Name : Yash Kesharwani Roll No : 530

Class : MSC CS Part 2 Subject : MLDL

Machine Learning Practicals

INDEX

Simple Linear Regression (Practical 1)

Importing the libraries

In [2]:

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

Importing the dataset

In [7]:

```
dataset = pd.read_csv('E:\Salary_Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

In [8]:

```
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]]
```

[9]:

```
print(y)
```

In

```
[ 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.]
```

Splitting the dataset into the Training set and Test set

In [10]:

```
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
```

In [11]:

```
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. ]] In
```

[12]:

```
print(X_test)
```

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

```
In
 [ 5.3]
 [ 7.9]]
   [13]:
```

```
print(y_train)
```

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

67938.  56957.] In [14]:
```

```
print(y_test)
```

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

Training the Simple Linear Regression model on the Training set

In [15]:

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

Out[15]:

```
LinearRegression()
```

Predicting the Test set results

In [ ]:

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

In

Visualising the Training set results

[11]:

```
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()
```

In

Visualising the Test set results

In [12]:

```
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()
```

In [ ]:

In

Multiple Linear Regression (Practical 2)

Importing the libraries

In [0]:

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

Importing the dataset

In [0]:

```
dataset = pd.read_csv('50_Startups.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

[3]:

```
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']]
```

In

Encoding categorical data

In [0]:

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

In [5]:

```
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]]
```

In

Splitting the dataset into the Training set and Test set

[0]:

```
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
```

Training the Multiple Linear Regression model on the Training set

In [7]:

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

Out[7]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fal
se)
```

Predicting the Test set results

In [8]:

```
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]]
```

K-Nearest Neighbors (KNN) (Practical 3)

Importing the libraries

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive In

[1]:

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

Importing the dataset

In [2]:

```
dataset = pd.read_csv('E:\Machine Learning\Datasets\Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

In [3]:

```
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 =
```

In [4]:

```
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]
```

In [5]:

```
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]
```

In [6]:

```
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]] In
```

[7]:

```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]
```

Feature Scaling

In [8]:

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

In [9]:

```
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]
```

In [10]:

```
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]]
```

Training the K-NN model on the Training set

In [11]:

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

Out[11]:

```
KNeighborsClassifier()
```

Predicting a new result

In [12]:

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

```
[1]
```

Predicting the Test set results

[13]:

```
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]
```

In

```
 [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]]
```

Making the Confusion Matrix

[14]:

```
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]]
```

Out[14]:

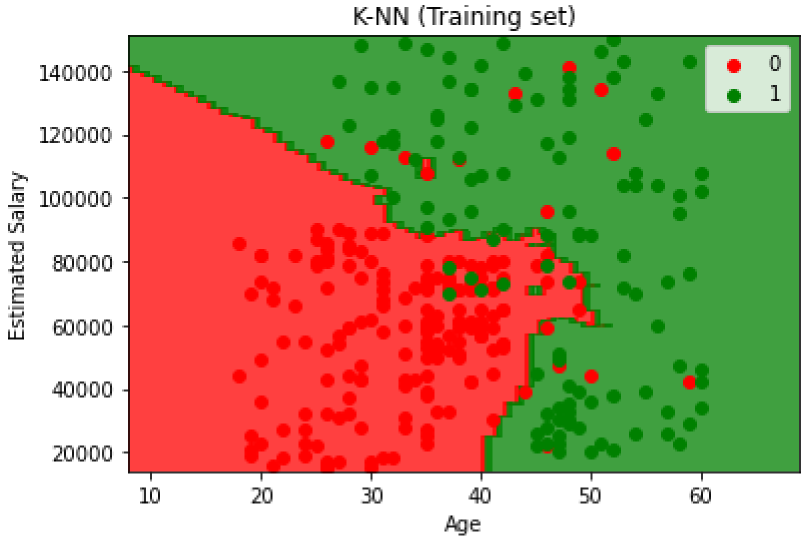
```
0.93
```

Visualising the Training set results

In [16]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 1
np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + plt.contourf(X1, X2,
classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T))            alpha =
0.75, cmap = ListedColormap(('red', 'green'))) plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max()) for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'gre
plt.title('K-NN (Training set)') plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend() plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2D arr ay
with a single row if you intend to specify the same RGB or RGBA value for all
points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2D arr ay
with a single row if you intend to specify the same RGB or RGBA value for all
points.
```



Visualising the Test set results

[ ]:

```
from matplotlib.colors import ListedColormap X_set,
y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 1
np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + plt.contourf(X1, X2,
classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T))            alpha =
0.75, cmap = ListedColormap(('red', 'green'))) plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max()) for i, j in enumerate(np.unique(y_set)):
```

```
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'gre
plt.title('K-NN (Test set)') plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend() plt.show()
```

In

Support Vector Machine (SVM) (Practical 4)

Importing the libraries

In [1]:

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

Importing the dataset

In [2]:

```
dataset = pd.read_csv('E:\Machine Learning\Datasets\Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

In [3]:

```
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 =
```

In [4]:

```
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]
```

In [5]:

```
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]
```

In [6]:

```
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]] In
```

[7]:

```
print(y_test)
```

```
[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]
```

Feature Scaling

In [8]:

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

In [9]:

```
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]
```

In [10]:

```
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]]
```

Training the SVM model on the Training set

In [11]:

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

Out[11]:

```
SVC(kernel='linear', random_state=0)
```

Predicting a new result

In [12]:

```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Predicting the Test set results

In [13]:

```
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]]
```

Making the Confusion Matrix

In [14]:

```
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]]
```

Out[14]:

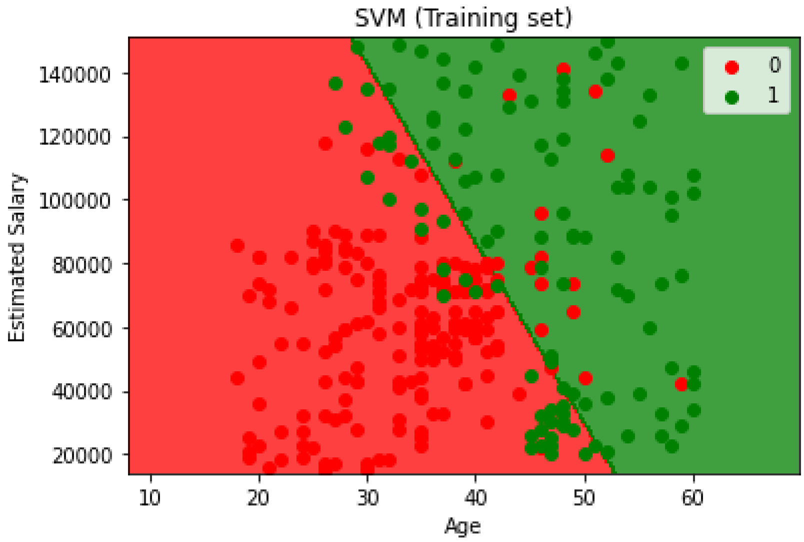
0.9

Visualising the Training set results

In [15]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 1
np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + plt.contourf(X1, X2,
classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T))          alpha =
0.75, cmap = ListedColormap(('red', 'green'))) plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max()) for i, j in enumerate(np.unique(y_set)):
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'gre
plt.title('SVM (Training set)') plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend() plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row if you really want to
specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row if you really want to
specify the same RGB or RGBA value for all points.
```



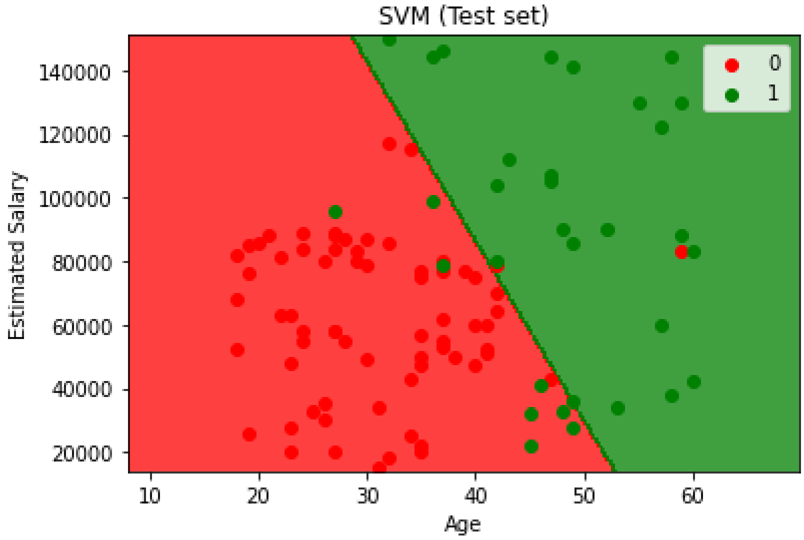
Visualising the Test set results

In [16]:

```
from matplotlib.colors import ListedColormap X_set,
y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 1
np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + plt.contourf(X1, X2,
classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T))          alpha =
0.75, cmap = ListedColormap(('red', 'green'))) plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max()) for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'gre
plt.title('SVM (Test set)') plt.xlabel('Age')
```

```
plt.ylabel('Estimated Salary')
plt.legend() plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row if you really want to
specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row if you really want to
specify the same RGB or RGBA value for all points.
```



K-Means Clustering (Practical 5)

Importing the libraries

In [1]:

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

Importing the dataset

In [2]:

```
dataset = pd.read_csv('E:\Machine Learning\Datasets\Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values
```

In [3]:

```
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]
```

Using the elbow method to find the optimal number of clusters

In [ ]:

```
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

```
C:\Users\Anurag\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1036:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting th
e environment variable OMP_NUM_THREADS=1.  warnings.warn(
```

Training the K-Means model on the dataset

In [ ]:

```
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)
```

Visualising the clusters

In [ ]:

```
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluste
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Clus
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yel
```

```
plt.title('Clusters of customers') plt.xlabel('Annual Income (k$)') plt.ylabel('Spending Score (1-100)') plt.legend() plt.show()
```

Hierarchical Clustering (Practical 6)

Importing the libraries

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, cal
l drive.mount("/content/drive", force_remount=True).
```

In [1]:

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

Importing the dataset

In [2]:

```
dataset = pd.read_csv('E:\Machine Learning\Datasets\Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values
```

In [3]:

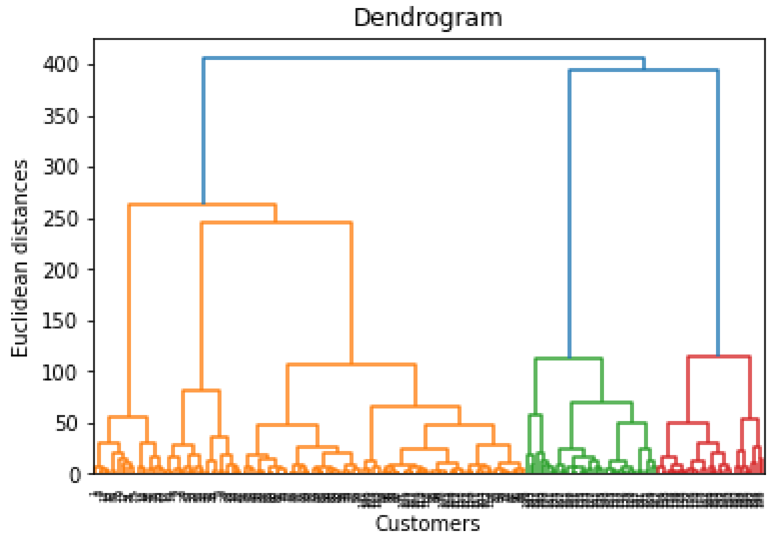
```
print(X)
```

```
[[ 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]
```

Using the dendrogram to find the optimal number of clusters

In [4]:

```
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()
```



Training the Hierarchical Clustering model on the dataset

In [5]:

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

In [ ]:

```
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 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 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 0 2]
```

In

Visualising the clusters

[ ]:

```
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 = 'blue', label = 'Cluster 2')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'green', 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()
```

Artificial Neural Network (Practical 7)

In [1]:

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

In [2]:

```
tf.__version__
```

Out[2]:

```
'2.9.1'
```

In [3]:

```
dataset=pd.read_csv('E:\Machine Learning\Datasets\Churn_Modelling.csv')
x=dataset.iloc[:,3:-1].values y=dataset.iloc[:,-1].values In [4]:
```

```
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]] In
```

[5]:

```
print(y)
```

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

[6]:

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

[7]:

```
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]
```

In

```
 [792 'France' 0 ... 1 0 38190.78]]
```

In [8]:

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

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

In [9]:

```
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]]
```

In [10]:

```
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)
```

In [11]:

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

In [12]:

```
ann=tf.keras.models.Sequential()
```

In [13]:

```
ann.add(tf.keras.layers.Dense(units=6 ,activation="relu"))    #input layer
```

In [14]:

```
ann.add(tf.keras.layers.Dense(units=6 ,activation="relu"))    #hidden layer
```

[15]:

```
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))   #output layer
```

In [16]:

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

In [17]:

In

```
ann.fit(x_train, y_train, batch_size=32, epochs=100)
```

```
Epoch 1/100
250/250 [==============================] - 9s 1ms/step - loss: 0.5553 - ac
curacy: 0.7929
Epoch 2/100
250/250 [==============================] - 0s 1ms/step - loss: 0.4850 - ac
curacy: 0.7972
Epoch 3/100
250/250 [==============================] - 0s 1ms/step - loss: 0.4448 - ac
curacy: 0.8089
Epoch 4/100
250/250 [==============================] - 0s 1ms/step - loss: 0.4235 - ac
curacy: 0.8181
Epoch 5/100
250/250 [==============================] - 0s 1ms/step - loss: 0.4093 - ac
curacy: 0.8229
Epoch 6/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3979 - ac
curacy: 0.8260
Epoch 7/100
250/250 [                              ]   0   1   / t     l     0 3878
```

In [18]:

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

```
1/1 [==============================] - 0s 383ms/step
[[False]]
```

In [19]:

```
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 974us/step
[[0 0]
 [0 1]
 [0 0] ...
 [0 0]
 [0 0]
 [0 0]]
```

[20]:

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

```
[[1517   78]
 [ 202  203]]
```

Out[20]:

```
0.86
```

Convolutional Neural Network (Practical 8)

Importing the libraries

+ Code + Text

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

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

```
tf.__version__
```

```
'2.8.2'
```

Part 1 - Data Preprocessing

Preprocessing the Training set

```
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/tra
target_size = (64, 64),                                                   batch_size = 32,
class_mode = 'binary')
```

Saved successfully!

```
Found 10 images belonging to 2 classes.
```

Preprocessing the Test set

```
test_datagen = ImageDataGenerator(rescale = 1./255) test_set =
test_datagen.flow_from_directory('/content/drive/MyDrive/small_dataset/test_set
target_size = (64, 64),                                                batch_size = 32,
class_mode = 'binary')
```

```
Found 10 images belonging to 2 classes.
```

Part 2 - Building the CNN

Initialising the CNN

```
cnn = tf.keras.models.Sequential()
```

Step 1 - Convolution

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

Step 2 - Pooling

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

Adding a second convolutional layer

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

Saved successfully!

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

Step 4 - Full Connection

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

Step 5 - Output Layer

```
cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

Part 3 - Training the CNN

Compiling the CNN

```
cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Training the CNN on the Training set and evaluating it on the Test set

```
cnn.fit(x = training_set, validation_data = test_set, epochs = 50)
```

```
Epoch 1/50
1/1 [==============================] - 5s 5s/step - loss: 0.6895 - accuracy: 0.5000
Epoch 2/50
1/1 [==============================] - 0s 217ms/step - loss: 0.6743 - accuracy: 0.7
Epoch 3/50
1/1 [==============================] - 0s 217ms/step - loss: 0.6113 - accuracy: 0.
Epoch 4/50
1/1 [==============================] - 0s 223ms/step - loss: 0.5911 - accuracy: 0.8
Epoch 5/50
1/1 [==============================] - 0s 211ms/step - loss: 0.5268 - accuracy: 0.8
Epoch 6/50
1/1 [==============================] - 0s 214ms/step - loss: 0.4662 - accuracy: 0.9
Epoch 7/50
1/1 [==============================] - 0s 216ms/step - loss: 0.4262 - accuracy: 0.9
Epoch 8/50
1/1 [==============================] - 0s 217ms/step - loss: 0.3625 - accuracy: 0.9
Epoch 9/50
1/1 [==============================] - 0s 216ms/step - loss: 0.4231 - accuracy: 0.9
Epoch 10/50
1/1 [==============================] - 0s 224ms/step - loss: 0.3852 - accuracy: 0.9
Epoch 11/50
1/1 [==============================] - 0s 231ms/step - loss: 0.2335 - accuracy: 1.0
Epoch 12/50
1/1 [==============================] - 0s 218ms/step - loss: 0.2532 - accuracy: 0.9
Saved successfully!Epoch 13/50
1/1 [==============================] - 0s 216ms/step - loss: 0.3169 - accuracy: 0.9
Epoch 14/50
1/1 [==============================] - 0s 226ms/step - loss: 0.1412 - accuracy: 1.0
Epoch 15/50
1/1 [==============================] - 0s 216ms/step - loss: 0.2598 - accuracy: 0.9
Epoch 16/50
1/1 [==============================] - 0s 230ms/step - loss: 0.2152 - accuracy: 0.8
Epoch 17/50
1/1 [==============================] - 0s 216ms/step - loss: 0.2076 - accuracy: 1.0
Epoch 18/50
1/1 [==============================] - 0s 229ms/step - loss: 0.1707 - accuracy: 0.9
Epoch 19/50
1/1 [==============================] - 0s 217ms/step - loss: 0.0839 - accuracy: 1.0
Epoch 20/50
1/1 [==============================] - 0s 224ms/step - loss: 0.1150 - accuracy: 1.0
Epoch 21/50
1/1 [==============================] - 0s 222ms/step - loss: 0.0378 - accuracy: 1.0
Epoch 22/50
1/1 [==============================] - 0s 238ms/step - loss: 0.2087 - accuracy: 0.9
Epoch 23/50
```

```
1/1 [==============================] - 0s 223ms/step - loss: 0.0454 - accuracy: 1.0
Epoch 24/50
1/1 [==============================] - 0s 220ms/step - loss: 0.3098 - accuracy: 0.8
Epoch 25/50
1/1 [==============================] - 0s 216ms/step - loss: 0.0393 - accuracy: 1.0
Epoch 26/50
1/1 [==============================] - 0s 235ms/step - loss: 0.3973 - accuracy: 0.8
Epoch 27/50
1/1 [==============================] - 0s 223ms/step - loss: 0.1023 - accuracy: 0.9
Epoch 28/50
1/1 [==============================] - 0s 209ms/step - loss: 0.0104 - accuracy: 1.0
E   h 29/50
```

Part 4 - Making a single prediction

```
import numpy as np from
keras.preprocessing import image
test_image = image.load_img('/content/drive/MyDrive/small_dataset/single_prediction/cat_or_
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) cat
```

Saved successfully!

Colab paid products - Cancel contracts here