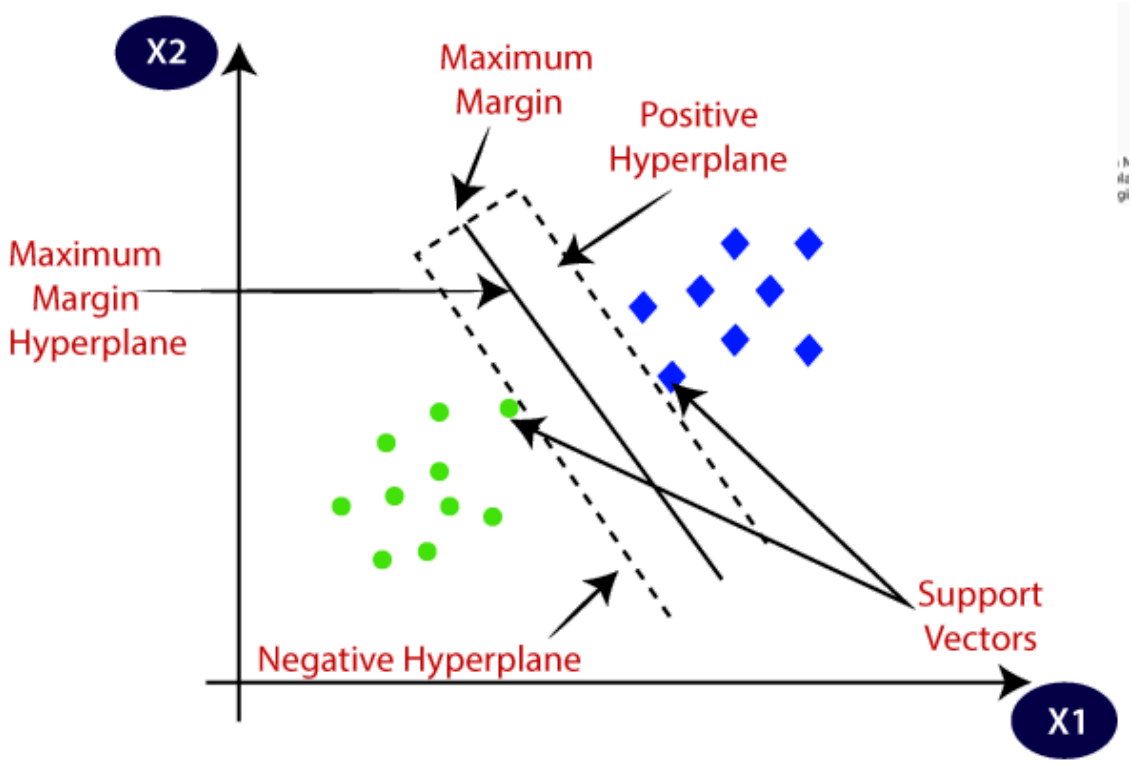
SVM (Support Vector Machine)

SVM works by mapping data to a high-dimensional feature space so that data points can be categorized, even when the data are not otherwise linearly separable. A separator between the categories is found, then the data is transformed in such a way that the separator could be drawn as a hyperplane. Following this, characteristics of new data can be used to predict the group to which a new record should belong.

Goal of SVM

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane.

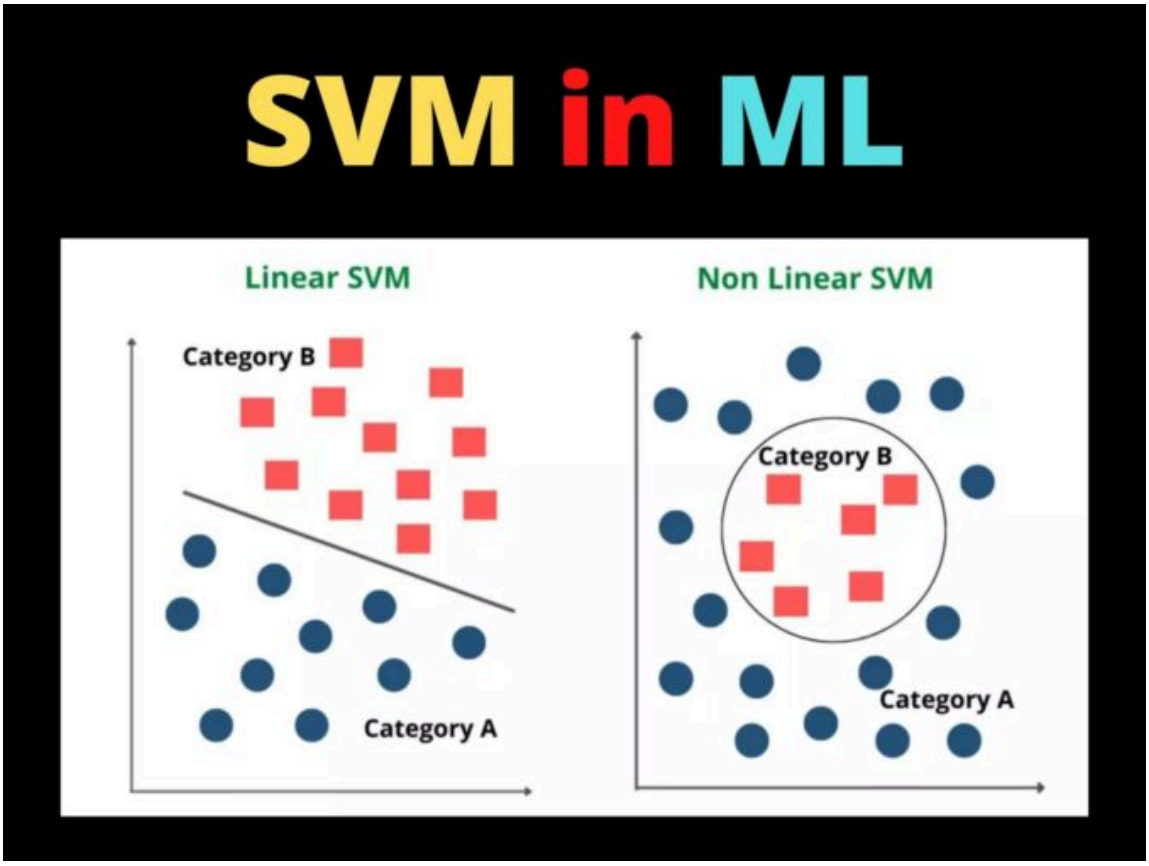


Types of SVM

**SVM have two types: **

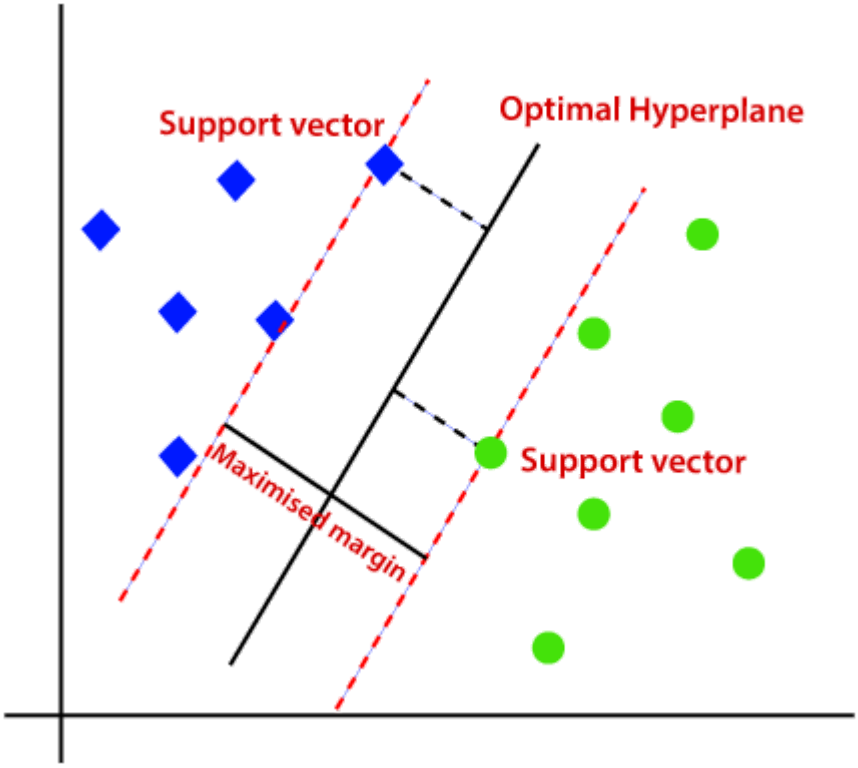
1. Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

2. Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.



Optimal Hyperplane

Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. The SVM algorithm helps to find the best line or decision boundary. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as margin. And the goal of SVM is to maximize this margin. The hyperplane with maximum margin is called the optimal hyperplane



Data Preprocessing

In [10]:
```
# Importing Necessary Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [2]:
```
# Load the Universal Bank Data
df = pd.read_csv('UniversalBank.csv')
df.head()
```

Out[2]:

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Sec Ac
0	1	25	1	49	91107	4	1.6	1	0	0	
1	2	45	19	34	90089	3	1.5	1	0	0	
2	3	39	15	11	94720	1	1.0	1	0	0	
3	4	35	9	100	94112	1	2.7	2	0	0	
4	5	35	8	45	91330	4	1.0	2	0	0	

In [3]:
```
# Checking for null values
df.isnull().sum()
```

Out[3]:

	0
ID	0
Age	0
Experience	0
Income	0
ZIP Code	0
Family	0
CCAvg	0
Education	0
Mortgage	0
Personal Loan	0
Securities Account	0
CD Account	0
Online	0
CreditCard	0

dtype: int64

In [4]:
```
# Dropping ID and ZIP Code columns from the dataset
df1 = df.drop(["ID","ZIP Code"], axis = 1)
df1.head()
```

Out[4]:

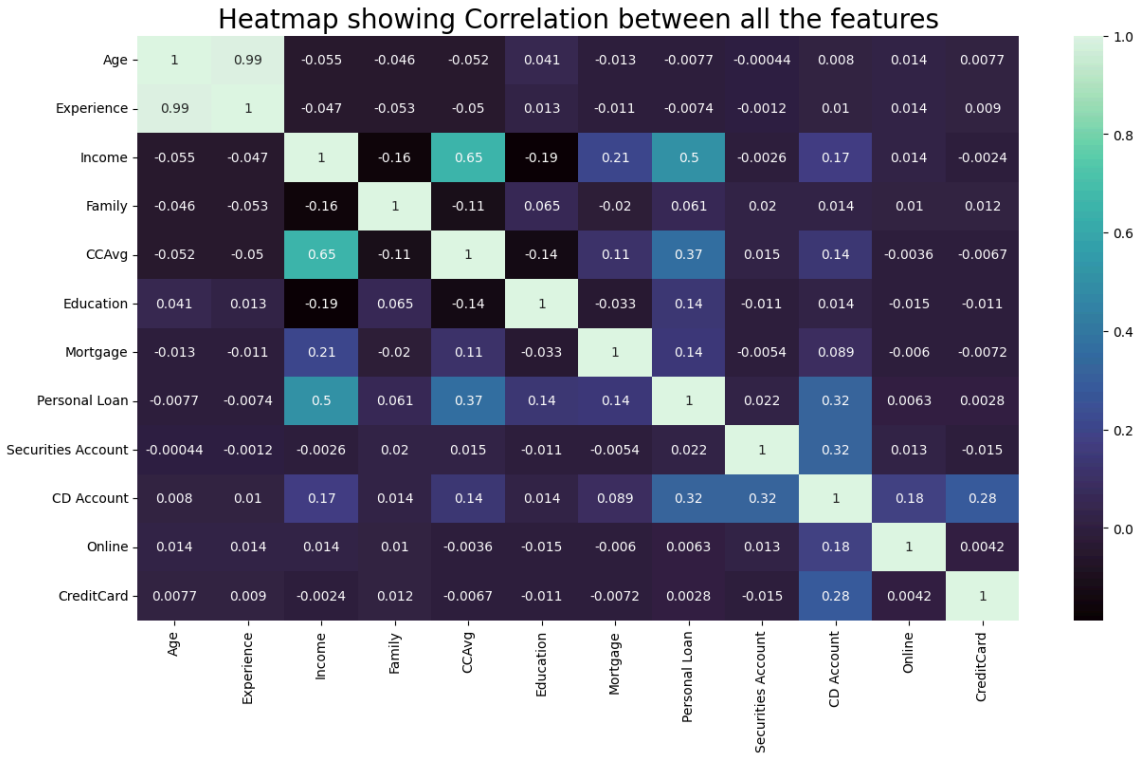
	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	Acc
0	25	1	49	4	1.6	1	0	0	1	
1	45	19	34	3	1.5	1	0	0	1	
2	39	15	11	1	1.0	1	0	0	0	
3	35	9	100	1	2.7	2	0	0	0	
4	35	8	45	4	1.0	2	0	0	0	

Heatmap

**Plotting the Heatmap to see the correlations between features. **

In [5]:
```
plt.figure(figsize=(15,8))
plt.title("Heatmap showing Correlation between all the features",
fontsize=20)
sns.heatmap(df1.corr(),annot = True, cmap='mako')
```

Out[5]:
```
<Axes: title={'center': 'Heatmap showing Correlation between all the featu
res'}>
```



Separating the class 0 and class 1 CreditCard data

In [6]:
```
zero_class = df1[df1.CreditCard==0]
zero_class.shape
```

Out[6]:
```
(3530, 12)
```

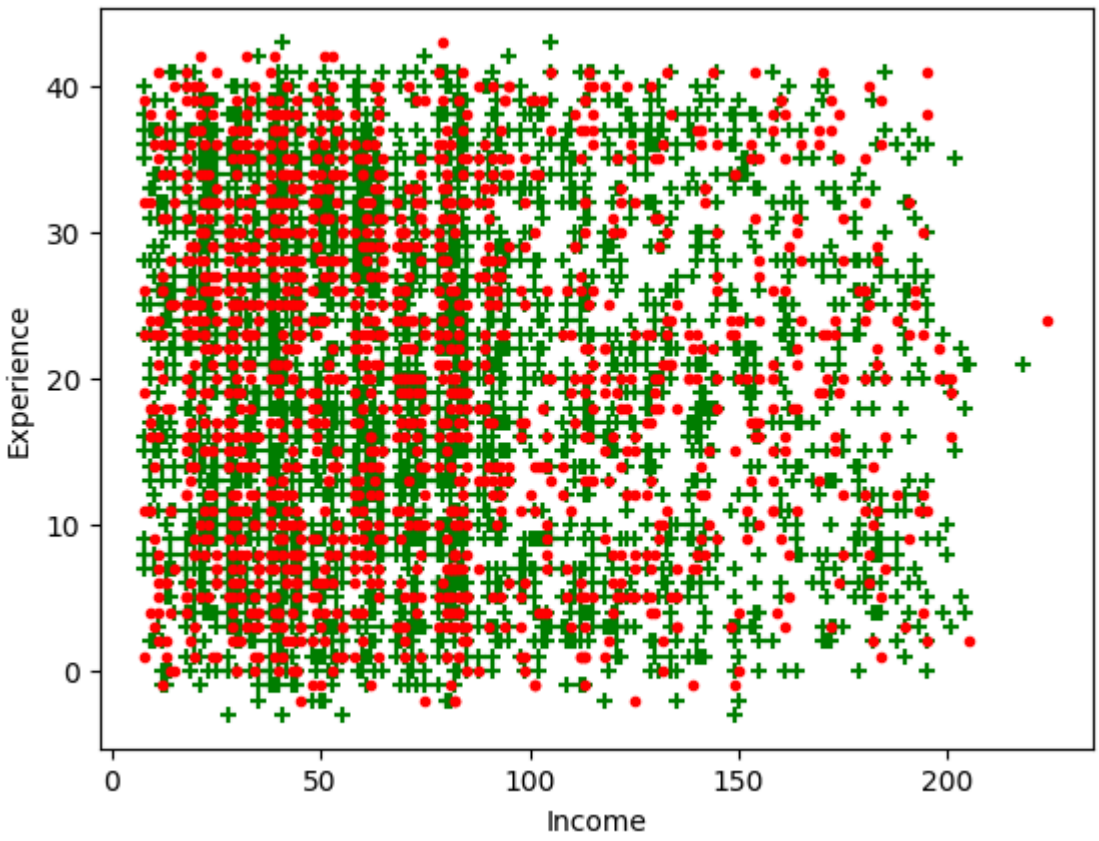
In [7]:
```
one_class = df1[df1.CreditCard==1]
one_class.shape
```

Out[7]:
```
(1470, 12)
```

Scatter Plot Here I have plotted the scatter plot using two features at a time "Income" and "Experience" & "CCAvg" and "Family" to visualize the distribution of our data. But, for the implimentation purpous we will take all the features.

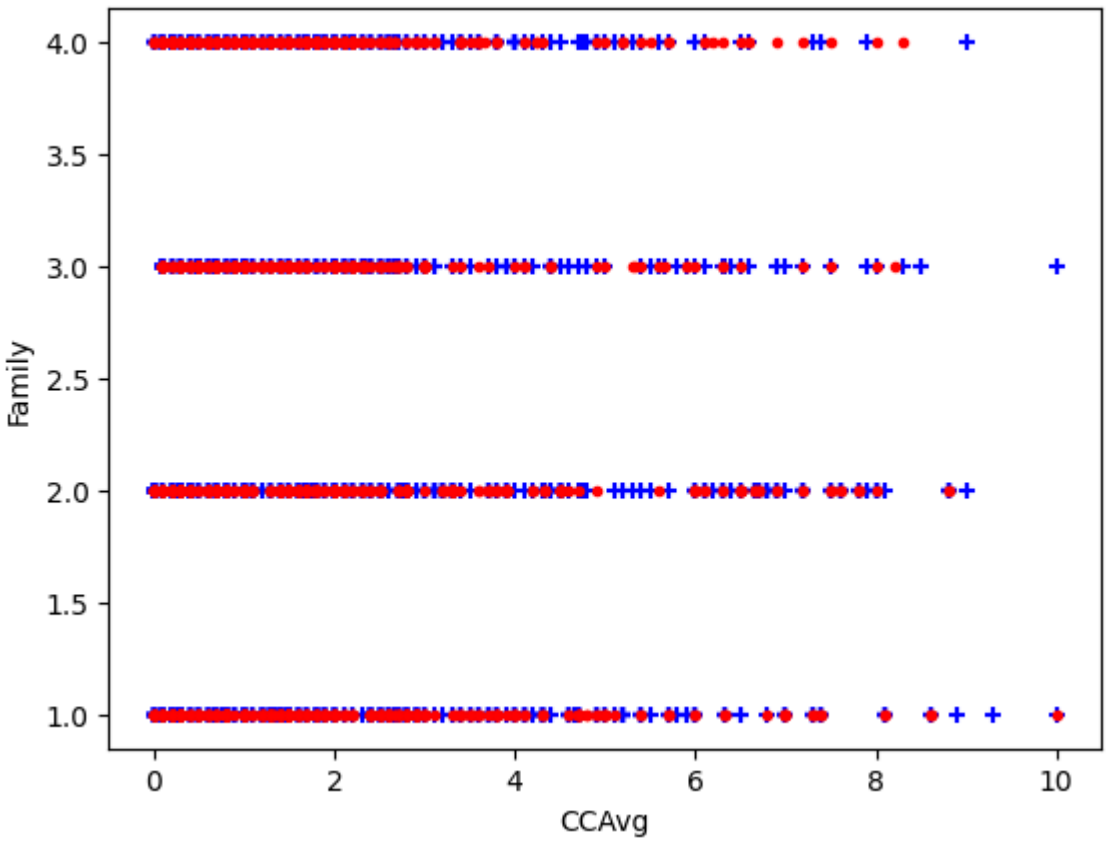
In [8]:
```
# Income vs Experience scatter plot
plt.xlabel('Income')
plt.ylabel('Experience')
plt.scatter(zero_class['Income'],zero_class['Experience'], color =
'green', marker='+')
plt.scatter(one_class['Income'], one_class['Experience'], color = 'red',
marker='.')
```

Out[8]: <matplotlib.collections.PathCollection at 0x7c9ff73c5cf0>



In [9]:
```
# CCAvg vs Family scatter plot
plt.xlabel('CCAvg')
plt.ylabel('Family')
plt.scatter(zero_class['CCAvg'],zero_class['Family'], color = 'blue',
marker='+')
plt.scatter(one_class['CCAvg'], one_class['Family'], color = 'red',
marker='.')
```

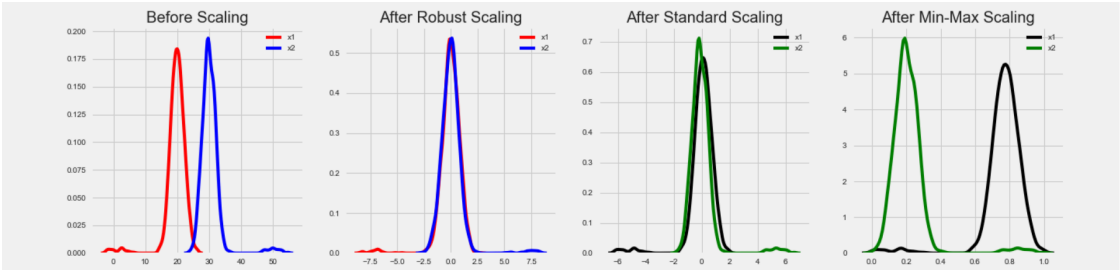
Out[9]: <matplotlib.collections.PathCollection at 0x7c9ff7442860>



Scaling the data¶

Here we can see our data is not in the same range. So, We need to scale our data in same range for that we will use Standard Scaler technique.

StandardScaler follows Standard Normal Distribution (SND). Therefore, it makes mean = 0 and scales the data to unit variance.



In [11]:
```
# Scaling the data using Standard Scaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled =
scaler.fit(df1.drop('CreditCard',axis=1)).transform(df1.drop('CreditCard'
,axis=1))
df_scaled = pd.DataFrame(scaled, columns=df1.columns[:-1])
df_scaled.head()
```

Out[11]:

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Personal Loan	Se
0	-1.774417	-1.666078	-0.538229	1.397414	-0.193371	-1.049078	-0.555524	-0.325875	2
1	-0.029524	-0.096330	-0.864109	0.525991	-0.250595	-1.049078	-0.555524	-0.325875	2
2	-0.552992	-0.445163	-1.363793	-1.216855	-0.536720	-1.049078	-0.555524	-0.325875	-0
3	-0.901970	-0.968413	0.569765	-1.216855	0.436103	0.141703	-0.555524	-0.325875	-0
4	-0.901970	-1.055621	-0.625130	1.397414	-0.536720	0.141703	-0.555524	-0.325875	-0

In [12]:
```
# Splitting the columns in to dependent variable (x) and independent
variable (y).
x = df_scaled
y = df1['CreditCard']
```

Implimentation of SVM¶

Now we will implement the SVM algorithm using Python. Here we will use the Universal Bank Dataset to understand the Support Vector Machine Algorithm.

In [13]:
```
# Split data in to train and test
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=0)
```

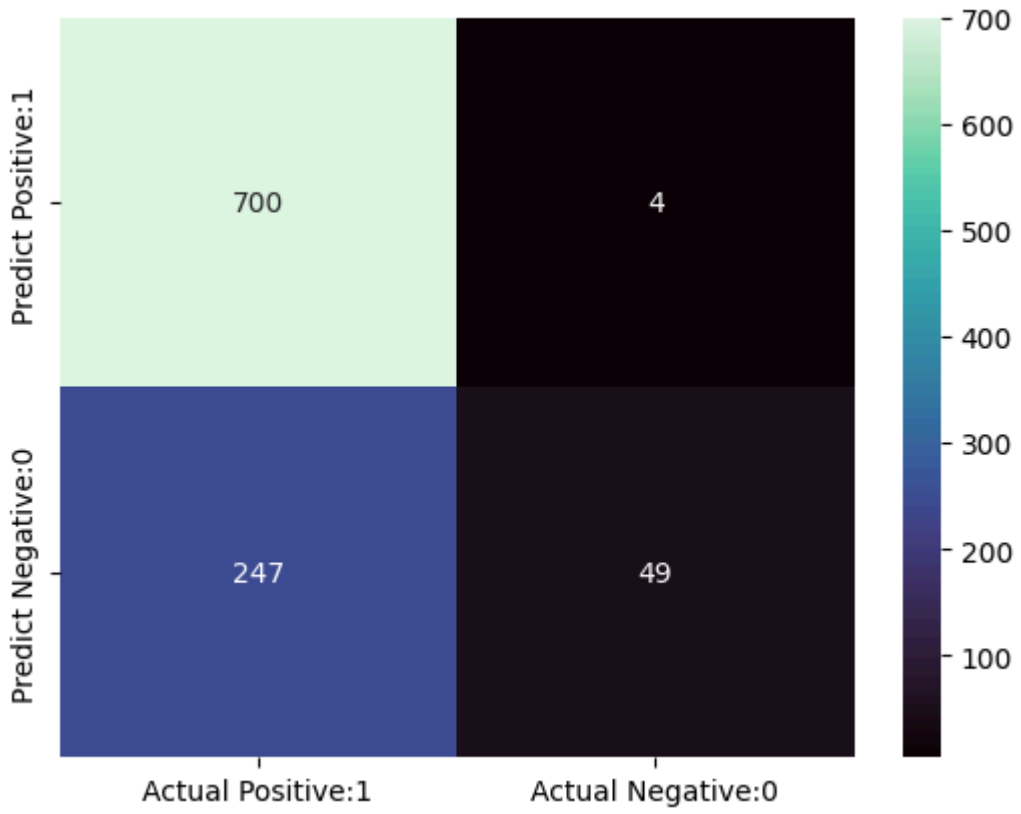
In [14]:
```
# Apply SVM Model
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
svc=SVC()
svc.fit(x_train, y_train)
y_pred=svc.predict(x_test)
print('Model accuracy : {0:0.3f}'. format(accuracy_score(y_test,
y_pred)))
```

```
Model accuracy : 0.749
```

#**A confusion matrix** is a performance evaluation tool in machine learning, representing the accuracy of a classification model. It displays the number of true positives, true negatives, false positives, and false negatives.

In [15]:
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual
Negative:0'],
                                  index=['Predict Positive:1', 'Predict
Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='mako')
```

Out[15]: <Axes: >



In [16]:
```
# Classification Report
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

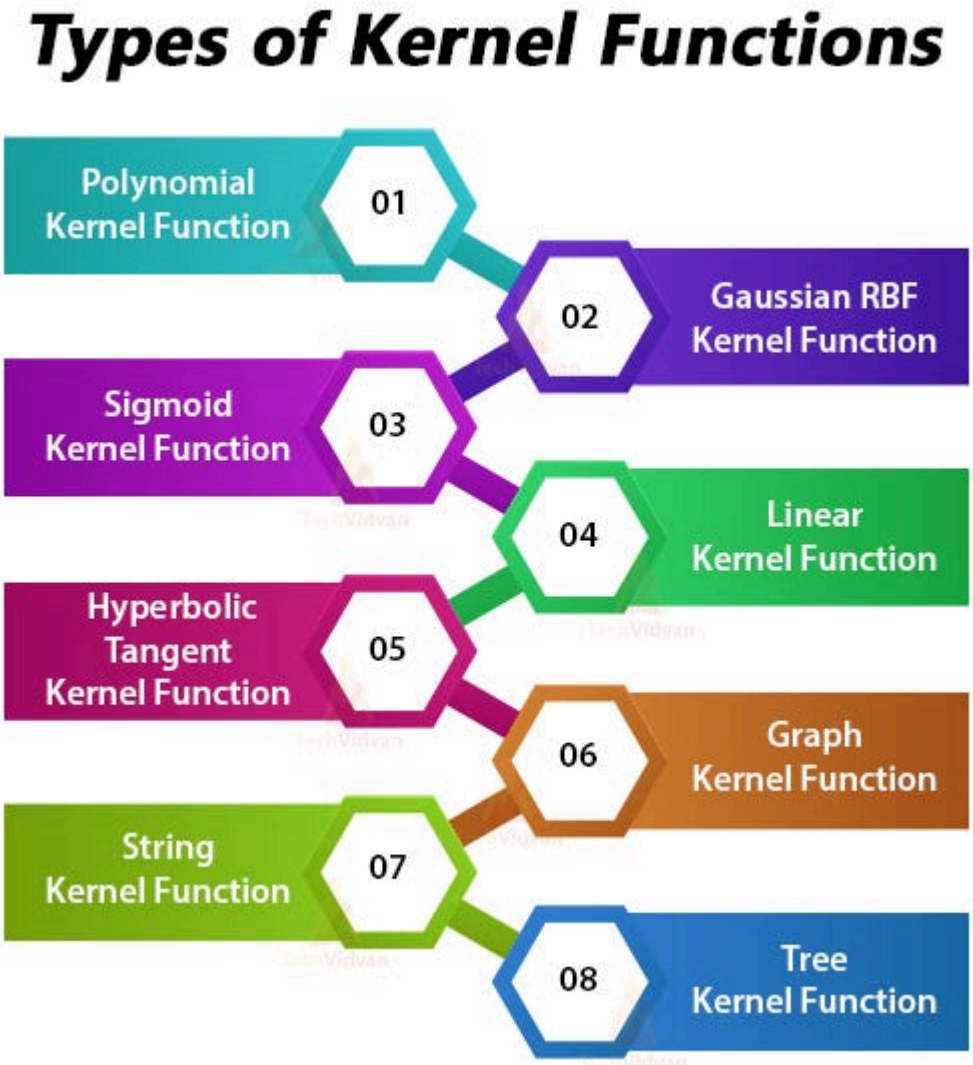
```
              precision    recall  f1-score   support

           0       0.74      0.99      0.85       704
           1       0.92      0.17      0.28       296

    accuracy                           0.75      1000
   macro avg       0.83      0.58      0.56      1000
weighted avg       0.79      0.75      0.68      1000
```

SVM Kernel Functions

A kernel is a function used in SVM for helping to solve problems. With the help of kernel we can go to higher dimensions and perform smooth calculations. We can go up to an infinite number of dimensions using kernels. Kernel plays a vital role in classification and is used to

analyze some patterns in the given dataset. They are very helpful in solving a non-linear problem by using a linear classifier.

Sometimes, we cannot have a hyperplane for certain problems. This problem arises when we go up to higher dimensions and try to form a hyperplane. We have various svm kernel functions to convert the non-linear data to linear. In this notebook, we listed 8 such popular



Now we wiil see how to implement these kernel functions.¶

Linear Kernel

It is the most basic type of kernel, usually one dimensional in nature. It proves to be the best function when there are lots of features.

Linear kernel functions are faster than other functions.

Linear Kernel Formula

F(x, xj) = sum(x.xj)

Here, x, xj represents the data we're trying to classify.

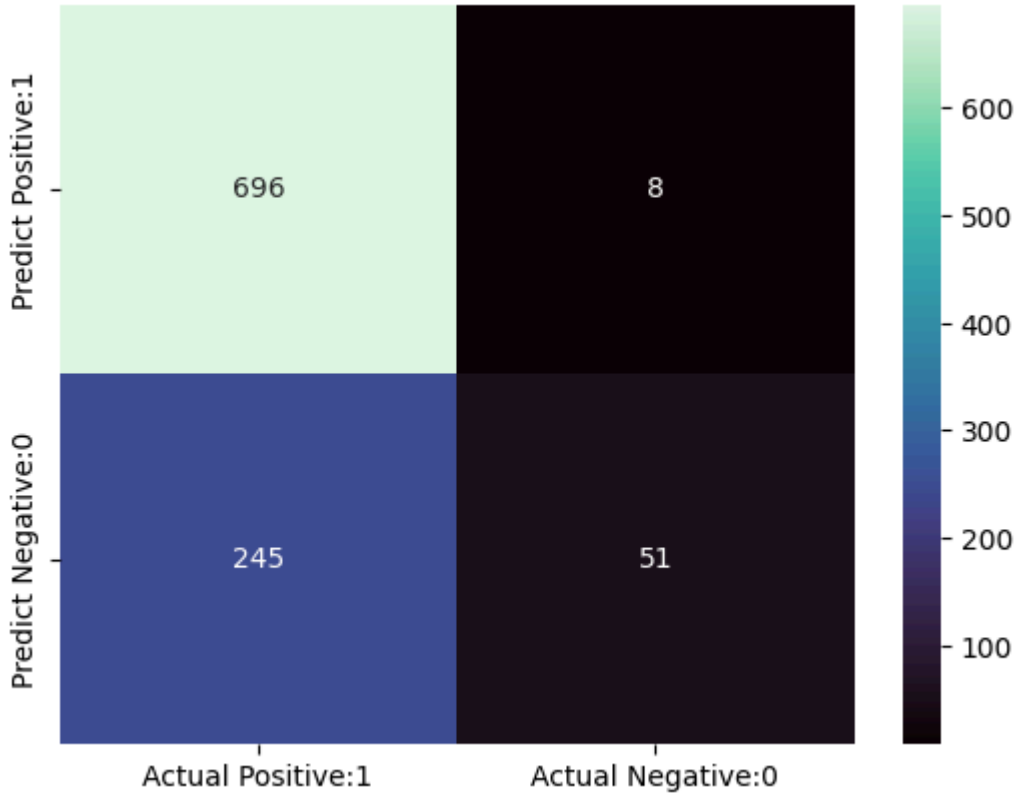
Now we will make our svc classifier using a linear kernel.

In [17]:
```
# Apply SVM model using Linear Kernel function
linear_classifier=SVC(kernel='linear').fit(x_train,y_train)
y_pred = linear_classifier.predict(x_test)
print('Model accuracy with linear kernel : {0:0.3f}'.
format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy with linear kernel : 0.747
```

In [18]:
```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual
Negative:0'],
                                 index=['Predict Positive:1', 'Predict
Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='mako')
```

Out[18]: <Axes: >



In [19]:
```
# Classification Report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.74      0.99      0.85       704
           1       0.86      0.17      0.29       296

    accuracy                           0.75      1000
   macro avg       0.80      0.58      0.57      1000
weighted avg       0.78      0.75      0.68      1000
```

Gaussian RBF kernel¶

It is one of the most preferred and used kernel functions in svm. It is usually chosen for non-linear data. It helps to make proper separation when there is no prior knowledge of data.

Gaussian Radial Basis Formula

F(x, xj) = exp(-gamma * ||x - xj||^2)

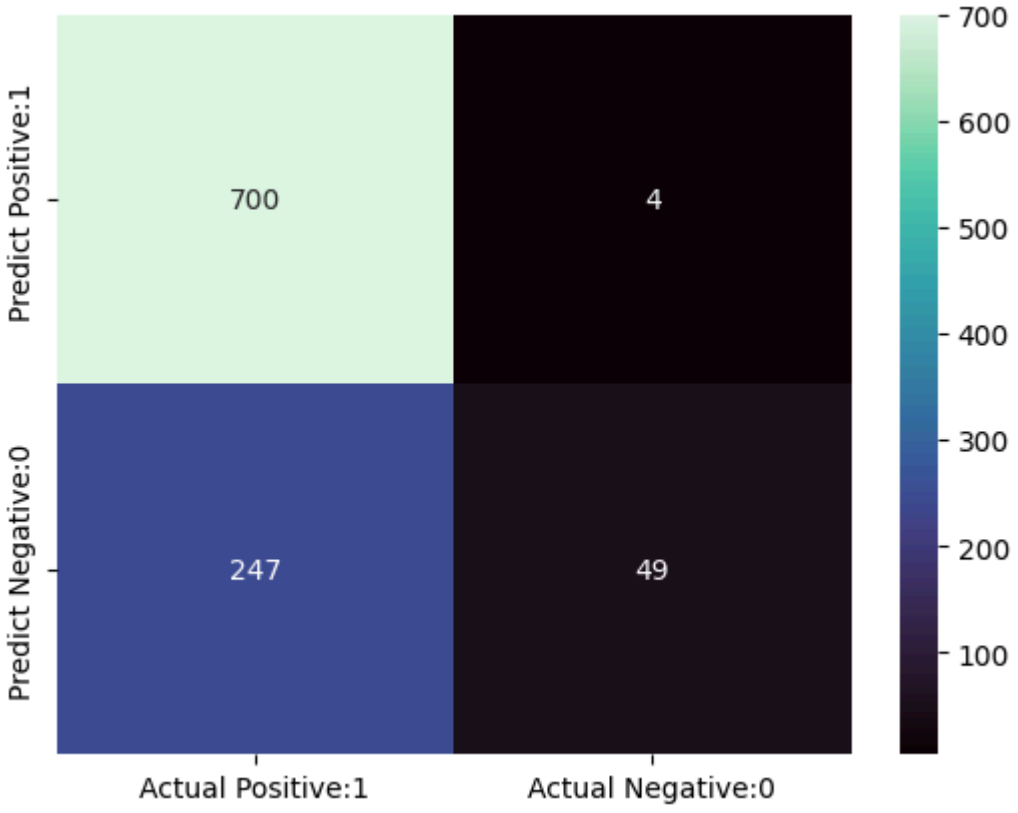
The value of gamma varies from 0 to 1. We have to manually provide the value of gamma in the code. The most preferred value for gamma is 0.1.

In [20]:
```
# Apply SVM model using Gaussian RBF kernel function
rbf_svc=SVC(kernel='rbf').fit(x_train,y_train)
y_pred = rbf_svc.predict(x_test)
print('Model accuracy with rbf kernel : {0:0.3f}'.
format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy with rbf kernel : 0.749
```

In [21]:
```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual
Negative:0'],
                                 index=['Predict Positive:1', 'Predict
Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='mako')
```

Out[21]: <Axes: >



In [22]:
```
# Classification Report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.74      0.99      0.85       704
           1       0.92      0.17      0.28       296

    accuracy                           0.75      1000
   macro avg       0.83      0.58      0.56      1000
weighted avg       0.79      0.75      0.68      1000
```

Polynomial Kernel¶

It is a more generalized representation of the linear kernel. It is not as preferred as other kernel functions as it is less efficient and accurate.

Polynomial Kernel Formula

F(x, xj) = (x.xj+1)^d

Here '.' shows the dot product of both the values, and d denotes the degree.

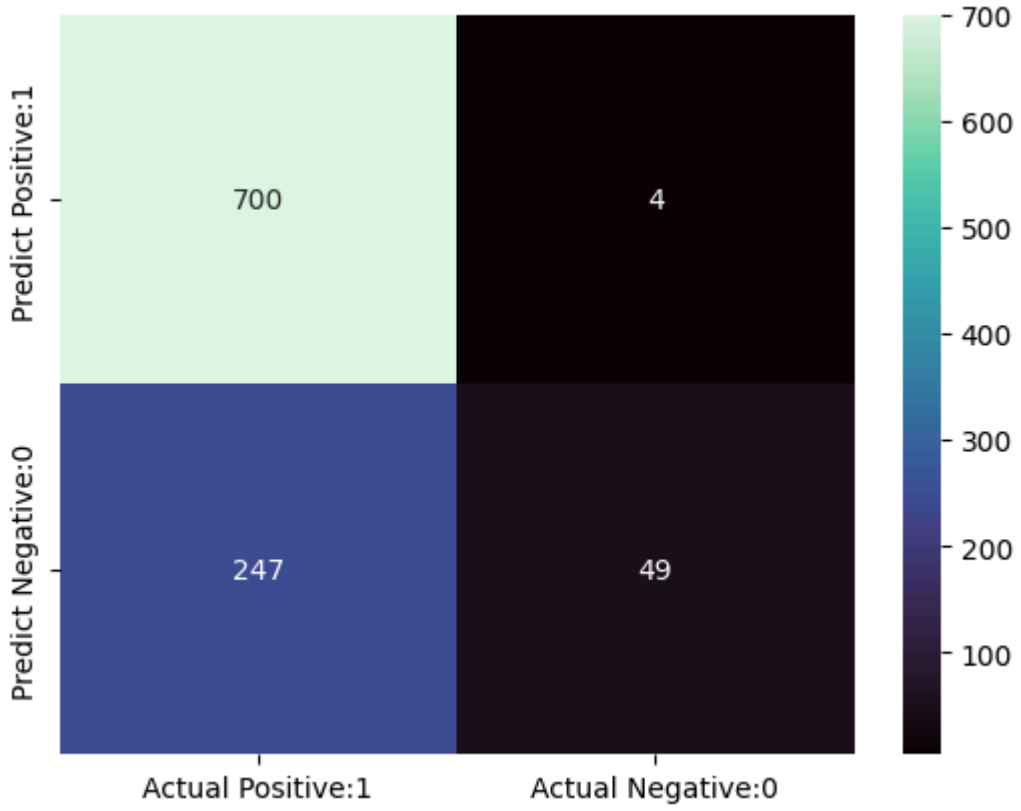
F(x, xj) representing the decision boundary to separate the given classes.

In [23]:
```
# Apply SVM model using Polynomial Kernel function
Poly_svc=SVC(kernel='poly', C=1).fit(x_train,y_train)
y_pred = Poly_svc.predict(x_test)
print('Model accuracy with rbf kernel : {0:0.3f}'.
format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy with rbf kernel : 0.749
```

In [24]:
```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual
Negative:0'],
                                  index=['Predict Positive:1', 'Predict
Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='mako')
```

Out[24]: <Axes: >



In [25]:
```
# Classification Report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.74      0.99      0.85       704
           1       0.92      0.17      0.28       296

    accuracy                           0.75      1000
   macro avg       0.83      0.58      0.56      1000
weighted avg       0.79      0.75      0.68      1000
```

Sigmoid Kernel¶

It is mostly preferred for neural networks. This kernel function is similar to a two-layer perceptron model of the neural network, which works as an activation function for neurons.

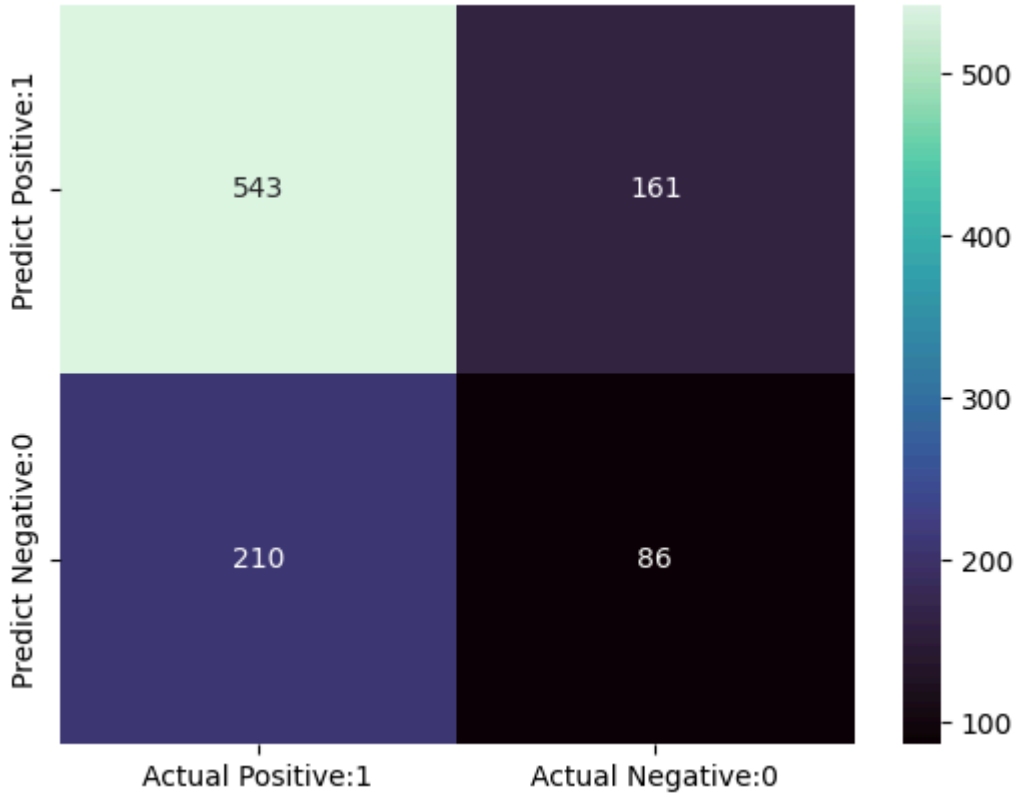
Sigmoid Kenel Formula F(x, xj) = tanh(αxay + c)

In [26]:
```
# Apply SVM model using Sigmoid Kernel function
Poly_svc=SVC(kernel='sigmoid', C=1).fit(x_train,y_train)
y_pred = Poly_svc.predict(x_test)
print('Model accuracy with rbf kernel : {0:0.3f}'.
format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy with rbf kernel : 0.629
```

In [27]:
```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual
Negative:0'],
                                  index=['Predict Positive:1', 'Predict
Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='mako')
```

Out[27]: <Axes: >



In [28]:
```
# Classification Report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.72      0.77      0.75       704
           1       0.35      0.29      0.32       296

    accuracy                           0.63      1000
   macro avg       0.53      0.53      0.53      1000
weighted avg       0.61      0.63      0.62      1000
```

#Among all these above kernels Gaussian RBF kernel is giving more accurate Prediction.
The model accuracy with Gaussian RBF kernel is 0.749

----------Ending----------

In [ ]: