

▼ Classification Logistic Regression

```
1 # Importing the libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import pandas as pd

1 # Importing the dataset
2 dataset = pd.read_csv(r'/content/drive/MyDrive/Social_Network_Ads.csv')
3 X = dataset.iloc[:, :-1].values
4 y = dataset.iloc[:, -1].values
```

Double-click (or enter) to edit

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

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

```
1 # Splitting the dataset into the Training set and Test set
2 from sklearn.model_selection import train_test_split
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
4 print(X_train)
5 print(y_train)
6 print(X_test)
7 print(y_test)
8
```

```
[ 33 69000]
[ 20 82000]
[ 31 74000]
[ 42 80000]
[ 35 72000]
[ 33 149000]
[ 40 71000]
```

```
[ 31 10000]
[ 43 129000]
[ 59 76000]
[ 18 44000]
[ 36 118000]
[ 42 90000]
[ 47 30000]
[ 26 43000]
[ 40 78000]
[ 46 59000]
[ 59 42000]
```

```
1 # Feature Scaling
2 from sklearn.preprocessing import StandardScaler
3 sc = StandardScaler()
4 X_train = sc.fit_transform(X_train)
5 X_test = sc.transform(X_test)
6 print(X_train)
7 print(X_test)
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761 1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761 1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972 0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
 [-0.21060859 -0.19087153]
 [-0.30964085 -1.29261101]
 [-0.30964085 -0.5677824 ]
 [ 0.38358493 0.09905991]
 [ 0.8787462 -0.59677555]
 [ 2.06713324 -1.17663843]
 [ 1.07681071 -0.13288524]
 [ 0.68068169 1.78066227]
 [-0.70576986 0.56295021]
 [ 0.77971394 0.35999821]
 [ 0.8787462 -0.53878926]
 [-1.20093113 -1.58254245]
 [ 2.1661655 0.93986109]
 [-0.01254409 1.22979253]
 [ 0.18552042 1.08482681]
 [ 0.38358493 -0.48080297]
 [-0.30964085 -0.30684411]
 [ 0.97777845 -0.8287207 ]
 [ 0.97777845 1.8676417 ]
 [-0.01254409 1.25878567]
 [-0.90383437 2.27354572]
 [-1.20093113 -1.58254245]
 [ 2.1661655 -0.79972756]
 [-1.39899564 -1.46656987]
 [ 0.38358493 2.30253886]
 [ 0.77971394 0.76590222]
 [-1.00286662 -0.30684411]
 [ 0.08648817 0.76590222]
 [-1.00286662 0.56295021]
 [ 0.28455268 0.07006676]
 [ 0.68068169 -1.26361786]
 [-0.50770535 -0.01691267]
 [-1.79512465 0.35999821]
 [-0.70576986 0.12805305]
 [ 0.38358493 0.30201192]
 [-0.30964085 0.07006676]
 [-0.50770535 2.30253886]
 [ 0.18552042 0.04107362]
 [ 1.27487521 2.21555943]
 [ 0.77971394 0.27301877]
 [-0.30964085 0.1570462 ]
 [-0.01254409 -0.53878926]
 [-0.21060859 0.1570462 ]
 [-0.11157634 0.24402563]
 [-0.01254409 -0.24885782]
 [ 2.1661655 1.11381995]
 [-1.79512465 0.35999821]
 [ 1.86906873 0.12805305]
 [ 0.38358493 -0.13288524]
```

```
1 # Training the Logistic Regression model on the Training set
2 from sklearn.linear_model import LogisticRegression
3 classifier = LogisticRegression(random_state = 0)
4 classifier.fit(X_train, y_train)
```

```
▼ LogisticRegression
LogisticRegression(random_state=0)
```

```
1 # Predicting a new result
2 print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

```
1 # Predicting the Test set results
2 y_pred = classifier.predict(X_test)
3 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]
 [1 1]
 [0 0]
 [1 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 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]
 [0 0]
 [1 1]
 [1 1]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 1]
 [0 0]
 [0 0]
```

```

1 # Making the Confusion Matrix
2 from sklearn.metrics import confusion_matrix, accuracy_score
3 cm = confusion_matrix(y_test, y_pred)
4 print(cm)
5 accuracy_score(y_test, y_pred)

```

```

[[65  3]
 [ 8 24]]
0.89

```

```

1 # Visualising the Training set results
2 from matplotlib.colors import ListedColormap
3 X_set, y_set = sc.inverse_transform(X_train), y_train
4 X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
5                       np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
6 plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
7              alpha = 0.75, cmap = ListedColormap(('red', 'green')))
8 plt.xlim(X1.min(), X1.max())
9 plt.ylim(X2.min(), X2.max())
10 for i, j in enumerate(np.unique(y_set)):
11     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
12 plt.title('Logistic Regression (Training set)')
13 plt.xlabel('Age')
14 plt.ylabel('Estimated Salary')
15 plt.legend()
16 plt.show()

```

```

<ipython-input-14-73b57679e135>:11: UserWarning: *c* argument looks like a single numerical
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'gr

```

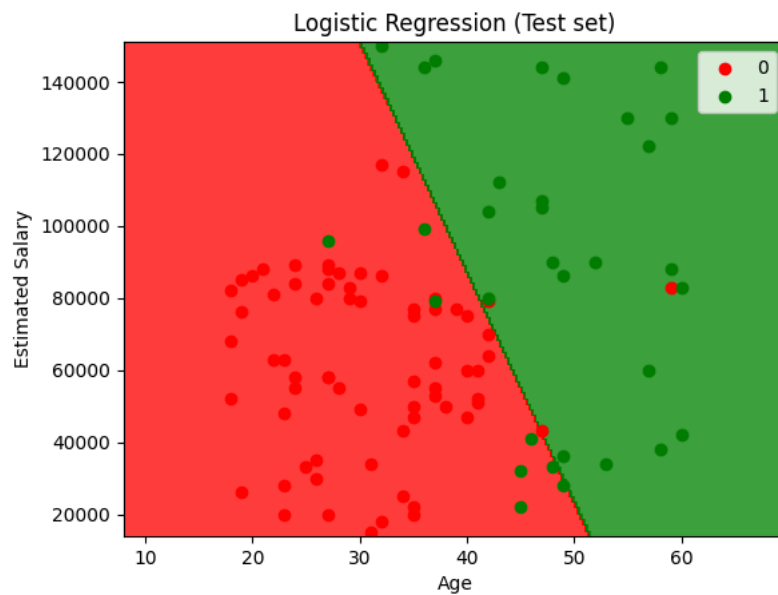


```

1 # Visualising the Test set results
2 from matplotlib.colors import ListedColormap
3 X_set, y_set = sc.inverse_transform(X_test), y_test
4 X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
5                       np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
6 plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
7              alpha = 0.75, cmap = ListedColormap(('red', 'green')))
8 plt.xlim(X1.min(), X1.max())
9 plt.ylim(X2.min(), X2.max())
10 for i, j in enumerate(np.unique(y_set)):
11     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
12 plt.title('Logistic Regression (Test set)')
13 plt.xlabel('Age')
14 plt.ylabel('Estimated Salary')
15 plt.legend()
16 plt.show()

```

```
<ipython-input-15-4a24fc64ffb4>:11: UserWarning: *c* argument looks like a single numeri
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(['red', 'gr
```



K -NN -- Classification K nearest neighbour

```
1 # Training the K-NN model on the Training set
2 from sklearn.neighbors import KNeighborsClassifier
3 classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
4 classifier.fit(X_train, y_train)
```

```
▼ KNeighborsClassifier
KNeighborsClassifier()
```

```
1 # Predicting a new result
2 print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

```
1 # Predicting the Test set results
2 y_pred = classifier.predict(X_test)
3 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]
 [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]
[1 1]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[1 1]
[0 0]
[1 1]
[1 1]
[0 0]
[0 0]
[1 0]
[1 1]
[1 1]
[0 0]
[0 0]
[1 0]
[1 1]
[1 1]
[0 0]
^- ^-

```

```

1 # Making the Confusion Matrix
2 from sklearn.metrics import confusion_matrix, accuracy_score
3 cm = confusion_matrix(y_test, y_pred)
4 print(cm)
5 accuracy_score(y_test, y_pred)

```

```

[[64  4]
 [ 3 29]]
0.93

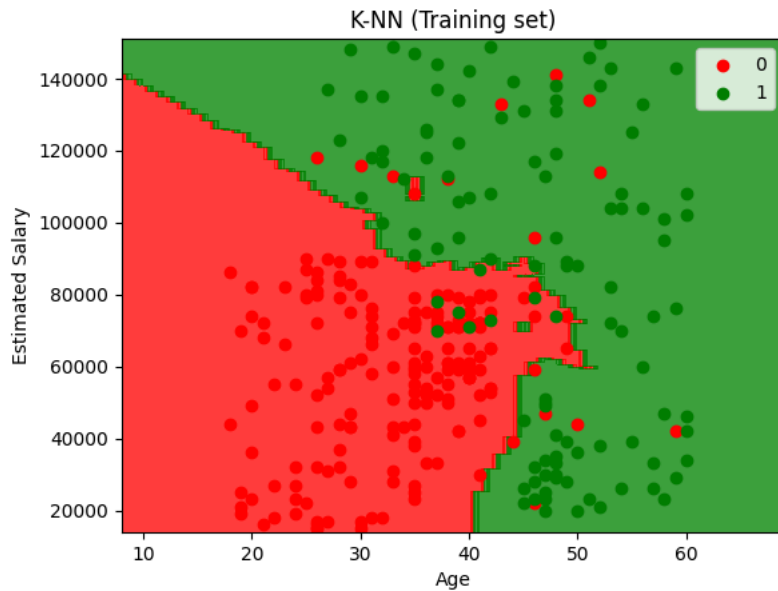
```

```

1 from matplotlib.colors import ListedColormap
2 X_set, y_set = sc.inverse_transform(X_train), y_train
3 X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 1),
4                       np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 1))
5 plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()])).T)).reshape(X1.shape),
6              alpha = 0.75, cmap = ListedColormap(('red', 'green')))
7 plt.xlim(X1.min(), X1.max())
8 plt.ylim(X2.min(), X2.max())
9 for i, j in enumerate(np.unique(y_set)):
10     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
11 plt.title('K-NN (Training set)')
12 plt.xlabel('Age')
13 plt.ylabel('Estimated Salary')
14 plt.legend()
15 plt.show()

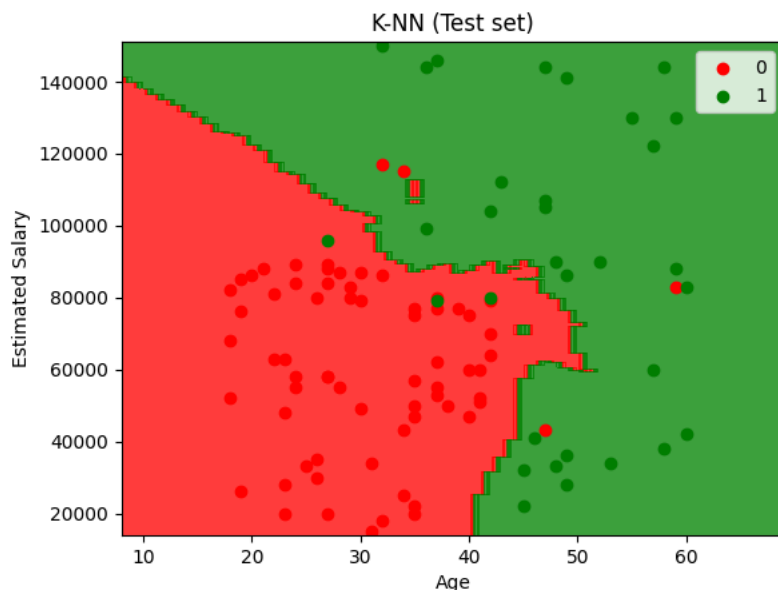
```

```
<ipython-input-20-9061e2cf8fe3>:10: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
```



```
1 # Visualising the Test set results
2 from matplotlib.colors import ListedColormap
3 X_set, y_set = sc.inverse_transform(X_test), y_test
4 X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 1),
5                       np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 1))
6 plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()])).T).reshape(X1.shape),
7             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
8 plt.xlim(X1.min(), X1.max())
9 plt.ylim(X2.min(), X2.max())
10 for i, j in enumerate(np.unique(y_set)):
11     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
12 plt.title('K-NN (Test set)')
13 plt.xlabel('Age')
14 plt.ylabel('Estimated Salary')
15 plt.legend()
16 plt.show()
```

```
<ipython-input-21-8ae8de7a7cff>:11: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
```



Support Vector Machine (SVM) Algorithm

```

1 # Importing the libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import pandas as pd

1 # Importing the dataset
2 dataset = pd.read_csv(r'/content/drive/MyDrive/Social_Network_Ads.csv')
3 X = dataset.iloc[:, :-1].values
4 y = dataset.iloc[:, -1].values

1 # Training the SVM model on the Training set
2 from sklearn.svm import SVC
3 classifier = SVC(kernel = 'linear', random_state = 0)
4 classifier.fit(X_train, y_train)
5



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



1 # Predicting a new result
2 print(classifier.predict(sc.transform([[30,87000]])))

[0]

1 # Predicting the Test set results
2 y_pred = classifier.predict(X_test)
3 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]
 [1 1]
 [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]
 [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]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [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]

```

```

1 # Making the Confusion Matrix
2 from sklearn.metrics import confusion_matrix, accuracy_score
3 cm = confusion_matrix(y_test, y_pred)
4 print(cm)
5 accuracy_score(y_test, y_pred)

```

```

[[66  2]
 [ 8 24]]
0.9

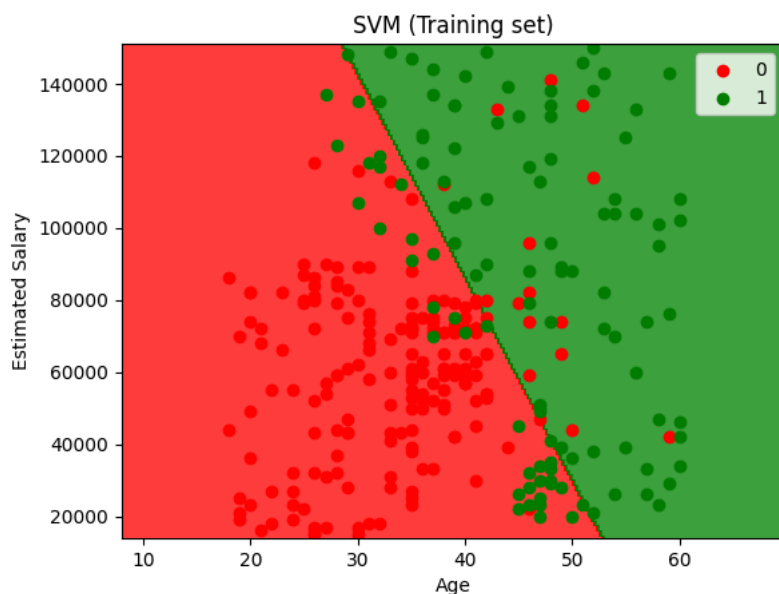
```

```

1 # Visualising the Training set results
2 from matplotlib.colors import ListedColormap
3 X_set, y_set = sc.inverse_transform(X_train, y_train)
4 X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
5                       np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
6 plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()])).T).reshape(X1.shape),
7              alpha = 0.75, cmap = ListedColormap(('red', 'green')))
8 plt.xlim(X1.min(), X1.max())
9 plt.ylim(X2.min(), X2.max())
10 for i, j in enumerate(np.unique(y_set)):
11     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
12 plt.title('SVM (Training set)')
13 plt.xlabel('Age')
14 plt.ylabel('Estimated Salary')
15 plt.legend()
16 plt.show()

```

<ipython-input-28-501d2325cffa>:11: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)

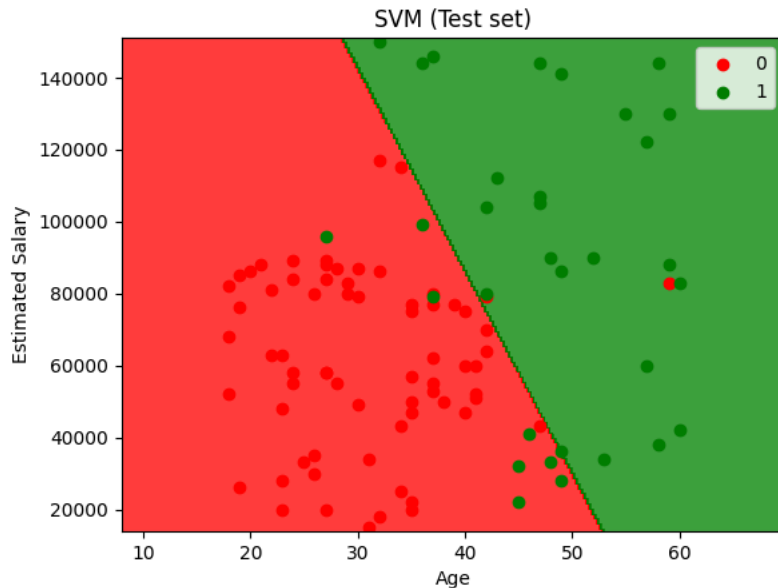


```

1 # Visualising the Test set results
2 from matplotlib.colors import ListedColormap
3 X_set, y_set = sc.inverse_transform(X_test), y_test
4 X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
5                       np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
6 plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()])).T)).reshape(X1.shape),
7             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
8 plt.xlim(X1.min(), X1.max())
9 plt.ylim(X2.min(), X2.max())
10 for i, j in enumerate(np.unique(y_set)):
11     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
12 plt.title('SVM (Test set)')
13 plt.xlabel('Age')
14 plt.ylabel('Estimated Salary')
15 plt.legend()
16 plt.show()

```

<ipython-input-29-8c7ca6815161>:11: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided
 plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)



Kernel SVM

```

1 # Importing the libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import pandas as pd

```

```

1 # Importing the dataset
2 dataset = pd.read_csv('Social_Network_Ads.csv')
3 X = dataset.iloc[:, :-1].values
4 y = dataset.iloc[:, -1].values

```

```

1 # Splitting the dataset into the Training set and Test set
2 from sklearn.model_selection import train_test_split
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
4 print(X_train)
5 print(y_train)
6 print(X_test)
7 print(y_test)

```

```

[[ 44  39000]
 [ 32 120000]
 [ 38  50000]
 [ 32 135000]
 [ 52  21000]
 [ 53 104000]
 [ 39  42000]
 [ 38  61000]
 [ 36  50000]
 [ 36  63000]

```

```
[ 35 25000]
[ 35 50000]
[ 42 73000]
[ 47 49000]
[ 59 29000]
[ 49 65000]
[ 45 131000]
[ 31 89000]
[ 46 82000]
[ 47 51000]
[ 26 15000]
[ 60 102000]
[ 38 112000]
[ 40 107000]
[ 42 53000]
[ 35 59000]
[ 48 41000]
[ 48 134000]
[ 38 113000]
[ 29 148000]
[ 26 15000]
[ 60 42000]
[ 24 19000]
[ 42 149000]
[ 46 96000]
[ 28 59000]
[ 39 96000]
[ 28 89000]
[ 41 72000]
[ 45 26000]
[ 33 69000]
[ 20 82000]
[ 31 74000]
[ 42 80000]
[ 35 72000]
[ 33 149000]
[ 40 71000]
[ 51 146000]
[ 46 79000]
[ 35 75000]
[ 38 51000]
[ 36 75000]
[ 37 78000]
[ 38 61000]
[ 60 108000]
[ 20 82000]
[ 57 74000]
[ 42 65000]
```

```
1 # Feature Scaling
2 from sklearn.preprocessing import StandardScaler
3 sc = StandardScaler()
4 X_train = sc.fit_transform(X_train)
5 X_test = sc.transform(X_test)
6 print(X_train)
7 print(X_test)
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
 [-0.21060859 -0.19087153]
 [-0.30964085 -1.29261101]
 [-0.30964085 -0.5677824 ]
 [ 0.38358493  0.09905991]
 [ 0.8787462  -0.59677555]
 [ 2.06713324 -1.17663843]
 [ 1.07681071 -0.13288524]
 [ 0.68068169  1.78066227]
 [-0.70576986  0.56295021]
 [ 0.77971394  0.35999821]
 [ 0.8787462  -0.53878926]
 [-1.20093113 -1.58254245]
 [ 2.1661655  0.93986109]
 [-0.01254409  1.22979253]
 [ 0.18552042  1.08482681]
 [ 0.38358493 -0.48080297]
 [-0.30964085 -0.30684411]
 [ 0.97777845 -0.8287207 ]]
```

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

```
1 # Training the Kernel SVM model on the Training set
2 from sklearn.svm import SVC
3 classifier = SVC(kernel = 'rbf', random_state = 0)
4 classifier.fit(X_train, y_train)
```

```
▼ SVC
SVC(random_state=0)
```

```
1 # Predicting a new result
2 print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

```
1 # Predicting the Test set results
2 y_pred = classifier.predict(X_test)
3 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]
[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 1]
[1 1]
[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]
[0 0]
[1 1]
[0 0]
[0 0]
[1 0]
[1 1]
[1 1]
[0 0]
[0 0]

```

```

1 # Making the Confusion Matrix
2 from sklearn.metrics import confusion_matrix, accuracy_score
3 cm = confusion_matrix(y_test, y_pred)
4 print(cm)
5 accuracy_score(y_test, y_pred)

```

```

[[64  4]
 [ 3 29]]
0.93

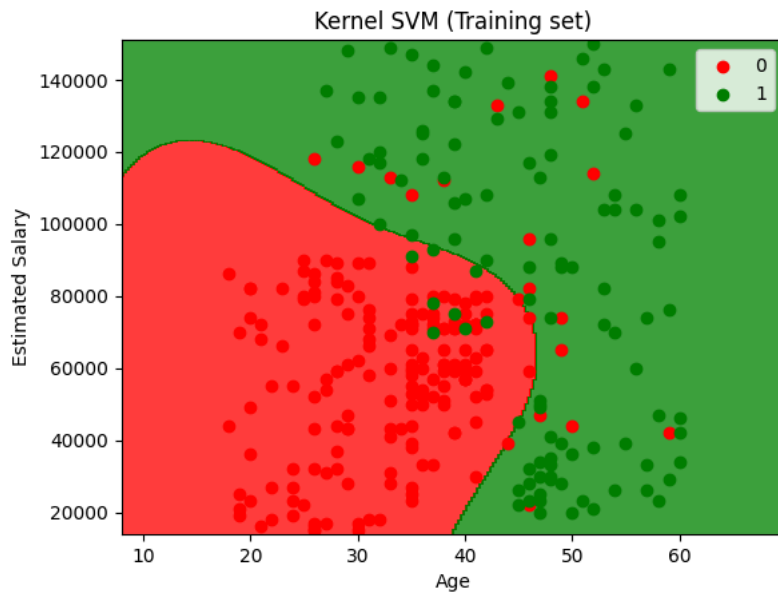
```

```

1 # Visualising the Training set results
2 from matplotlib.colors import ListedColormap
3 X_set, y_set = sc.inverse_transform(X_train), y_train
4 X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
5                      np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
6 plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()])).T)).reshape(X1.shape),
7             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
8 plt.xlim(X1.min(), X1.max())
9 plt.ylim(X2.min(), X2.max())
10 for i, j in enumerate(np.unique(y_set)):
11     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
12 plt.title('Kernel SVM (Training set)')
13 plt.xlabel('Age')
14 plt.ylabel('Estimated Salary')
15 plt.legend()
16 plt.show()

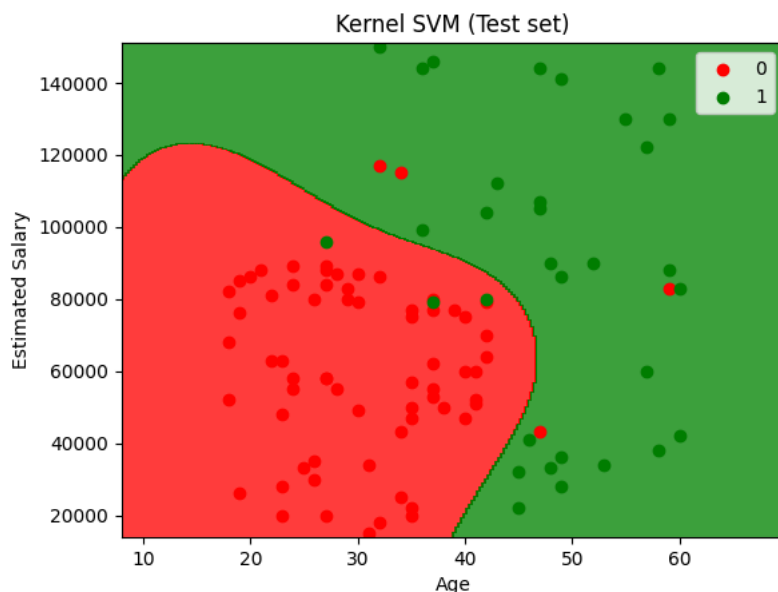
```

```
<ipython-input-9-e13552a9f4e8>:11: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
```



```
1 # Visualising the Test set results
2 from matplotlib.colors import ListedColormap
3 X_set, y_set = sc.inverse_transform(X_test), y_test
4 X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
5                       np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
6 plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()])).T).reshape(X1.shape),
7             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
8 plt.xlim(X1.min(), X1.max())
9 plt.ylim(X2.min(), X2.max())
10 for i, j in enumerate(np.unique(y_set)):
11     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
12 plt.title('Kernel SVM (Test set)')
13 plt.xlabel('Age')
14 plt.ylabel('Estimated Salary')
15 plt.legend()
16 plt.show()
```

```
<ipython-input-10-6fe3d2c99629>:11: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
```



Naive Bayes Algorithm

```
1 # Importing the libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import pandas as pd
```

```
1 # Importing the dataset
2 dataset = pd.read_csv('Social_Network_Ads.csv')
3 X = dataset.iloc[:, :-1].values
4 y = dataset.iloc[:, -1].values
```

```
1 # Splitting the dataset into the Training set and Test set
2 from sklearn.model_selection import train_test_split
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
4 print(X_train)
5 print(y_train)
6 print(X_test)
7 print(y_test)
8
```

```
[[ 44 39000]
 [ 32 120000]
 [ 38 50000]
 [ 32 135000]
 [ 52 21000]
 [ 53 104000]
 [ 39 42000]
 [ 38 61000]
 [ 36 50000]
 [ 36 63000]
 [ 35 25000]
 [ 35 50000]
 [ 42 73000]
 [ 47 49000]
 [ 59 29000]
 [ 49 65000]
 [ 45 131000]
 [ 31 89000]
 [ 46 82000]
 [ 47 51000]
 [ 26 15000]
 [ 60 102000]
 [ 38 112000]
 [ 40 107000]
 [ 42 53000]
 [ 35 59000]
 [ 48 41000]
 [ 48 134000]
 [ 38 113000]
 [ 29 148000]
 [ 26 15000]
 [ 60 42000]
 [ 24 19000]
 [ 42 149000]
 [ 46 96000]
 [ 28 59000]
 [ 39 96000]
 [ 28 89000]
 [ 41 72000]
 [ 45 26000]
 [ 33 69000]
 [ 20 82000]
 [ 31 74000]
 [ 42 80000]
 [ 35 72000]
 [ 33 149000]
 [ 40 71000]
 [ 51 146000]
 [ 46 79000]
 [ 35 75000]
 [ 38 51000]
 [ 36 75000]
 [ 37 78000]
 [ 38 61000]
 [ 60 108000]
 [ 20 82000]
 [ 57 74000]
 [ 42 65000]
```

```

1 # Feature Scaling
2 from sklearn.preprocessing import StandardScaler
3 sc = StandardScaler()
4 X_train = sc.fit_transform(X_train)
5 X_test = sc.transform(X_test)
6 print(X_train)
7 print(X_test)

```

```

[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
 [-0.21060859 -0.19087153]
 [-0.30964085 -1.29261101]
 [-0.30964085 -0.5677824 ]
 [ 0.38358493  0.09905991]
 [ 0.8787462  -0.59677555]
 [ 2.06713324 -1.17663843]
 [ 1.07681071 -0.13288524]
 [ 0.68068169  1.78066227]
 [-0.70576986  0.56295021]
 [ 0.77971394  0.35999821]
 [ 0.8787462  -0.53878926]
 [-1.20093113 -1.58254245]
 [ 2.1661655  0.93986109]
 [-0.01254409  1.22979253]
 [ 0.18552042  1.08482681]
 [ 0.38358493 -0.48080297]
 [-0.30964085 -0.30684411]
 [ 0.97777845 -0.8287207 ]
 [ 0.97777845  1.8676417 ]
 [-0.01254409  1.25878567]
 [-0.90383437  2.27354572]
 [-1.20093113 -1.58254245]
 [ 2.1661655  -0.79972756]
 [-1.39899564 -1.46656987]
 [ 0.38358493  2.30253886]
 [ 0.77971394  0.76590222]
 [-1.00286662 -0.30684411]
 [ 0.08648817  0.76590222]
 [-1.00286662  0.56295021]
 [ 0.28455268  0.07006676]
 [ 0.68068169 -1.26361786]
 [-0.50770535 -0.01691267]
 [-1.79512465  0.35999821]
 [-0.70576986  0.12805305]
 [ 0.38358493  0.30201192]
 [-0.30964085  0.07006676]
 [-0.50770535  2.30253886]
 [ 0.18552042  0.04107362]
 [ 1.27487521  2.21555943]
 [ 0.77971394  0.27301877]
 [-0.30964085  0.1570462 ]
 [-0.01254409 -0.53878926]
 [-0.21060859  0.1570462 ]
 [-0.11157634  0.24402563]
 [-0.01254409 -0.24885782]
 [ 2.1661655  1.11381995]
 [-1.79512465  0.35999821]
 [ 1.86906873  0.12805305]
 [ 0.38358493 -0.13288524]

```

```

1 # Training the Naive Bayes model on the Training set
2 from sklearn.naive_bayes import GaussianNB
3 classifier=GaussianNB ()
4 classifier.fit(X_train, y_train)

```

▼ GaussianNB
GaussianNB()

```

1 # Predicting a new result
2 print(classifier.predict(sc.transform([[30,87000]])))

```

```
[0]
```



```
1 # Predicting the Test set results
2 y_pred = classifier.predict(X_test)
3 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]
 [1 1]
 [0 0]
 [1 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]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [1 1]
 [1 1]
 [0 0]
 [0 0]
 [1 0]
 [1 1]
 [0 1]
 [0 0]
 [0 0]]
```

```
1 # Making the Confusion Matrix
2 from sklearn.metrics import confusion_matrix, accuracy_score
3 cm = confusion_matrix(y_test, y_pred)
4 print(cm)
5 accuracy_score(y_test, y_pred)
```

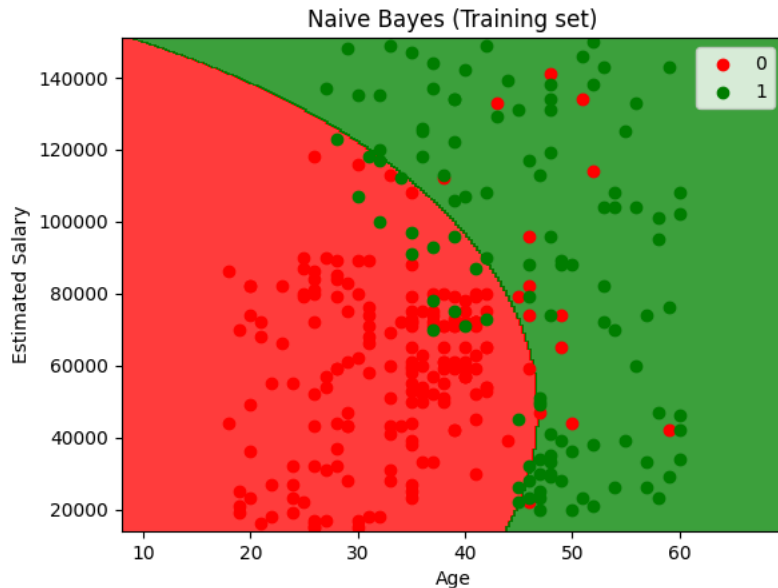
```
[[65  3]
 [ 7 25]]
0.9
```

```

1 # Visualising the Training set results
2 from matplotlib.colors import ListedColormap
3 X_set, y_set = sc.inverse_transform(X_train), y_train
4 X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
5                       np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
6 plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()])).T)).reshape(X1.shape),
7             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
8 plt.xlim(X1.min(), X1.max())
9 plt.ylim(X2.min(), X2.max())
10 for i, j in enumerate(np.unique(y_set)):
11     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
12 plt.title('Naive Bayes (Training set)')
13 plt.xlabel('Age')
14 plt.ylabel('Estimated Salary')
15 plt.legend()
16 plt.show()

```

<ipython-input-9-a3458d358c6d>:11: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)

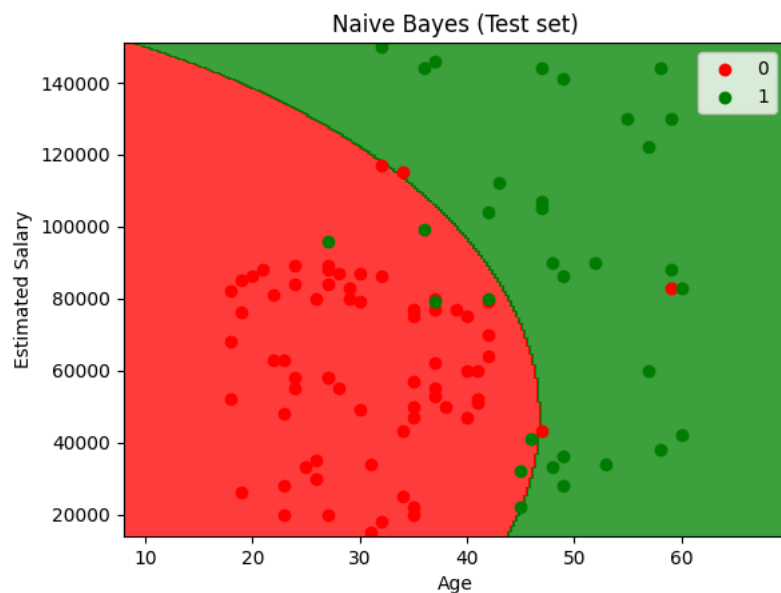


```

1 # Visualising the Test set results
2 from matplotlib.colors import ListedColormap
3 X_set, y_set = sc.inverse_transform(X_test), y_test
4 X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
5                       np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
6 plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()])).T)).reshape(X1.shape),
7             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
8 plt.xlim(X1.min(), X1.max())
9 plt.ylim(X2.min(), X2.max())
10 for i, j in enumerate(np.unique(y_set)):
11     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
12 plt.title('Naive Bayes (Test set)')
13 plt.xlabel('Age')
14 plt.ylabel('Estimated Salary')
15 plt.legend()
16 plt.show()

```

```
<ipython-input-10-aa7eb57ae484>:11: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
```



*Decision Tree Classifier

```
1 # Importing the libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import pandas as pd
```

```
1 # Importing the dataset
2 dataset = pd.read_csv('Social_Network_Ads.csv')
3 X = dataset.iloc[:, :-1].values
4 y = dataset.iloc[:, -1].values
```

```
1 # Splitting the dataset into the Training set and Test set
2 from sklearn.model_selection import train_test_split
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
4 print(X_train)
5 print(y_train)
6 print(X_test)
7 print(y_test)
```

```
[[ 44 39000]
 [ 32 120000]
 [ 38 50000]
 [ 32 135000]
 [ 52 21000]
 [ 53 104000]
 [ 39 42000]
 [ 38 61000]
 [ 36 50000]
 [ 36 63000]
 [ 35 25000]
 [ 35 50000]
 [ 42 73000]
 [ 47 49000]
 [ 59 29000]
 [ 49 65000]
 [ 45 131000]
 [ 31 89000]
 [ 46 82000]
 [ 47 51000]
 [ 26 15000]
 [ 60 102000]
 [ 38 112000]
 [ 40 107000]
 [ 42 53000]
 [ 35 59000]
 [ 48 41000]
 [ 48 134000]
```

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

```
1 # Feature Scaling
2 from sklearn.preprocessing import StandardScaler
3 sc = StandardScaler()
4 X_train = sc.fit_transform(X_train)
5 X_test = sc.transform(X_test)
6 print(X_train)
7 print(X_test)
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
 [-0.21060859 -0.19087153]
 [-0.30964085 -1.29261101]
 [-0.30964085 -0.5677824 ]
 [ 0.38358493  0.09905991]
 [ 0.8787462  -0.59677555]
 [ 2.06713324 -1.17663843]
 [ 1.07681071 -0.13288524]
 [ 0.68068169  1.78066227]
 [-0.70576986  0.56295021]
 [ 0.77971394  0.35999821]
 [ 0.8787462  -0.53878926]
 [-1.20093113 -1.58254245]
 [ 2.1661655  0.93986109]
 [-0.01254409  1.22979253]
 [ 0.18552042  1.08482681]
 [ 0.38358493 -0.48080297]
 [-0.30964085 -0.30684411]
 [ 0.97777845 -0.8287207 ]
 [ 0.97777845  1.8676417 ]
 [-0.01254409  1.25878567]
 [-0.90383437  2.27354572]
 [-1.20093113 -1.58254245]
 [ 2.1661655  -0.79972756]
 [-1.39899564 -1.46656987]
 [ 0.38358493  2.30253886]
 [ 0.77971394  0.76590222]
 [-1.00286662 -0.30684411]
 [ 0.08648817  0.76590222]
 [-1.00286662  0.56295021]
 [ 0.28455268  0.07006676]
 [ 0.68068169 -1.26361786]
 [-0.50770535 -0.01691267]
 [-1.79512465  0.35999821]
 [-0.70576986  0.12805305]
 [ 0.38358493  0.30201192]
 [-0.30964085  0.07006676]
```

```

[-0.50770535  2.30253886]
[ 0.18552042  0.04107362]
[ 1.27487521  2.21555943]
[ 0.77971394  0.27301877]
[-0.30964085  0.1570462 ]
[-0.01254409 -0.53878926]
[-0.21060859  0.1570462 ]
[-0.11157634  0.24402563]
[-0.01254409 -0.24885782]
[ 2.1661655   1.11381995]
[-1.79512465  0.35999821]
[ 1.86906873  0.12805305]
[ 0.38358493 -0.13288524]

```

```

1 from sklearn.tree import DecisionTreeClassifier
2 classifier= DecisionTreeClassifier(criterion='entropy',random_state=0)
3 classifier.fit(X_train, y_train)

```

```

▼ DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=0)

```

```

1 # Predicting a new result
2 print(classifier.predict(sc.transform([[30,8700]])))

```

```
[0]
```

```

1 # Predicting the Test set results
2 y_pred = classifier.predict(X_test)
3 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]
[1 1]
[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]
[0 0]
[0 0]
[1 1]
[0 0]
[0 1]
[0 0]
[0 0]
[0 0]
[0 0]
[0 1]
[1 1]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[1 1]
[0 0]
[1 1]

```

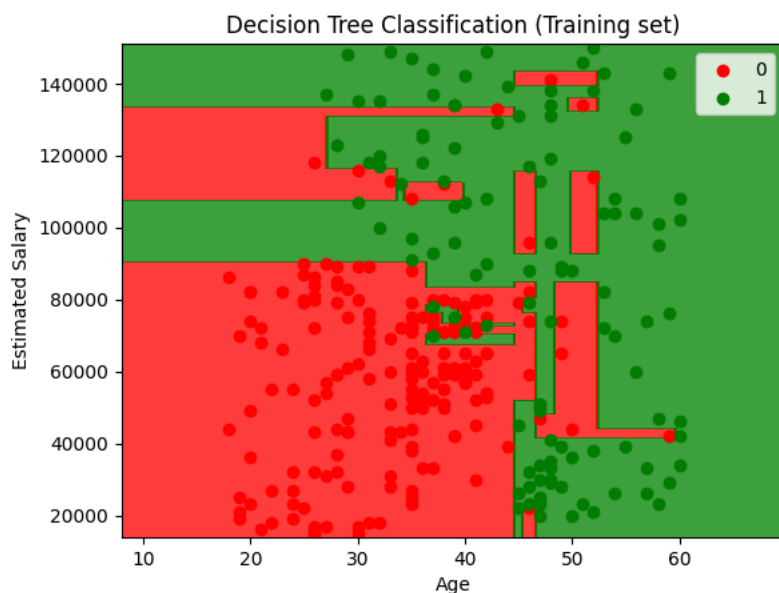
```
[1 1]
[0 0]
[0 0]
[1 0]
[1 1]
[1 1]
[0 0]
[0 0]
```

```
1 # Making the Confusion Matrix
2 from sklearn.metrics import confusion_matrix, accuracy_score
3 cm = confusion_matrix(y_test, y_pred)
4 print(cm)
5 accuracy_score(y_test, y_pred)
```

```
[[62  6]
 [ 3 29]]
0.91
```

```
1 # Visualising the Training set results
2 from matplotlib.colors import ListedColormap
3 X_set, y_set = sc.inverse_transform(X_train), y_train
4 X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
5                      np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
6 plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()])).T).reshape(X1.shape),
7             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
8 plt.xlim(X1.min(), X1.max())
9 plt.ylim(X2.min(), X2.max())
10 for i, j in enumerate(np.unique(y_set)):
11     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
12 plt.title('Decision Tree Classification (Training set)')
13 plt.xlabel('Age')
14 plt.ylabel('Estimated Salary')
15 plt.legend()
16 plt.show()
```

```
<ipython-input-10-0bdb4b7ac97b>:11: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
```

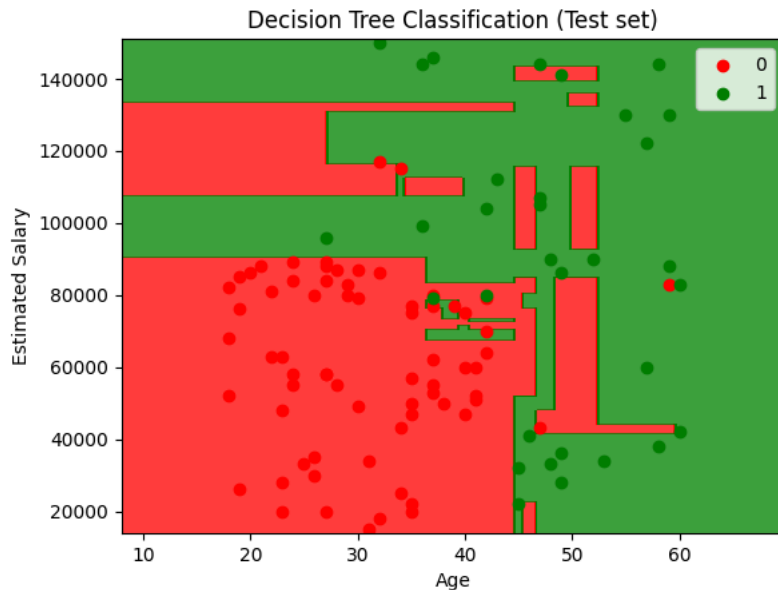


```

1 # Visualising the Test set results
2 from matplotlib.colors import ListedColormap
3 X_set, y_set = sc.inverse_transform(X_test), y_test
4 X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
5                       np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
6 plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()])).T)).reshape(X1.shape),
7             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
8 plt.xlim(X1.min(), X1.max())
9 plt.ylim(X2.min(), X2.max())
10 for i, j in enumerate(np.unique(y_set)):
11     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
12 plt.title('Decision Tree Classification (Test set)')
13 plt.xlabel('Age')
14 plt.ylabel('Estimated Salary')
15 plt.legend()
16 plt.show()

```

<ipython-input-9-0d0bde521908>:11: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)



* Random Forest Classifier *

```

1 # Importing the libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import pandas as pd

```

```

1 # Importing the dataset
2 dataset = pd.read_csv('Social_Network_Ads.csv')
3 X = dataset.iloc[:, :-1].values
4 y = dataset.iloc[:, -1].values

```

```

1 # Splitting the dataset into the Training set and Test set
2 from sklearn.model_selection import train_test_split
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
4 print(X_train)
5 print(y_train)
6 print(X_test)
7 print(y_test)

```

```

[[ 44  39000]
 [ 32 120000]
 [ 38  50000]
 [ 32 135000]
 [ 52  21000]
 [ 53 104000]
 [ 39  42000]
 [ 38  61000]
 [ 36  50000]
 [ 36  63000]

```

```
[ 35 25000]
[ 35 50000]
[ 42 73000]
[ 47 49000]
[ 59 29000]
[ 49 65000]
[ 45 131000]
[ 31 89000]
[ 46 82000]
[ 47 51000]
[ 26 15000]
[ 60 102000]
[ 38 112000]
[ 40 107000]
[ 42 53000]
[ 35 59000]
[ 48 41000]
[ 48 134000]
[ 38 113000]
[ 29 148000]
[ 26 15000]
[ 60 42000]
[ 24 19000]
[ 42 149000]
[ 46 96000]
[ 28 59000]
[ 39 96000]
[ 28 89000]
[ 41 72000]
[ 45 26000]
[ 33 69000]
[ 20 82000]
[ 31 74000]
[ 42 80000]
[ 35 72000]
[ 33 149000]
[ 40 71000]
[ 51 146000]
[ 46 79000]
[ 35 75000]
[ 38 51000]
[ 36 75000]
[ 37 78000]
[ 38 61000]
[ 60 108000]
[ 20 82000]
[ 57 74000]
[ 42 65000]
```

```
1 # Feature Scaling
2 from sklearn.preprocessing import StandardScaler
3 sc = StandardScaler()
4 X_train = sc.fit_transform(X_train)
5 X_test = sc.transform(X_test)
6 print(X_train)
7 print(X_test)
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
 [-0.21060859 -0.19087153]
 [-0.30964085 -1.29261101]
 [-0.30964085 -0.5677824 ]
 [ 0.38358493  0.09905991]
 [ 0.8787462  -0.59677555]
 [ 2.06713324 -1.17663843]
 [ 1.07681071 -0.13288524]
 [ 0.68068169  1.78066227]
 [-0.70576986  0.56295021]
 [ 0.77971394  0.35999821]
 [ 0.8787462  -0.53878926]
 [-1.20093113 -1.58254245]
 [ 2.1661655  0.93986109]
 [-0.01254409  1.22979253]
 [ 0.18552042  1.08482681]
 [ 0.38358493 -0.48080297]
 [-0.30964085 -0.30684411]
 [ 0.97777845 -0.8287207 ]]
```



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

```
1 # Training the Random Forest Classification model on the Training set
2 from sklearn.ensemble import RandomForestClassifier
3 classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
4 classifier.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=0)
```

```
1 # Predicting a new result
2 print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

```
1 # Predicting the Test set results
2 y_pred = classifier.predict(X_test)
3 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]
[1 1]
[0 0]
[1 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[1 0]
[1 0]
[0 0]
[1 1]
[0 0]
[0 0]
[1 1]
[0 0]
[0 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]
[1 1]
[0 0]
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[1 1]
[0 0]
[1 1]
[0 0]
[1 1]
[1 1]
[0 0]
[0 0]
[1 0]
[1 1]
[1 1]
[0 0]
[0 0]

```

```

1 # Making the Confusion Matrix
2 from sklearn.metrics import confusion_matrix, accuracy_score
3 cm = confusion_matrix(y_test, y_pred)
4 print(cm)
5 accuracy_score(y_test, y_pred)
6

```

```

[[63  5]
 [ 4 28]]
0.91

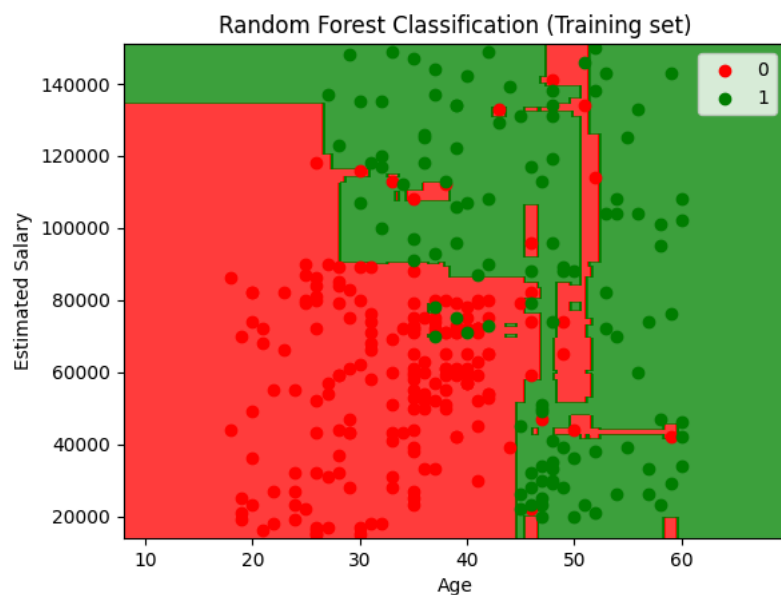
```

```

1 # Visualising the Training set results
2 from matplotlib.colors import ListedColormap
3 X_set, y_set = sc.inverse_transform(X_train), y_train
4 X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
5                      np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
6 plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()])).T)).reshape(X1.shape),
7             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
8 plt.xlim(X1.min(), X1.max())
9 plt.ylim(X2.min(), X2.max())
10 for i, j in enumerate(np.unique(y_set)):
11     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
12 plt.title('Random Forest Classification (Training set)')
13 plt.xlabel('Age')
14 plt.ylabel('Estimated Salary')
15 plt.legend()
16 plt.show()
17

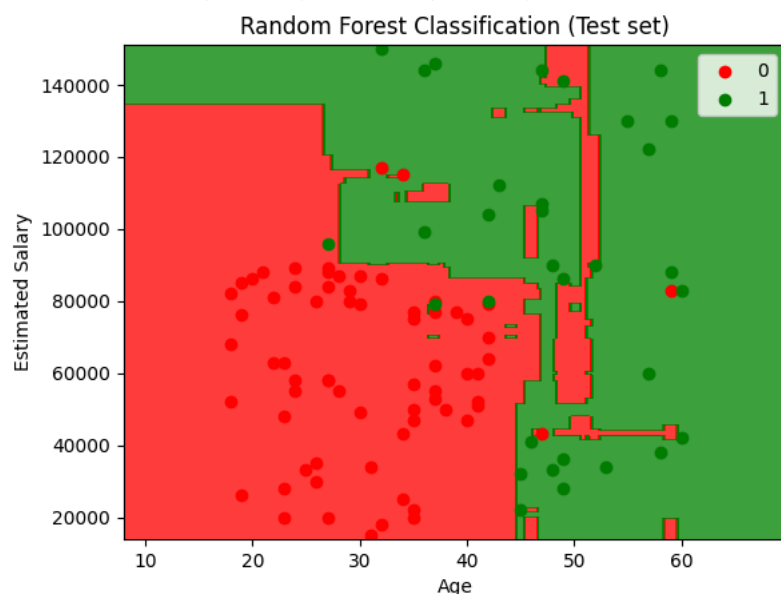
```

```
<ipython-input-19-d6b6ef8cb173>:11: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
```



```
1 # Visualising the Test set results
2 from matplotlib.colors import ListedColormap
3 X_set, y_set = sc.inverse_transform(X_test), y_test
4 X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
5                       np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
6 plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()])).T).reshape(X1.shape),
7             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
8 plt.xlim(X1.min(), X1.max())
9 plt.ylim(X2.min(), X2.max())
10 for i, j in enumerate(np.unique(y_set)):
11     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
12 plt.title('Random Forest Classification (Test set)')
13 plt.xlabel('Age')
14 plt.ylabel('Estimated Salary')
15 plt.legend()
16 plt.show()
```

```
<ipython-input-20-4f1e0d3254dd>:11: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
```



Clustering K-Means*

```
1 # Importing the libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import pandas as pd
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
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
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
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
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
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
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
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
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

1 dataset

	CustomerID	Genre	Age	Annual Income (k\$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

200 rows × 5 columns

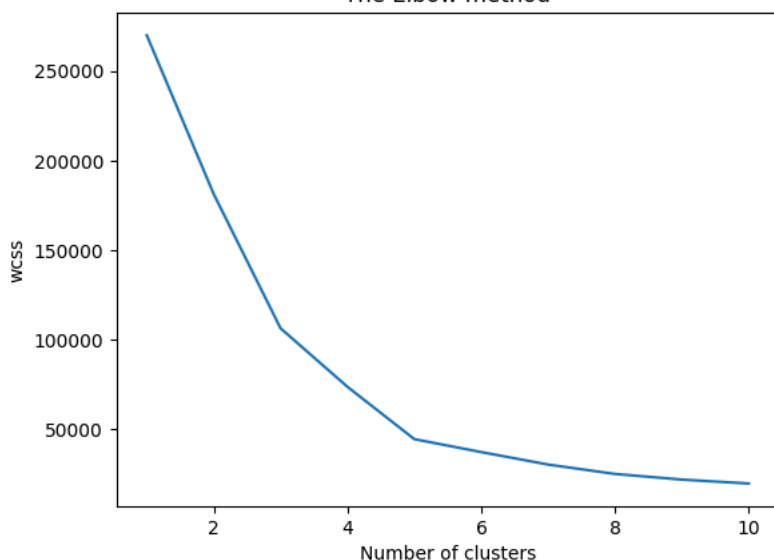
```
1 #Using Elbow method to find optimal number of cluster
2
3 from sklearn.cluster import KMeans
4 wcss=[]
5 for i in range(1,11):
6     kmeans=KMeans(n_clusters = i,init='k-means++',random_state=42)
7     kmeans.fit(X)
8     wcss.append(kmeans.inertia_)
9 plt.plot(range(1,11),wcss)
10 plt.title('The Elbow method')
11 plt.xlabel('Number of clusters')
12 plt.ylabel('wcss')
13 plt.show()
```

```

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
warnings.warn(

```

The Elbow method



```

1 #Training K-means model on dataset
2 kmeans=KMeans(n_clusters = 5,init='k-means++',random_state=42)
3 y_kmeans=kmeans.fit_predict(X)

```

```

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
warnings.warn(

```

```
1 print(y_kmeans)
```

```

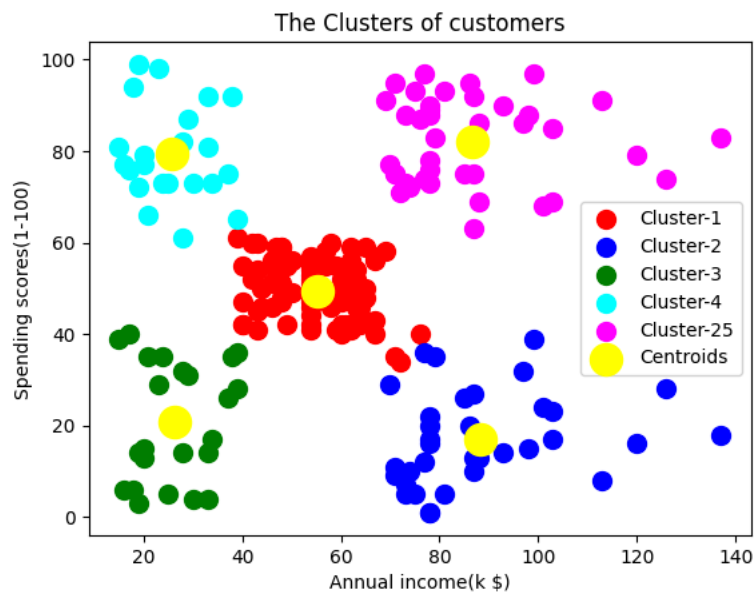
[2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2
 3 2 3 2 3 2 0 2 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 4 0 4 1 4 1 4 0 4 1 4 1 4 0 4 1 4 1 4 1 4 1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

```

```

1 #Visualize the clusters
2
3 plt.scatter(X[y_kmeans==0], X[y_kmeans==1],s=100,c='red',label='Cluster-1')
4 plt.scatter(X[y_kmeans==1,0], X[y_kmeans==1,1],s=100,c='blue',label='Cluster-2')
5 plt.scatter(X[y_kmeans==2,0], X[y_kmeans==2,1],s=100,c='green',label='Cluster-3')
6 plt.scatter(X[y_kmeans==3,0], X[y_kmeans==3,1],s=100,c='cyan',label='Cluster-4')
7 plt.scatter(X[y_kmeans==4,0], X[y_kmeans==4,1],s=100,c='magenta',label='Cluster-25')
8 plt.scatter(kmeans.cluster_centers[:,0],kmeans.cluster_centers[:,1],s=300,c='yellow',label='Centroids')
9 plt.title('The Clusters of customers')
10 plt.xlabel('Annual income(k $)')
11 plt.ylabel('Spending scores(1-100)')
12 plt.legend()
13 plt.show()

```



Hierarchical Clustering

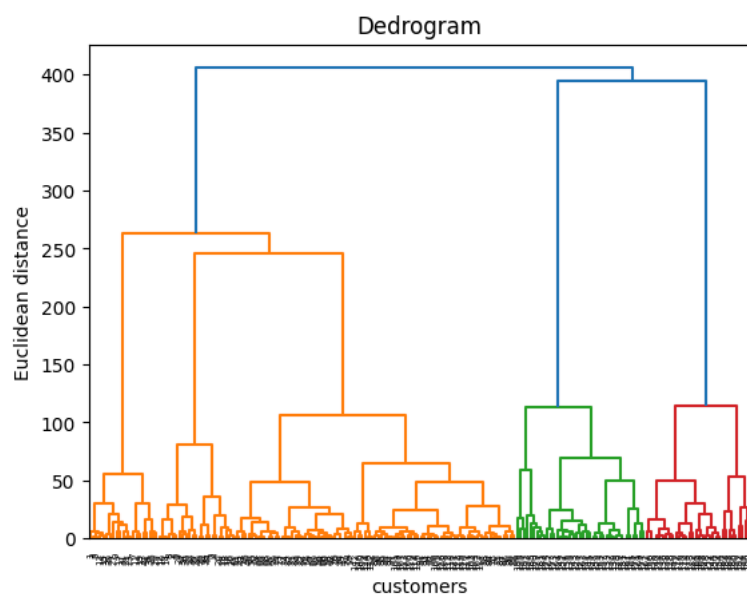
```

1 # Importing the libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import pandas as pd

1 # Importing the dataset
2 dataset = pd.read_csv('Mall_Customers.csv')
3 X = dataset.iloc[:, [3, 4]].values
4

1 #Using the dendrogram to find optimal number of cluster
2 import scipy.cluster.hierarchy as sch
3 dendrogram= sch.dendrogram(sch.linkage(X,method='ward'))
4 plt.title('Dedrogram')
5 plt.xlabel('customers')
6 plt.ylabel('Euclidean distance')
7 plt.show()

```



```

1 #Training the Hirerarchical Clustering model on the dataset
2 from sklearn.cluster import AgglomerativeClustering
3 hc=AgglomerativeClustering(n_clusters=5, affinity='euclidean',linkage='ward')
4 y_hc=hc.fit_predict(X)

```

```

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_agglomerative.py:983: FutureWarning: Attribute `affinity` was deprecated in ver
warnings.warn(

```

```
1 print(y_hc)
```

```

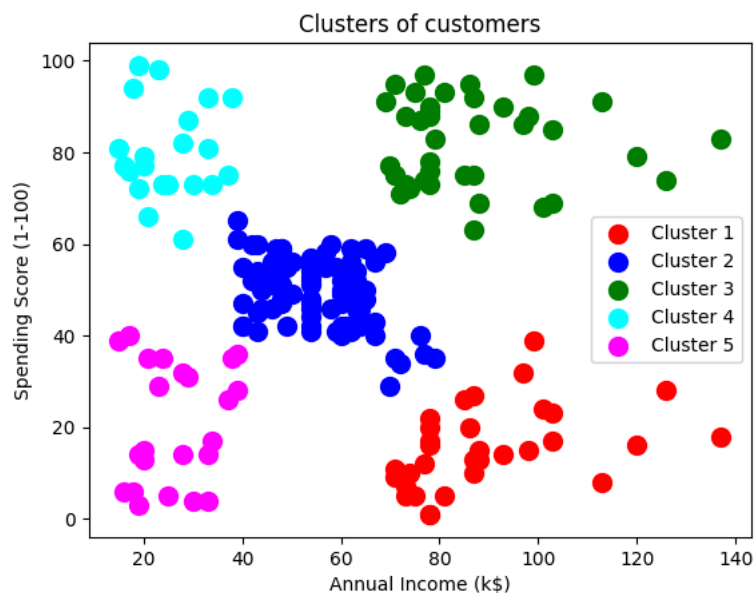
[4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4
 3 4 3 4 3 4 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 0 2 0 2 1 2 0 2 0 2 0 2 1 2 0 2 1 2
 0 2 0 2 0 2 0 2 0 2 0 2 1 2 0 2 0 2 0 2 0 2 0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

```

```

1 # Visualising the clusters
2 plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
3 plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
4 plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
5 plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
6 plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
7 plt.title('Clusters of customers')
8 plt.xlabel('Annual Income (k$)')
9 plt.ylabel('Spending Score (1-100)')
10 plt.legend()
11 plt.show()

```



Double-click (or enter) to edit

* Association Rule Leraning [People who bought also bought]*

```
1 pip install apyori
```

```

Collecting apyori
  Downloading apyori-1.1.2.tar.gz (8.6 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: apyori
  Building wheel for apyori (setup.py) ... done
  Created wheel for apyori: filename=apyori-1.1.2-py3-none-any.whl size=5955 sha256=9a9748efbc8b90c1ab24c2600d604558cf48e5b90bffb246925e
  Stored in directory: /root/.cache/pip/wheels/c4/1a/79/20f55c470a50bb3702a8cb7c94d8ada15573538c7f4baebe2d
Successfully built apyori
Installing collected packages: apyori
Successfully installed apyori-1.1.2

```

```
1 # Importing the libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import pandas as pd

1 # Data Preprocessing
2 dataset = pd.read_csv('Market_Basket_Optimisation.csv', header = None)
3 transactions = []
4 for i in range(0, 7501):
5     transactions.append([str(dataset.values[i,j]) for j in range(0, 20)])

1 # Training the Apriori model on the dataset
2 from apyori import apriori
3 rules = apriori(transactions = transactions, min_support = 0.003, min_confidence = 0.2, min_lift = 3, min_length = 2, max_length = 2)

1 results = list(rules)
2 results

[RelationRecord(items=frozenset({'light cream', 'chicken'}), support=0.004532728969470737, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'light cream'}), items_add=frozenset({'chicken'}), confidence=0.29059829059829057,
lift=4.84395061728395)]),
RelationRecord(items=frozenset({'mushroom cream sauce', 'escalope'}), support=0.005732568990801226, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'mushroom cream sauce'}), items_add=frozenset({'escalope'}), confidence=0.3006993006993007,
lift=3.790832696715049)]),
RelationRecord(items=frozenset({'pasta', 'escalope'}), support=0.005865884548726837, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'pasta'}), items_add=frozenset({'escalope'}), confidence=0.3728813559322034,
lift=4.700811850163794)]),
RelationRecord(items=frozenset({'fromage blanc', 'honey'}), support=0.003332888948140248, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'fromage blanc'}), items_add=frozenset({'honey'}), confidence=0.2450980392156863,
lift=5.164270764485569)]),
RelationRecord(items=frozenset({'ground beef', 'herb & pepper'}), support=0.015997866951073192, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'herb & pepper'}), items_add=frozenset({'ground beef'}), confidence=0.3234501347708895,
lift=3.2919938411349285)]),
RelationRecord(items=frozenset({'ground beef', 'tomato sauce'}), support=0.005332622317024397, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'tomato sauce'}), items_add=frozenset({'ground beef'}), confidence=0.3773584905660377,
lift=3.840659481324083)]),
RelationRecord(items=frozenset({'olive oil', 'light cream'}), support=0.003199573390214638, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'light cream'}), items_add=frozenset({'olive oil'}), confidence=0.20512820512820515,
lift=3.1147098515519573)]),
RelationRecord(items=frozenset({'olive oil', 'whole wheat pasta'}), support=0.007998933475536596, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'whole wheat pasta'}), items_add=frozenset({'olive oil'}), confidence=0.2714932126696833,
lift=4.122410097642296)]),
RelationRecord(items=frozenset({'shrimp', 'pasta'}), support=0.005065991201173177, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'pasta'}), items_add=frozenset({'shrimp'}), confidence=0.3220338983050847,
lift=4.506672147735896)])]

1 ## Putting the results well organised into a Pandas DataFrame
2 def inspect(results):
3     lhs          = [tuple(result[2][0][0])[0] for result in results]
4     rhs          = [tuple(result[2][0][1])[0] for result in results]
5     supports     = [result[1] for result in results]
6     confidences  = [result[2][0][2] for result in results]
7     lifts       = [result[2][0][3] for result in results]
8     return list(zip(lhs, rhs, supports, confidences, lifts))
9 resultsinDataFrame = pd.DataFrame(inspect(results), columns = ['Left Hand Side', 'Right Hand Side', 'Support', 'Confidence', 'Lift'])

1 ## Displaying the results non sorted
2 resultsinDataFrame
```

	Left Hand Side	Right Hand Side	Support	Confidence	Lift
0	light cream	chicken	0.004533	0.290598	4.843951
1	mushroom cream sauce	escalope	0.005733	0.300699	3.790833
2	pasta	escalope	0.005866	0.372881	4.700812
3	fromage blanc	honey	0.003333	0.245098	5.164271
4	herb & pepper	ground beef	0.015998	0.323450	3.291994
5	tomato sauce	ground beef	0.005333	0.377358	3.840659
6	light cream	olive oil	0.003200	0.205128	3.114710
7	whole wheat pasta	olive oil	0.007999	0.271493	4.122410
8	pasta	shrimp	0.005066	0.322034	4.506672


```
1 ## Displaying the results sorted by descending lifts
2 resultsinDataFrame.nlargest(n = 10, columns = 'Lift')
```

	Left Hand Side	Right Hand Side	Support	Confidence	Lift
3	fromage blanc	honey	0.003333	0.245098	5.164271
0	light cream	chicken	0.004533	0.290598	4.843951
2	pasta	escalope	0.005866	0.372881	4.700812
8	pasta	shrimp	0.005066	0.322034	4.506672
7	whole wheat pasta	olive oil	0.007999	0.271493	4.122410
5	tomato sauce	ground beef	0.005333	0.377358	3.840659
1	mushroom cream sauce	escalope	0.005733	0.300699	3.790833
4	herb & pepper	ground beef	0.015998	0.323450	3.291994
6	light cream	olive oil	0.003200	0.205128	3.114710

***Associate Rule Learning-- ECALT Tution ***

```
1 # Importing the libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import pandas as pd

1 # Data Preprocessing
2 dataset = pd.read_csv('Market_Basket_Optimisation.csv', header = None)
3 transactions = []
4 for i in range(0, 7501):
5     transactions.append([str(dataset.values[i,j]) for j in range(0, 20)])
```

```
1 pip install apyori

Collecting apyori
  Downloading apyori-1.1.2.tar.gz (8.6 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: apyori
  Building wheel for apyori (setup.py) ... done
  Created wheel for apyori: filename=apyori-1.1.2-py3-none-any.whl size=5955 sha256=6eb354319773b8de3c58e3feb3b2e0171cbc2055ee44f4d7cb62
  Stored in directory: /root/.cache/pip/wheels/c4/1a/79/20f55c470a50bb3702a8cb7c94d8ada15573538c7f4baebe2d
Successfully built apyori
Installing collected packages: apyori
Successfully installed apyori-1.1.2
```

```
1 # Training the Eclat model on the dataset
2 from apyori import apriori
3 rules = apriori(transactions = transactions, min_support = 0.003, min_confidence = 0.2, min_lift = 3, min_length = 2, max_length = 2)
```

```
1 # Visualising the results
2
3 ## Displaying the first results coming directly from the output of the apriori function
4 results = list(rules)
5 results
```

```
[RelationRecord(items=frozenset({'chicken', 'light cream'}), support=0.004532728969470737, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'light cream'}), items_add=frozenset({'chicken'}), confidence=0.29059829059829057,
lift=4.84395061728395)]),
RelationRecord(items=frozenset({'escalope', 'mushroom cream sauce'}), support=0.005732568990801226, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'mushroom cream sauce'}), items_add=frozenset({'escalope'}), confidence=0.3006993006993007,
lift=3.790832696715049)]),
RelationRecord(items=frozenset({'escalope', 'pasta'}), support=0.005865884548726837, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'pasta'}), items_add=frozenset({'escalope'}), confidence=0.3728813559322034,
lift=4.700811850163794)]),
RelationRecord(items=frozenset({'honey', 'fromage blanc'}), support=0.003332888948140248, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'fromage blanc'}), items_add=frozenset({'honey'}), confidence=0.2450980392156863,
lift=5.164270764485569)]),
RelationRecord(items=frozenset({'herb & pepper', 'ground beef'}), support=0.015997866951073192, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'herb & pepper'}), items_add=frozenset({'ground beef'}), confidence=0.3234501347708895,
lift=3.2919938411349285)]),
RelationRecord(items=frozenset({'tomato sauce', 'ground beef'}), support=0.005332622317024397, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'tomato sauce'}), items_add=frozenset({'ground beef'}), confidence=0.3773584905660377,
```

```
lift=3.840659481324083])),
  RelationRecord(items=frozenset({'olive oil', 'light cream'}), support=0.003199573390214638, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'light cream'}), items_add=frozenset({'olive oil'}), confidence=0.20512820512820515,
lift=3.1147098515519573])),
  RelationRecord(items=frozenset({'whole wheat pasta', 'olive oil'}), support=0.007998933475536596, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'whole wheat pasta'}), items_add=frozenset({'olive oil'}), confidence=0.2714932126696833,
lift=4.122410097642296])),
  RelationRecord(items=frozenset({'shrimp', 'pasta'}), support=0.005065991201173177, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'pasta'}), items_add=frozenset({'shrimp'}), confidence=0.3220338983050847,
lift=4.506672147735896)])]

1 ## Putting the results well organised into a Pandas DataFrame
2 def inspect(results):
3     lhs         = [tuple(result[2][0][0])[0] for result in results]
4     rhs         = [tuple(result[2][0][1])[0] for result in results]
5     supports    = [result[1] for result in results]
6     return list(zip(lhs, rhs, supports))
7 resultsinDataFrame = pd.DataFrame(inspect(results), columns = ['Product 1', 'Product 2', 'Support'])

1 ## Displaying the results sorted by descending supports
2 resultsinDataFrame.nlargest(n = 10, columns = 'Support')
```

	Product 1	Product 2	Support
4	herb & pepper	ground beef	0.015998
7	whole wheat pasta	olive oil	0.007999
2	pasta	escalope	0.005866
1	mushroom cream sauce	escalope	0.005733
5	tomato sauce	ground beef	0.005333
8	pasta	shrimp	0.005066
0	light cream	chicken	0.004533
3	fromage blanc	honey	0.003333
6	light cream	olive oil	0.003200

Upper Confidence Bound (UCB)

```
1 # Importing the libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import pandas as pd

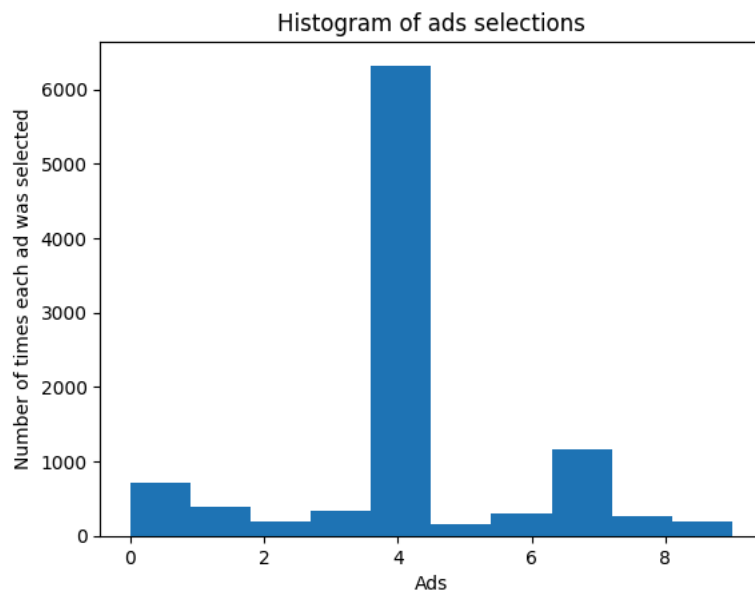
1 # Importing the dataset
2 dataset = pd.read_csv('Ads_CTR_Optimisation.csv')
```

```

1 # Implementing UCB
2 import math
3 N = 10000
4 d = 10
5 ads_selected = []
6 numbers_of_selections = [0] * d
7 sums_of_rewards = [0] * d
8 total_reward = 0
9 for n in range(0, N):
10     ad = 0
11     max_upper_bound = 0
12     for i in range(0, d):
13         if (numbers_of_selections[i] > 0):
14             average_reward = sums_of_rewards[i] / numbers_of_selections[i]
15             delta_i = math.sqrt(3/2 * math.log(n + 1) / numbers_of_selections[i])
16             upper_bound = average_reward + delta_i
17         else:
18             upper_bound = 1e400
19         if upper_bound > max_upper_bound:
20             max_upper_bound = upper_bound
21     ad = i
22     ads_selected.append(ad)
23     numbers_of_selections[ad] = numbers_of_selections[ad] + 1
24     reward = dataset.values[n, ad]
25     sums_of_rewards[ad] = sums_of_rewards[ad] + reward
26     total_reward = total_reward + reward

1 # Visualising the results
2 plt.hist(ads_selected)
3 plt.title('Histogram of ads selections')
4 plt.xlabel('Ads')
5 plt.ylabel('Number of times each ad was selected')
6 plt.show()

```



NLP Natural language processing

```

1 #Bag of word
2 # Importing the libraries
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import pandas as pd

1 #Importing the dataset : note quoting 3 is used to remove "" from comment
2 dataset = pd.read_csv('Restaurant_Reviews.tsv', delimiter = '\t', quoting=3)

```

```

1 #Cleaning the text
2 import re
3 import nltk
4 nltk.download('stopwords')
5 from nltk.corpus import stopwords
6 from nltk.stem.porter import PorterStemmer
7 corpus = []
8 for i in range(0, 1000):
9     review = re.sub('[^a-zA-Z]', ' ', dataset['Review'][i])
10    review = review.lower()
11    review = review.split()
12    ps = PorterStemmer()
13    all_stopwords = stopwords.words('english')
14    all_stopwords.remove('not')
15    review = [ps.stem(word) for word in review if not word in set(all_stopwords)]
16    review = ' '.join(review)
17    corpus.append(review)
18 print(corpus)

```

[nltk_data] Downloading package stopwords to /root/nltk_data...

[nltk_data] Unzipping corpora/stopwords.zip.

['wow love place', 'crust not good', 'not tasti textur nasti', 'stop late may bank holiday rick steve recommend love', 'select menu grea

```

1 #Creating a Bag words
2 from sklearn.feature_extraction.text import CountVectorizer
3 cv = CountVectorizer(max_features = 1500)
4 X = cv.fit_transform(corpus).toarray()
5 y = dataset.iloc[:, -1].values

```

```
1 len(X[0])
```

1500

```

1 # Splitting the dataset into the Training set and Test set
2 from sklearn.model_selection import train_test_split
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)

```

```

1 # Training the Naive Bayes model on the Training set
2 from sklearn.naive_bayes import GaussianNB
3 classifier = GaussianNB()
4 classifier.fit(X_train, y_train)

```

▼ GaussianNB

GaussianNB()

```

1 # Predicting the Test set results
2 y_pred = classifier.predict(X_test)
3 print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))

```

```

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

```

```

[0 1]
[1 1]
[1 1]
[1 1]
[0 0]
[1 1]
[1 1]
[1 1]
[1 1]
[1 1]
[0 0]
[1 0]
[0 0]
[1 0]
[1 1]
[1 1]
[1 1]
[1 0]
[1 1]
[0 0]
[0 0]
[0 0]
[0 0]
[1 0]
[1 0]
[0 0]
[0 0]
[1 1]
[1 1]
[1 1]
[1 1]
[1 0]
[0 0]
[1 1]
-- --

```

```

1 # Making the Confusion Matrix
2 from sklearn.metrics import confusion_matrix, accuracy_score
3 cm = confusion_matrix(y_test, y_pred)
4 print(cm)
5 accuracy_score(y_test, y_pred)

```

```

[[55 42]
 [12 91]]
0.73

```

Predicting if a single review is positive or negative

```

1 new_review = 'I love this restaurant so much'
2 new_review = re.sub('[^a-zA-Z]', ' ', new_review)
3 new_review = new_review.lower()
4 new_review = new_review.split()
5 ps = PorterStemmer()
6 all_stopwords = stopwords.words('english')
7 all_stopwords.remove('not')
8 new_review = [ps.stem(word) for word in new_review if not word in set(all_stopwords)]
9 new_review = ' '.join(new_review)
10 new_corpus = [new_review]
11 new_X_test = cv.transform(new_corpus).toarray()
12 new_y_pred = classifier.predict(new_X_test)
13 print(new_y_pred)

```

```
[1]
```

```

1 new_review = 'I hate this restaurant so much'
2 new_review = re.sub('[^a-zA-Z]', ' ', new_review)
3 new_review = new_review.lower()
4 new_review = new_review.split()
5 ps = PorterStemmer()
6 all_stopwords = stopwords.words('english')
7 all_stopwords.remove('not')
8 new_review = [ps.stem(word) for word in new_review if not word in set(all_stopwords)]
9 new_review = ' '.join(new_review)
10 new_corpus = [new_review]
11 new_X_test = cv.transform(new_corpus).toarray()
12 new_y_pred = classifier.predict(new_X_test)
13 print(new_y_pred)

```

```
[0]
```

Importing the Libraries

```

1 # Importing the libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import pandas as pd

1 #Importing dataset - quoting is used to remove " from the text
2 dataset=pd.read_csv('/content/Restaurant_Reviews.tsv',sep='\t', quoting=3)

```

```
1 dataset.head()
```

	Review	Liked
0	Wow... Loved this place.	1
1	Crust is not good.	0
2	Not tasty and the texture was just nasty.	0
3	Stopped by during the late May bank holiday of...	1
4	The selection on the menu was great and so wer...	1

```

1 # Cleaning the texts
2 import re
3 import nltk
4 nltk.download('stopwords')
5 from nltk.corpus import stopwords
6 from nltk.stem.porter import PorterStemmer
7 corpus = []
8 for i in range(0, 1000):
9     review = re.sub('[^a-zA-Z]', ' ', dataset['Review'][i])
10    review = review.lower()
11    review = review.split()
12    ps = PorterStemmer()
13    all_stopwords = stopwords.words('english')
14    all_stopwords.remove('not')
15    review = [ps.stem(word) for word in review if not word in set(all_stopwords)]
16    review = ' '.join(review)
17    corpus.append(review)
18 print(corpus)

```

```

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
['wow love place', 'crust not good', 'not tasti textur nasti', 'stop late may bank holiday rick steve recommend love', 'select menu grea

```

```
1 print(corpus)
```

```

['wow love place', 'crust not good', 'not tasti textur nasti', 'stop late may bank holiday rick steve recommend love', 'select menu grea

```

```

1 # Creating the Bag of Words model
2 from sklearn.feature_extraction.text import CountVectorizer
3 cv = CountVectorizer(max_features = 1500)
4 X = cv.fit_transform(corpus).toarray()
5 y = dataset.iloc[:, -1].values

```

```
1 len(X[0])
```

```
1500
```

```

1 # Splitting the dataset into the Training set and Test set
2 from sklearn.model_selection import train_test_split
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)

```

```

1 # Training the Naive Bayes model on the Training set
2 from sklearn.naive_bayes import GaussianNB
3 classifier = GaussianNB()
4 classifier.fit(X_train, y_train)

```

▼ GaussianNB
GaussianNB()

```
1 # Predicting the Test set results
2 y_pred = classifier.predict(X_test)
3 print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

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