_kmeans == 3,1], s=100, c = 'cyan', label =␣
 ↪'Clusters 4')
plt.scatter(x[y_kmeans == 4,0],x[y_kmeans == 4,1], s=100, c = 'magenta', label =␣
 ↪'Clusters 5')
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], s=100,␣
 ↪c='yellow', label = 'Centroid')
plt.title('Clusters of customers')
plt.xlabel('Annual income(k$)')
plt.ylabel('Spending Score (1 - 100)')
plt.legend()
plt.show()
```



7 Practical 07: Implement Hierarchical clustering

[26]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[27]:
```
data = pd.read_csv('/content/drive/MyDrive/Mall_Customers.csv')
x=data.iloc[:,[3,4]].values
```

[28]:
```
print(x)
```

```
[[ 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]
[ 25  73]
[ 28  14]
[ 28  82]
[ 28  32]
[ 28  61]
[ 29  31]
[ 29  87]
[ 30   4]
[ 30  73]
[ 33   4]
[ 33  92]
[ 33  14]
[ 33  81]
[ 34  17]
[ 34  73]
[ 37  26]
[ 37  75]
[ 38  35]
[ 38  92]
[ 39  36]
[ 39  61]
[ 39  28]
[ 39  65]
[ 40  55]
[ 40  47]
[ 40  42]
[ 40  42]
[ 42  52]
[ 42  60]
[ 43  54]
[ 43  60]
[ 43  45]
[ 43  41]
```

```
[ 44  50]
[ 44  46]
[ 46  51]
[ 46  46]
[ 46  56]
[ 46  55]
[ 47  52]
[ 47  59]
[ 48  51]
[ 48  59]
[ 48  50]
[ 48  48]
[ 48  59]
[ 48  47]
[ 49  55]
[ 49  42]
[ 50  49]
[ 50  56]
[ 54  47]
[ 54  54]
[ 54  53]
[ 54  48]
[ 54  52]
[ 54  42]
[ 54  51]
[ 54  55]
[ 54  41]
[ 54  44]
[ 54  57]
[ 54  46]
[ 57  58]
[ 57  55]
[ 58  60]
[ 58  46]
[ 59  55]
[ 59  41]
[ 60  49]
[ 60  40]
[ 60  42]
[ 60  52]
[ 60  47]
[ 60  50]
[ 61  42]
[ 61  49]
[ 62  41]
[ 62  48]
[ 62  59]
[ 62  55]
```

```
[ 62  56]
[ 62  42]
[ 63  50]
[ 63  46]
[ 63  43]
[ 63  48]
[ 63  52]
[ 63  54]
[ 64  42]
[ 64  46]
[ 65  48]
[ 65  50]
[ 65  43]
[ 65  59]
[ 67  43]
[ 67  57]
[ 67  56]
[ 67  40]
[ 69  58]
[ 69  91]
[ 70  29]
[ 70  77]
[ 71  35]
[ 71  95]
[ 71  11]
[ 71  75]
[ 71   9]
[ 71  75]
[ 72  34]
[ 72  71]
[ 73   5]
[ 73  88]
[ 73   7]
[ 73  73]
[ 74  10]
[ 74  72]
[ 75   5]
[ 75  93]
[ 76  40]
[ 76  87]
[ 77  12]
[ 77  97]
[ 77  36]
[ 77  74]
[ 78  22]
[ 78  90]
[ 78  17]
[ 78  88]
```

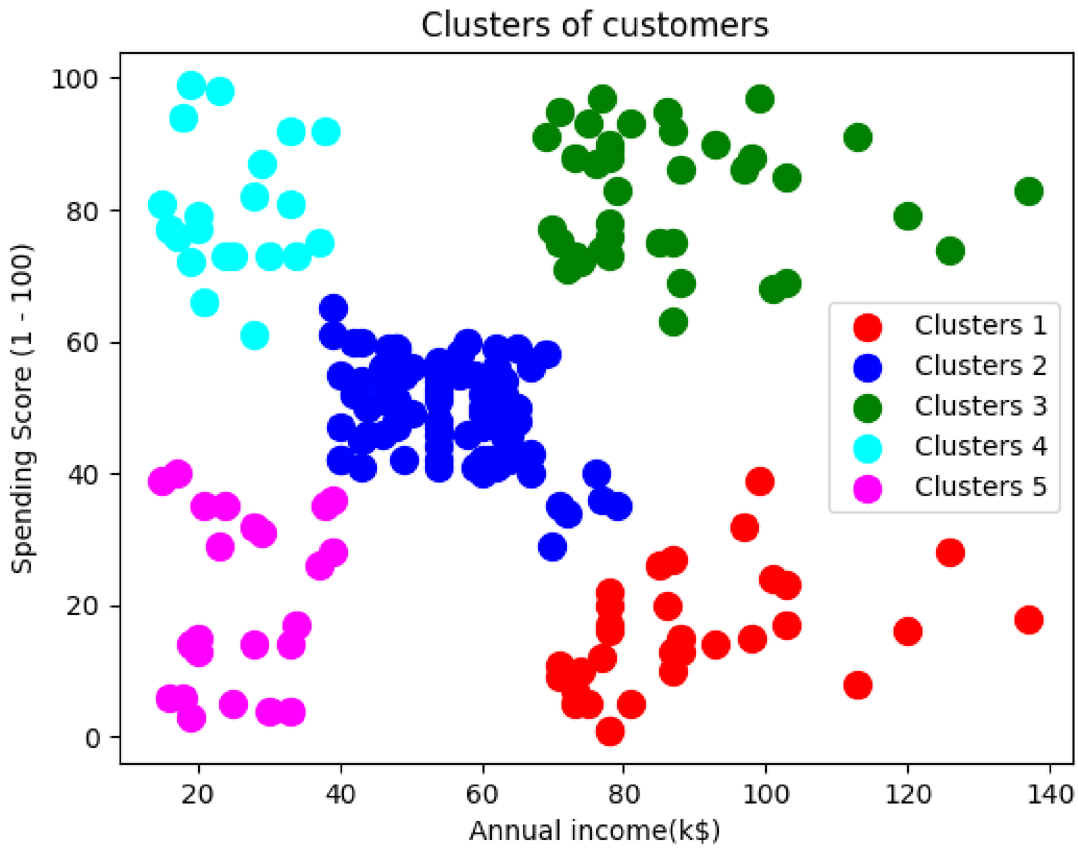
```
[ 78  20]
[ 78  76]
[ 78  16]
[ 78  89]
[ 78   1]
[ 78  78]
[ 78   1]
[ 78  73]
[ 79  35]
[ 79  83]
[ 81   5]
[ 81  93]
[ 85  26]
[ 85  75]
[ 86  20]
[ 86  95]
[ 87  27]
[ 87  63]
[ 87  13]
[ 87  75]
[ 87  10]
[ 87  92]
[ 88  13]
[ 88  86]
[ 88  15]
[ 88  69]
[ 93  14]
[ 93  90]
[ 97  32]
[ 97  86]
[ 98  15]
[ 98  88]
[ 99  39]
[ 99  97]
[101  24]
[101  68]
[103  17]
[103  85]
[103  23]
[103  69]
[113   8]
[113  91]
[120  16]
[120  79]
[126  28]
[126  74]
[137  18]
[137  83]]
```

[29]:
```
from sklearn.cluster import AgglomerativeClustering as AC
hc = AC(n_clusters = 5, affinity = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(x)
print(y_hc)
```

```
[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]
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_agglomerative.py:983:
FutureWarning: Attribute `affinity` was deprecated in version 1.2 and will be
removed in 1.4. Use `metric` instead
  warnings.warn(
```

[30]:
```
plt.scatter(x[y_hc == 0,0],x[y_hc == 0,1], s=100, c = 'red', label = 'Clusters␣
↪1')
plt.scatter(x[y_hc == 1,0],x[y_hc == 1,1], s=100, c = 'blue', label = 'Clusters␣
↪2')
plt.scatter(x[y_hc == 2,0],x[y_hc == 2,1], s=100, c = 'green', label = 'Clusters␣
↪3')
plt.scatter(x[y_hc == 3,0],x[y_hc == 3,1], s=100, c = 'cyan', label = 'Clusters␣
↪4')
plt.scatter(x[y_hc == 4,0],x[y_hc == 4,1], s=100, c = 'magenta', label =␣
↪'Clusters 5')
plt.title('Clusters of customers')
plt.xlabel('Annual income(k$)')
plt.ylabel('Spending Score (1 - 100)')
plt.legend()
plt.show()
```



8 Practical 08: Implement Artificial Neural Network

[ ]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
```

[11]:
```
tf.__version__
```

[11]: '2.12.0'

[12]:
```
dataset = pd.read_csv('/content/drive/MyDrive/MSC CS/SEM 3/1. Machine Learning &␣
 ↪Deep Learning/Practicals/10. Artificial Neural Network (ANN)/Churn_Modelling.
 ↪csv')
x = dataset.iloc[:, 3:-1].values
y = dataset.iloc[:, -1].values
```

[13]:
```
print(x)
```

```
[[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]]
```

[14]:
```python
print(y)
```

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

8.1 Encoding categorical data

[15]:
```python
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
x[:, 2] = le.fit_transform(x[:, 2])
```

[16]:
```python
print(x)
```

```
[[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]]
```

8.1.1 One Hot Encoding the "Geography" column

[17]:
```python
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))
```

[18]:
```python
print(x)
```

```
[[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]]
```

8.2 Splitting the dataset into the Training set and Test set

[19]:
```
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)
```

8.3 Feature Scaling

[20]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

8.4 Part 2 - Building the ANN

[21]:
```
ann = tf.keras.models.Sequential()
```

[22]:
```
ann.add(tf.keras.layers.Dense(units = 6, activation = 'relu'))
```

[23]:
```
ann.add(tf.keras.layers.Dense(units = 6, activation = 'relu'))
```

[24]:
```
ann.add(tf.keras.layers.Dense(units = 1, activation = 'sigmoid'))
```

[25]:
```
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =␣
↪['accuracy'])
```

[26]:
```
ann.fit(x_train, y_train, batch_size = 32, epochs = 100)
```

```
Epoch 1/100
250/250 [==============================] - 3s 5ms/step - loss: 0.5950 -
accuracy: 0.7820
Epoch 2/100
250/250 [==============================] - 1s 6ms/step - loss: 0.4888 -
accuracy: 0.7960
Epoch 3/100
250/250 [==============================] - 2s 7ms/step - loss: 0.4501 -
accuracy: 0.7960
Epoch 4/100
250/250 [==============================] - 2s 6ms/step - loss: 0.4315 -
accuracy: 0.8005
Epoch 5/100
250/250 [==============================] - 2s 7ms/step - loss: 0.4202 -
accuracy: 0.8058
Epoch 6/100
250/250 [==============================] - 2s 7ms/step - loss: 0.4119 -
accuracy: 0.8146
Epoch 7/100
250/250 [==============================] - 1s 5ms/step - loss: 0.4044 -
```

```
accuracy: 0.8221
Epoch 8/100
250/250 [==============================] - 1s 5ms/step - loss: 0.3968 -
accuracy: 0.8270
Epoch 9/100
250/250 [==============================] - 1s 4ms/step - loss: 0.3892 -
accuracy: 0.8319
Epoch 10/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3823 -
accuracy: 0.8374
Epoch 11/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3758 -
accuracy: 0.8424
Epoch 12/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3700 -
accuracy: 0.8436
Epoch 13/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3651 -
accuracy: 0.8481
Epoch 14/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3616 -
accuracy: 0.8480
Epoch 15/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3586 -
accuracy: 0.8509
Epoch 16/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3566 -
accuracy: 0.8512
Epoch 17/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3549 -
accuracy: 0.8519
Epoch 18/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3538 -
accuracy: 0.8524
Epoch 19/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3525 -
accuracy: 0.8530
Epoch 20/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3518 -
accuracy: 0.8534
Epoch 21/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3509 -
accuracy: 0.8543
Epoch 22/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3503 -
accuracy: 0.8553
Epoch 23/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3497 -
```

```
accuracy: 0.8544
Epoch 24/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3491 -
accuracy: 0.8560
Epoch 25/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3483 -
accuracy: 0.8561
Epoch 26/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3478 -
accuracy: 0.8572
Epoch 27/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3475 -
accuracy: 0.8571
Epoch 28/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3468 -
accuracy: 0.8596
Epoch 29/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3461 -
accuracy: 0.8576
Epoch 30/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3459 -
accuracy: 0.8579
Epoch 31/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3456 -
accuracy: 0.8602
Epoch 32/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3453 -
accuracy: 0.8589
Epoch 33/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3448 -
accuracy: 0.8597
Epoch 34/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3444 -
accuracy: 0.8596
Epoch 35/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3439 -
accuracy: 0.8591
Epoch 36/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3438 -
accuracy: 0.8597
Epoch 37/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3431 -
accuracy: 0.8586
Epoch 38/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3429 -
accuracy: 0.8605
Epoch 39/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3425 -
```

```
accuracy: 0.8597
Epoch 40/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3424 -
accuracy: 0.8610
Epoch 41/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3421 -
accuracy: 0.8590
Epoch 42/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3420 -
accuracy: 0.8595
Epoch 43/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3415 -
accuracy: 0.8602
Epoch 44/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3414 -
accuracy: 0.8608
Epoch 45/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3407 -
accuracy: 0.8604
Epoch 46/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3406 -
accuracy: 0.8596
Epoch 47/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3402 -
accuracy: 0.8610
Epoch 48/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3403 -
accuracy: 0.8611
Epoch 49/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3399 -
accuracy: 0.8639
Epoch 50/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3394 -
accuracy: 0.8631
Epoch 51/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3390 -
accuracy: 0.8622
Epoch 52/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3390 -
accuracy: 0.8622
Epoch 53/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3384 -
accuracy: 0.8616
Epoch 54/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3385 -
accuracy: 0.8618
Epoch 55/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3381 -
```

```
accuracy: 0.8619
Epoch 56/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3379 -
accuracy: 0.8629
Epoch 57/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3375 -
accuracy: 0.8624
Epoch 58/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3381 -
accuracy: 0.8614
Epoch 59/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3375 -
accuracy: 0.8624
Epoch 60/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3372 -
accuracy: 0.8625
Epoch 61/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3370 -
accuracy: 0.8621
Epoch 62/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3369 -
accuracy: 0.8630
Epoch 63/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3365 -
accuracy: 0.8627
Epoch 64/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3363 -
accuracy: 0.8621
Epoch 65/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3361 -
accuracy: 0.8620
Epoch 66/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3360 -
accuracy: 0.8614
Epoch 67/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3356 -
accuracy: 0.8644
Epoch 68/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3359 -
accuracy: 0.8610
Epoch 69/100
250/250 [==============================] - 1s 4ms/step - loss: 0.3355 -
accuracy: 0.8639
Epoch 70/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3355 -
accuracy: 0.8635
Epoch 71/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3354 -
```

```
accuracy: 0.8633
Epoch 72/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3354 -
accuracy: 0.8627
Epoch 73/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3352 -
accuracy: 0.8636
Epoch 74/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3352 -
accuracy: 0.8629
Epoch 75/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3350 -
accuracy: 0.8634
Epoch 76/100
250/250 [==============================] - 1s 4ms/step - loss: 0.3350 -
accuracy: 0.8641
Epoch 77/100
250/250 [==============================] - 1s 5ms/step - loss: 0.3347 -
accuracy: 0.8643
Epoch 78/100
250/250 [==============================] - 1s 4ms/step - loss: 0.3347 -
accuracy: 0.8651
Epoch 79/100
250/250 [==============================] - 1s 4ms/step - loss: 0.3346 -
accuracy: 0.8637
Epoch 80/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3344 -
accuracy: 0.8626
Epoch 81/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3346 -
accuracy: 0.8629
Epoch 82/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3343 -
accuracy: 0.8646
Epoch 83/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3344 -
accuracy: 0.8646
Epoch 84/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3341 -
accuracy: 0.8652
Epoch 85/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3347 -
accuracy: 0.8649
Epoch 86/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3343 -
accuracy: 0.8646
Epoch 87/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3341 -
```

```
accuracy: 0.8641
Epoch 88/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3346 -
accuracy: 0.8633
Epoch 89/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3340 -
accuracy: 0.8637
Epoch 90/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3342 -
accuracy: 0.8650
Epoch 91/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3341 -
accuracy: 0.8636
Epoch 92/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3340 -
accuracy: 0.8641
Epoch 93/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3342 -
accuracy: 0.8660
Epoch 94/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3338 -
accuracy: 0.8652
Epoch 95/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3340 -
accuracy: 0.8643
Epoch 96/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3336 -
accuracy: 0.8641
Epoch 97/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3342 -
accuracy: 0.8640
Epoch 98/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3340 -
accuracy: 0.8639
Epoch 99/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3339 -
accuracy: 0.8630
Epoch 100/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3339 -
accuracy: 0.8644
```

[26]:
```
<keras.callbacks.History at 0x7c19386b3f40>
```

[27]:
```
print(ann.predict(sc.transform([[1, 0, 0, 600, 1, 40, 3, 60000, 2, 1, 1,
 50000]])) >0.5)
```

```
1/1 [==============================] - 0s 238ms/step
[[False]]
```

[28]:
```
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))
```

```
63/63 [==============================] - 0s 2ms/step
[[0 0]
 [0 1]
 [0 0]
 ...
 [0 0]
 [0 0]
 [0 0]]
```

[29]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[1527   68]
 [ 205  200]]
```

[29]:
```
0.8635
```

[ ]:
```

```

[ ]:
```

```

9 Practical 09: Cat vs Dog classification using Convolution neural network

[ ]:
```
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator as IDG
```

[ ]:
```
tf.__version__
```

[ ]:
```
'2.12.0'
```

[ ]:
```
train_datagen =IDG(rescale = 1./255,
                   shear_range = 0.2,
                   zoom_range = 0.2,
                   horizontal_flip=True)
training_set = train_datagen.flow_from_directory('/content/drive/MyDrive/MSC CS/
 ↪SEM 3/1. Machine Learning & Deep Learning/Practicals/11. Convolutional Neural␣
 ↪Network (CNN)/small_dataset/training_set',
                                                 target_size = (64,64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')
```

Found 10 images belonging to 2 classes.

[ ]:
```
test_datagen = IDG(rescale = 1./255)
test_set = test_datagen.flow_from_directory('/content/drive/MyDrive/MSC CS/SEM 3/
↪1. Machine Learning & Deep Learning/Practicals/11. Convolutional Neural␣
↪Network (CNN)/small_dataset/test_set',
                                             target_size=(64,64),
                                             batch_size=32,
                                             class_mode = 'binary')
```

Found 10 images belonging to 2 classes.

[ ]:
```
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)
```

```
Epoch 1/25
1/1 [==============================] - 4s 4s/step - loss: 0.7091 - accuracy:
0.3000 - val_loss: 0.7031 - val_accuracy: 0.5000
Epoch 2/25
1/1 [==============================] - 0s 146ms/step - loss: 0.6483 - accuracy:
0.5000 - val_loss: 1.0494 - val_accuracy: 0.5000
Epoch 3/25
1/1 [==============================] - 0s 132ms/step - loss: 0.9677 - accuracy:
0.5000 - val_loss: 0.7256 - val_accuracy: 0.5000
Epoch 4/25
1/1 [==============================] - 0s 138ms/step - loss: 0.5912 - accuracy:
0.6000 - val_loss: 0.7697 - val_accuracy: 0.5000
Epoch 5/25
1/1 [==============================] - 0s 134ms/step - loss: 0.6717 - accuracy:
0.5000 - val_loss: 0.7872 - val_accuracy: 0.5000
```

```
Epoch 6/25
1/1 [==============================] - 0s 151ms/step - loss: 0.6907 - accuracy:
0.5000 - val_loss: 0.7485 - val_accuracy: 0.3000
Epoch 7/25
1/1 [==============================] - 0s 133ms/step - loss: 0.5686 - accuracy:
0.6000 - val_loss: 0.7458 - val_accuracy: 0.4000
Epoch 8/25
1/1 [==============================] - 0s 138ms/step - loss: 0.5452 - accuracy:
0.9000 - val_loss: 0.7798 - val_accuracy: 0.5000
Epoch 9/25
1/1 [==============================] - 0s 135ms/step - loss: 0.5628 - accuracy:
0.6000 - val_loss: 0.7959 - val_accuracy: 0.5000
Epoch 10/25
1/1 [==============================] - 0s 136ms/step - loss: 0.5208 - accuracy:
0.8000 - val_loss: 0.7930 - val_accuracy: 0.5000
Epoch 11/25
1/1 [==============================] - 0s 129ms/step - loss: 0.5094 - accuracy:
0.8000 - val_loss: 0.7977 - val_accuracy: 0.3000
Epoch 12/25
1/1 [==============================] - 0s 135ms/step - loss: 0.4396 - accuracy:
0.9000 - val_loss: 0.8243 - val_accuracy: 0.3000
Epoch 13/25
1/1 [==============================] - 0s 136ms/step - loss: 0.4396 - accuracy:
0.8000 - val_loss: 0.8532 - val_accuracy: 0.3000
Epoch 14/25
1/1 [==============================] - 0s 131ms/step - loss: 0.3826 - accuracy:
0.8000 - val_loss: 0.8776 - val_accuracy: 0.3000
Epoch 15/25
1/1 [==============================] - 0s 128ms/step - loss: 0.4714 - accuracy:
0.9000 - val_loss: 0.9454 - val_accuracy: 0.7000
Epoch 16/25
1/1 [==============================] - 0s 156ms/step - loss: 0.3074 - accuracy:
0.9000 - val_loss: 0.9897 - val_accuracy: 0.7000
Epoch 17/25
1/1 [==============================] - 0s 132ms/step - loss: 0.3913 - accuracy:
0.8000 - val_loss: 1.0137 - val_accuracy: 0.4000
Epoch 18/25
1/1 [==============================] - 0s 136ms/step - loss: 0.2495 - accuracy:
1.0000 - val_loss: 1.0508 - val_accuracy: 0.3000
Epoch 19/25
1/1 [==============================] - 0s 133ms/step - loss: 0.2436 - accuracy:
1.0000 - val_loss: 1.1137 - val_accuracy: 0.3000
Epoch 20/25
1/1 [==============================] - 0s 135ms/step - loss: 0.2296 - accuracy:
1.0000 - val_loss: 1.1719 - val_accuracy: 0.4000
Epoch 21/25
1/1 [==============================] - 0s 145ms/step - loss: 0.2185 - accuracy:
1.0000 - val_loss: 1.2464 - val_accuracy: 0.4000
```

```
Epoch 22/25
1/1 [==============================] - 0s 136ms/step - loss: 0.2822 - accuracy:
0.9000 - val_loss: 1.3149 - val_accuracy: 0.4000
Epoch 23/25
1/1 [==============================] - 0s 133ms/step - loss: 0.2895 - accuracy:
0.9000 - val_loss: 1.4116 - val_accuracy: 0.4000
Epoch 24/25
1/1 [==============================] - 0s 135ms/step - loss: 0.2046 - accuracy:
1.0000 - val_loss: 1.4139 - val_accuracy: 0.2000
Epoch 25/25
1/1 [==============================] - 0s 132ms/step - loss: 0.0799 - accuracy:
1.0000 - val_loss: 1.6286 - val_accuracy: 0.3000
```

[ ]:
```
<keras.callbacks.History at 0x7929b4bd3310>
```

[ ]:
```python
import numpy as np
from tensorflow.keras.preprocessing import image
test_image = image.load_img('/content/drive/MyDrive/MSC CS/SEM 3/1. Machine␣
 ↪Learning & Deep Learning/Practicals/11. Convolutional Neural Network (CNN)/
 ↪small_dataset/single_prediction/cat_or_dog_2.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"
```

```
1/1 [==============================] - 0s 17ms/step
```

[ ]:
```python
print(prediction)
```

```
Cat
```