PRACTICAL-1

AIM: **Implement Data Preprocessing in** R or Python

1.Importing Libraries

In [1]:
```
#Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

2.Importing Dataset

In [2]:
```
#Importing dataset
df = pd.read_csv("Data Pre.csv", sep=' ')
df
```

Out[2]:

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

In [3]:
```
X = df.iloc[:,[0,1,2]].values
y = df.iloc[:,3].values
```

In [4]:
```
X
```

Out[4]:
```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, nan],
       ['France', 35.0, 58000.0],
       ['Spain', nan, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

In [5]:
```
y
```

Out[5]:
```
array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'],
      dtype=object)
```

3.Handling missing values

In [6]:
```
#Handling missing values
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan,strategy="mean")
imputer.fit(X[:,1:3])
X[:,1:3] = imputer.transform(X[:,1:3])
X
```

Out[6]:
```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, 63777.77777777778],
       ['France', 35.0, 58000.0],
       ['Spain', 38.77777777777778, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

4.Encoding the independent variables

In [7]:
```
#encoding the independent variables
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainder='passthrough')
X = ct.fit_transform(X)
X = np.array(X)
X
```

Out[7]:
```
array([[1.0, 0.0, 0.0, 44.0, 72000.0],
       [0.0, 0.0, 1.0, 27.0, 48000.0],
       [0.0, 1.0, 0.0, 30.0, 54000.0],
       [0.0, 0.0, 1.0, 38.0, 61000.0],
       [0.0, 1.0, 0.0, 40.0, 63777.77777777778],
       [1.0, 0.0, 0.0, 35.0, 58000.0],
       [0.0, 0.0, 1.0, 38.77777777777778, 52000.0],
       [1.0, 0.0, 0.0, 48.0, 79000.0],
       [0.0, 1.0, 0.0, 50.0, 83000.0],
       [1.0, 0.0, 0.0, 37.0, 67000.0]], dtype=object)
```

5.Encoding the dependent variables

In [8]:
```
#encoding the dependent variables
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
y
```

Out[8]:
```
array([0, 1, 0, 0, 1, 1, 0, 1, 0, 1])
```

6.Split data into train and test dataset

In [9]:
```
#split data into train and test dataset
from sklearn.model_selection import train_test_split  #(for python2)
#from sklearn.model_selection import train_test_split  (for python3)
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=10)
print('X_train.shape: ', X_train.shape)
print('X_test.shape:  ', X_test.shape)
print('y_train.shape: ', y_train.shape)
print('y_test.shape:  ', y_test.shape)
X_train
```

```
X_train.shape:  (8, 5)
X_test.shape:   (2, 5)
y_train.shape:  (8,)
y_test.shape:   (2,)
```

Out[9]:
```
array([[1.0, 0.0, 0.0, 35.0, 58000.0],
       [0.0, 0.0, 1.0, 38.77777777777778, 52000.0],
       [0.0, 0.0, 1.0, 38.0, 61000.0],
       [0.0, 0.0, 1.0, 27.0, 48000.0],
       [1.0, 0.0, 0.0, 44.0, 72000.0],
       [1.0, 0.0, 0.0, 48.0, 79000.0],
       [0.0, 1.0, 0.0, 40.0, 63777.77777777778],
       [1.0, 0.0, 0.0, 37.0, 67000.0]], dtype=object)
```

7.Feature scaling of training dataset

In [10]:
```
y_train
```

Out[10]:
```
array([1, 0, 0, 1, 0, 1, 1, 1])
```

In [11]:
```
#feature scaling of training dataset
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

In [12]:
```
X_train
```

Out[12]:
```
array([[ 1.        , -0.37796447, -0.77459667, -0.59788085, -0.48214934],
       [-1.        , -0.37796447,  1.29099445,  0.05261351, -1.11141978],
       [-1.        , -0.37796447,  1.29099445, -0.0813118 , -0.16751412],
       [-1.        , -0.37796447,  1.29099445, -1.97539832, -1.53093341],
       [ 1.        , -0.37796447, -0.77459667,  0.95182631,  0.98614835],
       [ 1.        , -0.37796447, -0.77459667,  1.64058505,  1.7202972 ],
       [-1.        ,  2.64575131, -0.77459667,  0.26306757,  0.12381479],
       [ 1.        , -0.37796447, -0.77459667, -0.25350148,  0.46175632]])
```

In [13]:
```
X_test
```

Out[13]:
```
array([[-1.        ,  2.64575131, -0.77459667,  1.98496442,  2.13981082],
       [-1.        ,  2.64575131, -0.77459667, -1.45882927, -0.90166297]])
```

In [14]:
```
y_train
```

Out[14]:
```
array([1, 0, 0, 1, 0, 1, 1, 1])
```

In [15]:
```
y_test
```

Out[15]:
```
array([0, 0])
```

In [ ]:

PRACTICAL -2

AIM: IMPLEMENT LINEAR REGRESSION IN R OR PYTHON.

Importing The Libraries:

In [29]:
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import metrics as me
```

Importing The Dataset

In [30]:
```
# Importing the dataset
dataset = pd.read_csv('Data.csv')
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

print(f"{x} \n")
print(f"{y} \n\n\n")
```

```
[[ 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]]

[ 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 [31]:
```
# Splitting the dataset into the Training set and Test set
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)

print("After Splitting, The Train And The Test Set:-")
print(f"x_train:-\n\n{x_train} \n")
print(f"x_test:-\n\n{x_test} \n")
print(f"y_train:-\n\n{y_train} \n")
print(f"y_test:-\n\n{y_test} \n\n\n")
```

```
After Splitting, The Train And The Test Set:-
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. ]]


x_test:-

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

y_train:-

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

y_test:-

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

Training The Simple Linear Regression Model On The Training Set

In [32]:
```
# Training the Simple Linear Regression model on the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(x_train, y_train)
```

Out[32]:
```
LinearRegression()
```

Predicting The Test Set Results

In [33]:
```
# Predicting the Test set results
y_pred = regressor.predict(x_test)

print("After Predicting Salaries For The Test Set:-")
print(f"x_test:-\n\n{x_test} \n")
print(f"y_pred:-\n\n{y_pred} \n")
print(f"y_test:-\n\n{y_test} \n\n\n")
```

```
After Predicting Salaries For The Test Set:-
x_test:-

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

y_pred:-

[ 40835.10590871 123079.39940819  65134.55626083  63265.36777221
 115602.64545369 108125.8914992  116537.23969801  64199.96201652
  76349.68719258 100649.1375447 ]

y_test:-

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

Visualising The Training Set Sesults

In [34]:
```
# Visualising the Training set results
plt.scatter(x_train, y_train, color = 'red')
plt.plot(x_train, regressor.predict(x_train), color = 'green')
plt.title('Salary VS. Years of Experience (TRAINING SET)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

Visualising The Test Set Results

In [35]:
```
# Visualising the Test set results
plt.scatter(x_test, y_test, color = 'red')
plt.plot(x_train, regressor.predict(x_train), color = 'green')
# plt.plot(x_test, regressor.predict(x_test), color = 'green') => WILL ALSO GIVE THE SAME
# RESULT AS THE REGRESSION LINE THAT WE GET IS ACTUALLY RESULTING FROM A UNIQUE EQUATION
plt.title('Salary VS. Years of Experience (TEST SET)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

Model Evaluation

In [37]:
```
#Model Evaluation
# Mean-Squared_Error(MSE)
print(f"MSE:-{me.mean_squared_error(y_test,y_pred)}")
# Root_Mean-Squared_Error(RMSE)
print(f"RMSE:-{np.sqrt(me.mean_squared_error(y_test,y_pred))}")
```

```
MSE:-21026037.329511296
RMSE:-4585.4157204675885
```

PRACTICAL- 3

AIM: IMPLEMENT LOGISTIC REGRESSION IN R OR PYTHON.

Importing The Libraries

In [70]:
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import metrics as me
```

Importing The Dataset

In [71]:
```
# Importing the dataset
dataset = pd.read_csv('Practical_3_Data.csv')
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

print(f"{x} \n")
print(f"{y} \n\n\n")
```

```
[[  19 19000]
 [  35 20000]
 [  26 43000]
 [  27 57000]
 [  19 76000]
 [  27 58000]
 [  27 84000]
 [  32 150000]
 [  25 33000]
 [  35 65000]
 [  26 80000]
 [  26 52000]
 [  20 86000]
 [  32 18000]
 [  18 82000]
 [  29 80000]
 [  47 25000]
 [  45 26000]
 [  46 28000]
```

Splitting The Dataset Into The Training Set And Test Set

In [72]:
```
# Splitting the dataset into the Training set and Test set
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("After Splitting, The Train And The Test Set:-")
print(f"x_train:-\n\n{x_train} \n")
print(f"x_test:-\n\n{x_test} \n")
print(f"y_train:-\n\n{y_train} \n")
print(f"y_test:-\n\n{y_test} \n\n\n")
```

```
After Splitting, The Train And The Test Set:-
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]
```

Feature Scaling

In [73]:
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
print("After Scaling, The Train And The Test Set:-")
print(f"x_train:-\n\n{x_train} \n")
print(f"x_test:-\n\n{x_test} \n")
```

```
[-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]
```

Training The Logistic Regression Model On The Training Set

In [74]:
```
# Training the Logistic Regression model on the Training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(x_train, y_train)
```

Out[74]:
```
LogisticRegression(random_state=0)
```

Predicting A New Result

In [75]:
```
# Predicting a new result
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Predicting The Test Set Results

In [76]:
```
# Predicting the Test set results
y_pred = classifier.predict(x_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
print(f"x_test:-\n\n{x_test} \n")
print(f"y_pred:-\n\n{y_pred} \n")
print(f"y_test:-\n\n{y_test} \n\n\n")
```

```
[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]
```

Making The Confusion Matrix

In [77]:
```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(f"CONFUSION MATRIX:")
for i in range(len(cm)):
  print("\t")
  for j in range(len(cm[i])):
    print(cm[i][j],end="\t")
  print()
score=accuracy_score(y_test, y_pred)
print(f"\n\nACCURACY: {score}")
```

```
CONFUSION MATRIX:

65  3

8   24


ACCURACY: 0.89
```

Plotting The Training Set

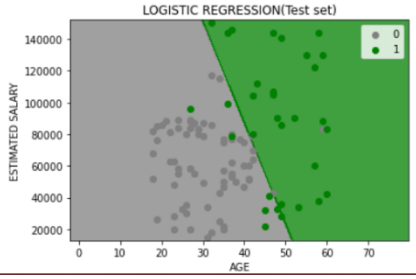
In [12]:
```
# Plotting the Training set
from matplotlib.colors import ListedColormap
x_set, y_set = sc.inverse_transform(x_train), y_train
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 20, stop = x_set[:, 0].max() + 20, step = 0.25),
                     np.arange(start = x_set[:, 1].min() - 2000, stop = x_set[:, 1].max() + 2000, step = 0.25))
plt.contourf(x1, x2, classifier.predict(sc.transform(np.array([x1.ravel(), x2.ravel()]).T)).reshape(x1.shape),
             alpha = 0.75, cmap = ListedColormap(('grey', '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(('grey', 'green'))(i), label = j)
plt.title('LOGISTIC REGRESSION(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 2-D array 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 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```

Plotting The Test Set

In [13]:
```
# Plotting the Test set
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() - 20, stop = x_set[:, 0].max() + 20, step = 0.25),
                     np.arange(start = x_set[:, 1].min() - 2000, stop = x_set[:, 1].max() + 2000, step = 0.25))
plt.contourf(x1, x2, classifier.predict(sc.transform(np.array([x1.ravel(), x2.ravel()]).T)).reshape(x1.shape),
             alpha = 0.75, cmap = ListedColormap(('grey', '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(('grey', 'green'))(i), label = j)
plt.title('LOGISTIC REGRESSION(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 the *color* keyword-argument or provide a 2-D array 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 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```



PRACTICAL -4

AIM: IMPLEMENT SVM CLASSIFIER IN R OR PYTHON

Importing The Libraries

In [55]:
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import metrics as me
```

Importing The Dataset

In [56]:
```
# Importing the dataset
dataset = pd.read_csv('Practical_4_Data.csv')
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

print(f"{x} \n")
print(f"{y} \n\n\n")
```

```
[[ 19 19000]
 [ 35 20000]
 [ 26 43000]
 [ 27 57000]
 [ 19 76000]
 [ 27 58000]
 [ 27 84000]
 [ 32 150000]
 [ 25 33000]
 [ 35 65000]
 [ 26 80000]
 [ 26 52000]
 [ 20 86000]
 [ 32 18000]
 [ 18 82000]
 [ 29 80000]
 [ 47 25000]
 [ 45 26000]
 [ 46 28000]
```

Splitting The Dataset Into The Training Set And Test Set

In [57]:
```
# Splitting the dataset into the Training set and Test set
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("After Splitting, The Train And The Test Set:-")
print(f"x_train:-\n\n{x_train} \n")
print(f"x_test:-\n\n{x_test} \n")
print(f"y_train:-\n\n{y_train} \n")
print(f"y_test:-\n\n{y_test} \n\n\n")
```

```
After Splitting, The Train And The Test Set:-
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]
```

Feature Scaling

In [58]:
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
print("After Scaling, The Train And The Test Set:-")
print(f"x_train:-\n\n{x_train} \n")
print(f"x_test:-\n\n{x_test} \n")
```

```
After Scaling, The Train And The Test Set:-
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]
```

Training The SVM Model On The Training Set

In [59]:
```
# Training the SVM model on the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(x_train, y_train)
```

Out[59]:
```
SVC(kernel='linear', random_state=0)
```

Predicting A New Result

In [60]:
```
# Predicting a new result
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Predicting The Test Set Results

In [61]:
```
# Predicting the Test set results
y_pred = classifier.predict(x_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
print(f"x_test:-\n\n{x_test} \n")
print(f"y_pred:-\n\n{y_pred} \n")
print(f"y_test:-\n\n{y_test} \n\n\n")
```

```
[[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]
```

Making The Confusion Matrix

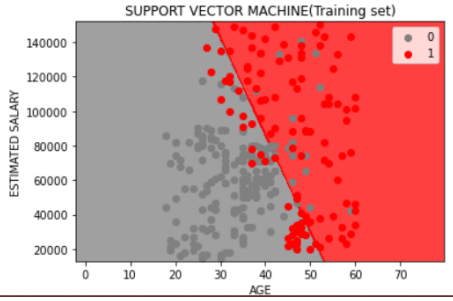
In [62]:
```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(f"CONFUSION MATRIX:")
for i in range(len(cm)):
  print("\t")
  for j in range(len(cm[i])):
    print(cm[i][j],end="\t")
  print()
score=accuracy_score(y_test, y_pred)
print(f"\n\nACCURACY: {score}")
```

```
CONFUSION MATRIX:

66	2	

8	24	


ACCURACY: 0.9
```

Plotting The Training Set

In [13]:
```
# Plotting the Training set
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() - 20, stop = x_set[:, 0].max() + 20, step = 0.25),
                     np.arange(start = x_set[:, 1].min() - 2000, stop = x_set[:, 1].max() + 2000, step = 0.25))
plt.contourf(x1, x2, classifier.predict(sc.transform(np.array([x1.ravel(), x2.ravel()]).T)).reshape(x1.shape),
             alpha = 0.75, cmap = ListedColormap(('grey', 'red')))
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(('grey', 'red'))(i), label = j)
plt.title('SUPPORT VECTOR MACHINE(Training set)')
plt.xlabel('AGE')
plt.ylabel('ESTIMATED SALARY')
plt.legend()
plt.show()
```

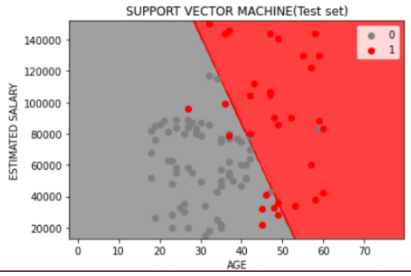
```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y
*.  Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA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



Plotting The Test Set

In [14]:
```
# Plotting the Test set
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() - 20, stop = x_set[:, 0].max() + 20, step = 0.25),
                     np.arange(start = x_set[:, 1].min() - 2000, stop = x_set[:, 1].max() + 2000, step = 0.25))
plt.contourf(x1, x2, classifier.predict(sc.transform(np.array([x1.ravel(), x2.ravel()]).T)).reshape(x1.shape),
             alpha = 0.75, cmap = ListedColormap(('grey', 'red')))
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(('grey', 'red'))(i), label = j)
plt.title('SUPPORT VECTOR MACHINE(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 the *color* keyword-argument or provide a 2-D array 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 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```



PRACTICAL -5

AIM: IMPLEMENT K-NN CLASSIFIER IN R OR PYTHON.

Importing The Libraries

In [2]:
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import metrics as me
```

Importing The Dataset

In [8]:
```
# Importing the dataset
dataset = pd.read_csv('Practical_5_Data.csv')
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

print(f"{x} \n")
print(f"{y} \n\n\n")
```

```
[[  19  19000]
 [  35  20000]
 [  26  43000]
 [  27  57000]
 [  19  76000]
 [  27  58000]
 [  27  84000]
 [  32 150000]
 [  25  33000]
 [  35  65000]
 [  26  80000]
 [  26  52000]
 [  20  86000]
 [  32  18000]
 [  18  82000]
 [  29  80000]
 [  47  25000]
 [  45  26000]
 [  46  28000]
```

Splitting The Dataset Into The Training Set And Test Set

In [4]:
```
# Splitting the dataset into the Training set and Test set
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("After Splitting, The Train And The Test Set:-")
print(f"x_train:-\n\n{x_train} \n")
print(f"x_test:-\n\n{x_test} \n")
print(f"y_train:-\n\n{y_train} \n")
print(f"y_test:-\n\n{y_test} \n\n\n")
```

```
[  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]
```

Feature Scaling

In [5]:
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
print("After Scaling, The Train And The Test Set:-")
print(f"x_train:-\n\n{x_train} \n")
print(f"x_test:-\n\n{x_test} \n")
```

```
After Scaling, The Train And The Test Set:-
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]
```

Training The K-NN Model On The Training Set

In [6]:
```
# Training the K-NN model on the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(x_train, y_train)
```

Out[6]:
```
KNeighborsClassifier()
```

Predicting A New Result

In [7]:
```
# Predicting a new result
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Predicting The Test Set Results

In [8]:
```
# Predicting the Test set results
y_pred = classifier.predict(x_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
print(f"x_test:-\n\n{x_test} \n")
print(f"y_pred:-\n\n{y_pred} \n")
print(f"y_test:-\n\n{y_test} \n\n\n")
```

```
[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]
```

Making The Confusion Matrix

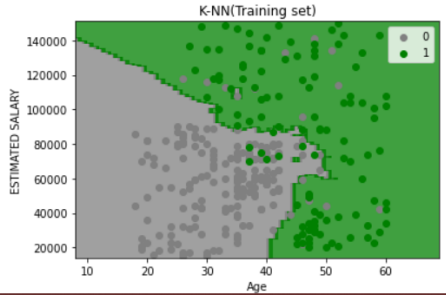
In [9]:
```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(f"CONFUSION MATRIX:")
for i in range(len(cm)):
  print("\t")
  for j in range(len(cm[i])):
    print(cm[i][j],end="\t")
  print()
score=accuracy_score(y_test, y_pred)
print(f"\n\nACCURACY: {score}")
```

```
CONFUSION MATRIX:

64	4	

3	29	


ACCURACY: 0.93
```

Plotting The Training Set

In [10]:
```
# Plotting the Training set
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() + 10, step = 1),
                     np.arange(start = x_set[:, 1].min() - 1000, stop = x_set[:, 1].max() + 1000, step = 1))
plt.contourf(x1, x2, classifier.predict(sc.transform(np.array([x1.ravel(), x2.ravel()]).T)).reshape(x1.shape),
             alpha = 0.75, cmap = ListedColormap(('grey', '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(('grey', 'green'))(i), label = j)
plt.title('K-NN(Training set)')
plt.xlabel('Age')
plt.ylabel('ESTIMATED SALARY')
plt.legend()
plt.show()
```

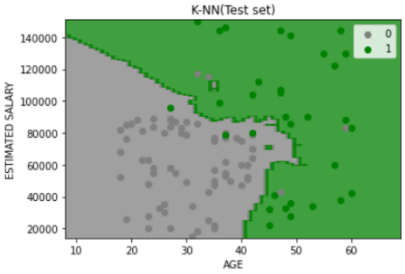
```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y
*.  Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA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



Plotting The Training Set

In [11]:
```
# Plotting the Test set
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() + 10, step = 1),
                     np.arange(start = x_set[:, 1].min() - 1000, stop = x_set[:, 1].max() + 1000, step = 1))
plt.contourf(x1, x2, classifier.predict(sc.transform(np.array([x1.ravel(), x2.ravel()]).T)).reshape(x1.shape),
             alpha = 0.75, cmap = ListedColormap(('grey', '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(('grey', 'green'))(i), label = j)
plt.title('K-NN(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 the *color* keyword-argument or provide a 2-D array 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 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```



PRACTICAL- 6

AIM: STUDY AND IMPLEMENT K-FOLD CROSS VALIDATION AND ROC

Importing The Libraries

In [39]:
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import metrics as me
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

Importing The Dataset

In [40]:
```
# Importing the dataset
dataset = pd.read_csv('Practical_6_Data.csv')
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

print(f"{x} \n")
print(f"{y} \n\n\n")
```

```
[[   19 19000]
 [   35 20000]
 [   26 43000]
 [   27 57000]
 [   19 76000]
 [   27 58000]
 [   27 84000]
 [   32 150000]
 [   25 33000]
 [   35 65000]
 [   26 80000]
 [   26 52000]
 [   20 86000]
 [   32 18000]
 [   18 82000]
 [   29 80000]
 [   47 25000]
 [   45 26000]
 [   46 28000]
```

Perform Cross Validation And Evaluate The Model

In [41]:
```
# Perform Cross Validation And Evaluate The Model

cv = KFold(n_splits=10, random_state=1, shuffle=True)
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
scores = cross_val_score(classifier, x, y, scoring='accuracy', cv=cv, n_jobs=-1)
for train_index, test_index in cv.split(x):
  print(f"TRAIN: \n{train_index}\nTEST: \n{test_index}\n\n")
  x_train, x_test = x[train_index], x[test_index]
  y_train, y_test = y[train_index], y[test_index]
```

```
TRAIN:
[ 0  1  2  3  7  8  9 10 11 12 13 14 15 16 18 19 20 21
 22 23 24 25 26 27 28 30 31 32 33 34 35 36 37 38 39 40
 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 59
 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77
 79 82 83 84 85 86 87 88 89 90 91 93 94 95 96 97 98 99
100 101 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118
119 120 121 123 124 126 127 128 129 130 131 132 133 134 135 136 137 138
140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157
158 159 160 161 162 163 164 166 167 168 169 170 171 173 175 176 177 178
179 180 181 182 183 184 185 186 187 189 190 191 192 193 194 195 196 198
199 200 202 203 204 205 206 208 209 210 211 212 213 215 216 217 218 219
220 221 222 224 225 226 228 229 230 231 232 233 234 235 236 237 238 239
240 241 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258
259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 276 277
278 279 280 281 282 283 285 286 287 288 289 290 292 293 294 295 296 297
298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315
316 317 318 319 320 322 323 325 326 327 329 330 331 332 333 334 335 336
337 338 340 341 343 344 345 346 347 348 349 350 351 352 353 354 355 356
```

In [41]:
```
# Perform Cross Validation And Evaluate The Model

cv = KFold(n_splits=10, random_state=1, shuffle=True)
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
scores = cross_val_score(classifier, x, y, scoring='accuracy', cv=cv, n_jobs=-1)
for train_index, test_index in cv.split(x):
  print(f"TRAIN: \n{train_index}\nTEST: \n{test_index}\n\n")
  x_train, x_test = x[train_index], x[test_index]
  y_train, y_test = y[train_index], y[test_index]
```

```
100 101 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118
119 120 121 123 124 126 127 128 129 130 131 132 133 134 135 136 137 138
140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157
158 159 160 161 162 163 164 166 167 168 169 170 171 173 175 176 177 178
179 180 181 182 183 184 185 186 187 189 190 191 192 193 194 195 196 198
199 200 202 203 204 205 206 208 209 210 211 212 213 215 216 217 218 219
220 221 222 224 225 226 228 229 230 231 232 233 234 235 236 237 238 239
240 241 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258
259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 276 277
278 279 280 281 282 283 285 286 287 288 289 290 292 293 294 295 296 297
298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315
316 317 318 319 320 322 323 325 326 327 329 330 331 332 333 334 335 336
337 338 340 341 343 344 345 346 347 348 349 350 351 352 353 354 355 356
357 358 359 360 362 363 364 365 367 368 369 370 373 374 375 376 377 378
379 380 381 383 384 385 386 387 388 389 390 391 392 393 394 395 396 399]
TEST:
[ 4  5  6 17 29 58 78 80 81 92 102 122 125 139 165 172 174 188
197 201 207 214 223 227 242 275 284 291 321 324 328 339 342 361 366 371
372 382 397 398]
```

Accuracy Of The Model

In [42]:
```
# Accuracy Of The Model
print(f'Accuracy Of The Model Using K-Fold Cross Validation: {np.mean(scores)*100}%')
```

```
Accuracy Of The Model Using K-Fold Cross Validation: 64.25%
```

In [ ]:

PRACTICAL-7

AIM: Implement Decision Tree Classifier in R or python

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('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 [0]: from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

```
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]
```

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]
```

In [7]:
```
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 [0]:
```
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]
```

```
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]
```

Training the Decision Tree Classification model on the Training set

```
In [11]: from sklearn.tree import DecisionTreeClassifier
         classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
         classifier.fit(X_train, y_train)
```

```
Out[11]: DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                                max_depth=None, max_features=None, max_leaf_nodes=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, presort='deprecated',
                                random_state=0, splitter='best')
```

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

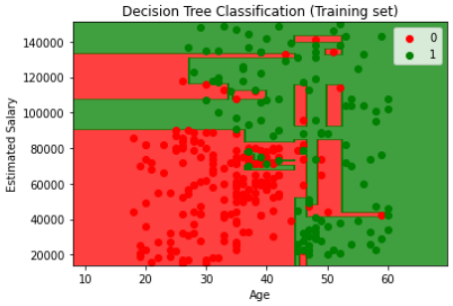
```
[[62  6]
 [ 3 29]]
```

Out[14]:
```
0.91
```

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() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             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', 'green'))(i), label = j)
plt.title('Decision Tree Classification (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

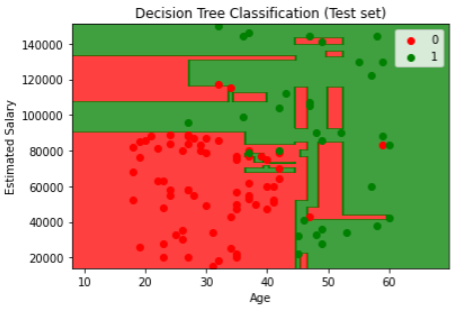
```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks 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() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             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', 'green'))(i), label = j)
plt.title('Decision Tree Classification (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.
```



PRACTICAL-8

AIM: Study and Implement various Ensemble method of classifier: Bagging, Boosting andStacking.

Importing Libraries

In [1]:
```
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.ensemble import BaggingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from matplotlib import pyplot
from sklearn.ensemble import GradientBoostingClassifier
```

Bagging

In [2]:
```
#Bagging
X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=5, random_state=5)
```

In [3]:
```
model = BaggingClassifier()
```

In [4]:
```
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')
print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
Accuracy: 0.853 (0.042)
```

Bagging

In [5]:
```
#Boosting
from sklearn.ensemble import GradientBoostingClassifier
X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=5, random_state=7)
```

In [8]:
```
model = GradientBoostingClassifier()
```

In [7]:
```
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
```

In [9]:
```
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
```

In [10]:
```
print('Mean Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
Mean Accuracy: 0.900 (0.030)
```

In [11]:
```
def evaluate_model(model, X, y):
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')
    return scores
```

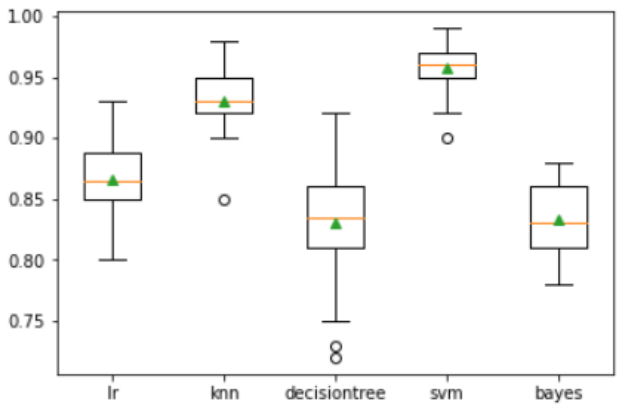
Stacking

In [12]:
```
#Stacking

X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=5, random_state=1)
```

In [14]:
```
models = dict()
models['lr'] = LogisticRegression()
models['knn'] = KNeighborsClassifier()
models['decisiontree'] = DecisionTreeClassifier()
models['svm'] = SVC()
models['bayes'] = GaussianNB()

results, names = list(), list()
for name, model in models.items():
    scores = evaluate_model(model, X, y)
    results.append(scores)
    names.append(name)
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
```

```
>lr 0.866 (0.029)
>knn 0.931 (0.025)
>decisiontree 0.830 (0.048)
>svm 0.957 (0.020)
>bayes 0.833 (0.031)
```

In [15]:
```
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()
```



PRACTICAL-9

AIM: Implement various Clustering algorithm in R and python

1.K-Means Clustering

Importing the libraries

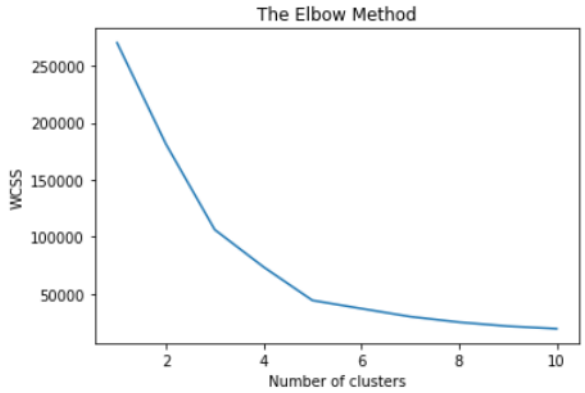
```
In [0]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
```

Importing the dataset

```
In [0]: dataset = pd.read_csv('Mall_Customers.csv')
        X = dataset.iloc[:, [3, 4]].values
```

Using the elbow method to find the optimal number of clusters

In [3]:
```
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()
```



Training the K-Means model on the dataset

In [0]:
```
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
```

Visualising the clusters

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

2. **Hierarchical** Clustering

Importing the libraries

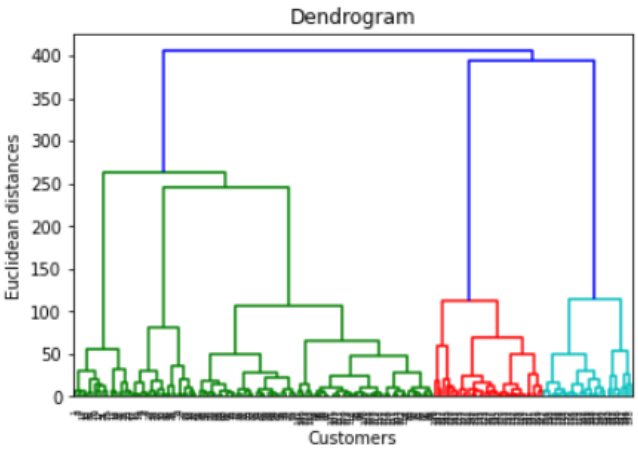
```
In [0]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
```

Importing the dataset

```
In [0]: dataset = pd.read_csv('Mall_Customers.csv')
        X = dataset.iloc[:, [3, 4]].values
```

Using the dendrogram to find the optimal number of clusters

In [3]:
```
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 [0]:
```
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(X)
```

Visualising the clusters

In [5]:
```
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()
```