**SOURCE CODE:**

// IMPORTING PACKAGES :

import pandas as pd

import matplotlib.pyplot as plt

from itertools import combinations

from collections import Counter

import numpy as np

//READING DATA:

annual\_sales=pd.read\_csv("C:/Users/surya/OneDrive/Desktop/annual\_sales.csv")

annual\_sales.head()

//Summary of data

annual\_sales.info()

**DATA CLEANING**

//Dropping NaN values

annual\_sales = annual\_sales.dropna(how='all')

annual\_sales.head()

// Removing string in Order Date column

annual\_sales =annual\_sales[annual\_sales['Order Date'].str[0:2]!='Or']

//Adding month column

annual\_sales['Month'] = pd.to\_datetime(annual\_sales['Order Date']).dt.month

// Converting 'Purchased\_Quantity' & 'Cost\_Each' to the correct data type (INT & FLOAT respectively).

annual\_sales['Purchased\_Quantity'] = pd.to\_numeric(annual\_sales['Purchased\_Quantity'])

annual\_sales['Cost\_Each'] = pd.to\_numeric(annual\_sales['Cost\_Each'])

//Adding 'SALES' column

annual\_sales['Sales'] = annual\_sales['Purchased\_Quantity'] \* annual\_sales['Cost\_Each']

// ADDING A CITY COLUMN

annual\_sales['City'] = annual\_sales['Purchase\_Address'].apply(lambda x: x.split(',')[1])

// RE-ARRANGING COLUMNS

annual\_sales=annual\_sales[['OrderID','Name\_Of\_Purchase','Purchased\_Quantity','Cost\_Each','Sales','OrderDate','Purchase\_Address','Month','City']]

//RE-FORMATING 'Order Date' column

annual\_sales['Order Date']= pd.to\_datetime(annual\_sales['Order Date'])

// ADDING HOUR & MINUITE COLUMN

annual\_sales['Hour'] = pd.to\_datetime(annual\_sales['Order Date']).dt.hour

annual\_sales['Minute'] = pd.to\_datetime(annual\_sales['Order Date']).dt.minute

// Finding REDUNDANT values to group them for EDA!

df=annual\_sales[annual\_sales['Order ID'].duplicated(keep=False)]

**Exploratory Data Analysis [EDA]**

//Importing packages

import pandas as pd

import matplotlib.pyplot as plt

from itertools import combinations

from collections import Counter

import numpy as np

import seaborn as sns

from scipy import stats

%matplotlib inline

import matplotlib.font\_manager

//1. Magnitude of sales on monthly basis

cleaned\_data.groupby(['Month']).sum()

months = range(1,13)

print(months)

plt.bar(months,cleaned\_data.groupby(['Month']).sum()['Sales'])

plt.xticks(months)

plt.ylabel('Sales in Millions ($)')

plt.xlabel('Month ')

plt.show()

//2. Product and sales correlation

product\_group=cleaned\_data.groupby('Name\_Of\_Purchase')

Purchased\_Quantity=product\_group.sum()['Purchased\_Quantity']

products= [Name\_Of\_Purchase for Name\_Of\_Purchase,df in product\_group ]

plt.bar(products,Purchased\_Quantity )

plt.ylabel('Purchased\_Quantity')

plt.xlabel('product')

plt.xticks(products, rotation='vertical', size=8)

Cost\_Each=cleaned\_data.groupby('Name\_Of\_Purchase').mean()['Cost\_Each']

fig, ax1=plt.subplots()

ax2 = ax1.twinx()

ax1.bar(products, Purchased\_Quantity)

ax2.plot(products, Cost\_Each, 'b-')

ax1.set\_xlabel('Name\_Of\_Purchase')

ax1.set\_ylabel('Purchased\_Quantity')

ax2.set\_ylabel('Cost\_Each ($)')

ax1.set\_xticklabels(products, rotation='vertical', size=8)

plt.show()

//3. General Correlation using heat-map.

cleaned\_data.corr()

sns.heatmap(cleaned\_data.corr(),annot=True)

//4. Cumulative Probability

H, edges = np.histogram(cleaned\_data.describe().Cost\_Each, bins=np.linspace(0, 2E4, 81))

sales = [5000, 10000, 15000, 20000, 25000, 30000]

fig = plt.figure(figsize=(10, 6))

fig.subplots\_adjust(hspace=0)

ax = plt.subplot(211)

ax.bar(edges[:-1], H / float(sum(H)), width=edges[1] - edges[0])

for ii in sales:

ax.plot([ii, ii], [0, 0.06], linestyle="--", color='k')

ax.set\_ylabel("Probability of product sold")

ax.set\_xticklabels([])

ax.minorticks\_on()

ax = plt.subplot(212)

probabilities = H.cumsum() / float(sum(H))

ax.plot(edges[:-1], probabilities, linewidth=5, color="r")

for ii in incomes:

prob = probabilities[edges[:-1] == ii]

ax.plot([ii, ii], [0, prob], linestyle="--", color='k')

ax.plot([0, ii], [prob, prob], linestyle="--", color='k')

ax.text(ii + 2.5E2, prob \* 0.5, "p(x): %.2f" % prob)

ax.set\_xlabel("Sales)")

ax.set\_ylabel("Cumulative Probability")

ax.set\_ylim(0, 0.99)

ax.minorticks\_on()

plt.show()

//5. Multi-varite Distributional data Summarisation based on products

hi\_volume\_products\_pivots = cleaned\_data.pivot\_table(index='Order Date', columns='Name\_Of\_Purchase', values='Sales')

hi\_volume\_products\_pivots.plot(kind='box', figsize=[16,8])

plt.xticks(rotation=90)

//6. Statistical Calculation of Mean, Standard Deviation and corresponding Z Score

cleaned\_data.Sales.describe()

upper = cleaned\_data.Sales.mean() + 3\*cleaned\_data.Sales.std()

lower = cleaned\_data.Sales.mean() -3\*cleaned\_data.Sales.std()

print(upper)

print(lower)

new\_df= cleaned\_data[(cleaned\_data.Sales<upper) & (cleaned\_data.Sales>lower)]

new\_df.head()

cleaned\_data['zscore'] = ( cleaned\_data.Sales - cleaned\_data.Sales.mean() ) / cleaned\_data.Sales.std()

cleaned\_data.head(5)

cleaned\_data[cleaned\_data['zscore']>3]

//7. Detection of Outliers

plt.title('Outliers Detection')

sns.boxplot(data=cleaned\_data, y = 'Name\_Of\_Purchase', x='Sales')

plt.show()

//8. Covariation in the sales frequency among different cities.

cleaned\_data.groupby(['City']).sum()

import matplotlib.pyplot as plt

keys = [city for city, df in cleaned\_data.groupby(['City'])]

plt.bar(keys,cleaned\_data.groupby(['City']).sum()['Sales'])

plt.ylabel('Sales in USD ($)')

plt.xlabel('Month number')

plt.xticks(keys, rotation='vertical', size=8)

plt.show()

City=["Atlanta","Austin","Boston","Dallas","Los Angeles","New York City","Portland","San Francisco","Seattle"]

Sales\_vals=[2.795499e+06,1.819582e+06,3.661642e+06,2.767975e+06,5.452571e+06,4.664317e+06,2.320491e+06,8.262204e+06,2.747755e+06]

plt.axis("equal")

plt.pie(Sales\_vals,labels=City,radius=2,autopct='%0.1f%%',shadow=True,explode=[0,0.3,0,0,0,0,0,0.3,0],startangle=180)

plt.show()

//9. Identifying the skewness in the data elements

from scipy.stats import skewnorm

plt.title('Data Skewness')

df = cleaned\_data.groupby(['City']).sum()

X = np.linspace(min(df.Sales), max(df.Sales))

plt.plot(X, skewnorm.pdf(X, \*skewnorm.fit(df.Sales)))

df = cleaned\_data.groupby(['State']).sum()

X = np.linspace(min(df.Sales), max(df.Sales))

plt.plot(X, skewnorm.pdf(X, \*skewnorm.fit(df.Sales)))

df = cleaned\_data.groupby(['Month']).sum()

X = np.linspace(min(df.Sales), max(df.Sales))

plt.plot(X, skewnorm.pdf(X, \*skewnorm.fit(df.Sales)))

//10. Geographical potrayal of Sales across the States (USA )

cleaned\_data.groupby(['State']).sum()

import chart\_studio.plotly as py

import plotly.graph\_objs as go

from plotly.offline import download\_plotlyjs, init\_notebook\_mode, plot, iplot

data = dict(type='choropleth',

colorscale = 'YIOrRd',

locations = annual\_sales['State'],

z = annual\_sales['Sales'],

locationmode = 'USA-states',

marker = dict(line = dict(color = 'rgb(255,255,255)',width = 2)),

colorbar = {'title':"Millions USD"}

)

data = dict(type = 'choropleth',

locations = ['WA','CA','NY','GA','MA','ME','OR','TX'],

locationmode = 'USA-states',

colorscale= 'Portland',

text= ['2.747755e+06','1.371477e+07','4.664317e+06','2.795499e+06','3.661642e+06',

'4.497583e+05','1.870732e+06','4.587557e+06'],

z=[1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0],

colorbar = {'title':'Colorbar Title'})

layout = dict(geo = {'scope':'usa'})

choromap = go.Figure(data = [data],layout = layout)

iplot(choromap)

keys = [State for State, df in cleaned\_data.groupby(['State'])]

plt.bar(keys,cleaned\_data.groupby(['State']).sum()['Sales'])

plt.ylabel('Sales in USD ($)')

plt.xlabel('Month number')

plt.xticks(keys, rotation='vertical', size=8)

plt.show()

//11. Develop Ordering Hours and Sales Frequency relation

cleaned\_data['Hour'] = pd.to\_datetime(cleaned\_data['Order Date']).dt.hour

cleaned\_data['Minute'] = pd.to\_datetime(cleaned\_data['Order Date']).dt.minute

cleaned\_data['Count'] = 1

keys = [pair for pair, df in cleaned\_data.groupby(['Hour'])]

plt.plot(keys, cleaned\_data.groupby(['Hour']).count()['Count'])

plt.xticks(keys)

plt.grid()

plt.show()

//12. Combination of products ordered together

df = cleaned\_data[cleaned\_data['Order ID'].duplicated(keep=False)]

df['Grouped'] = df.groupby('Order ID')['Name\_Of\_Purchase'].transform(lambda x: ','.join(x))

df2 = df[['Order ID', 'Grouped']].drop\_duplicates()

from itertools import combinations

from collections import Counter

count = Counter()

for row in df2['Grouped']:

row\_list = row.split(',')

count.update(Counter(combinations(row\_list, 2)))

for key,value in count.most\_common(10):

print(key, value)

print("@@The sorting of products using a pivot table@@:")

cleaned\_data.pivot\_table(columns='Name\_Of\_Purchase', values='Sales').sort\_values

//13. Determining the density distribution for cost\_Each variable.

sns.distplot(clean\_data['Cost\_Each'],kde=False,bins=20)

**MODELLING/APPLYING ML ALGORITHMS**

// TEST-TRAIN SPLIT

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data, label, test\_size = 0.3, random\_state = 42)

print("Training set has {} samples.".format(X\_train.shape[0]))

print("Testing set has {} samples.".format(X\_test.shape[0]))

// IMPORTING REQUIRED ML ALGORITHMS

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from xgboost import XGBClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

from sklearn.metrics import roc\_auc\_score,roc\_curve

from sklearn.model\_selection import RandomizedSearchCV, GridSearchCV, train\_test\_split

from sklearn.ensemble import AdaBoostClassifier

from sklearn.utils import resample

import warnings

warnings.filterwarnings("ignore")

// DEFINING THE FUNCTION CLASSIFIER

def apply\_classifier(clf,xTrain,xTest,yTrain,yTest):

clf.fit(xTrain, yTrain)

predictions = clf.predict(xTest)

conf\_mtx = confusion\_matrix(yTest,predictions)

f, axes = plt.subplots(ncols=2, figsize=(15, 5))

sns.heatmap(conf\_mtx,annot=True,cmap='tab20c',cbar = False,fmt = "g",ax = axes[0])

axes[0].set\_xlabel('Predicted labels')

axes[0].set\_ylabel('True labels')

axes[0].set\_title('Confusion Matrix');

axes[0].xaxis.set\_ticklabels(['Not SOLD', 'SOLD']);

axes[0].yaxis.set\_ticklabels(['Not SOLD', 'SOLD']);

print("\n Classification report : \n {}".format(classification\_report(yTest,predictions)))

roc\_auc = roc\_auc\_score(yTest,predictions)

print ("Area under ROC curve : ",roc\_auc,"\n")

fpr, tpr,\_ = roc\_curve(yTest, predictions)

axes[1].plot(fpr,tpr,label= "auc="+str(roc\_auc));

axes[1].plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic')

plt.legend(loc="lower right")

// DEFINING GRID SEARCH

def grid\_search(clf,parameters,xTrain,Ytrain):

grid\_obj = GridSearchCV(clf,parameters,scoring = 'roc\_auc',cv = 5)

grid\_fit = grid\_obj.fit(xTrain,Ytrain)

best\_clf = grid\_fit.best\_estimator\_

return best\_clf

//1.LOGISTIC REGRESSION ALGORITHM

def apply\_classifier2(clf,xTrain,xTest,yTrain,yTest):

clf.fit(xTrain, yTrain)

predictions = clf.predict(xTest)

conf\_mtx = confusion\_matrix(yTest,predictions)

f, axes = plt.subplots(ncols=2, figsize=(15, 5))

sns.heatmap(conf\_mtx,annot=True,cmap='tab20c',cbar = False,fmt = "g",ax = axes[0])

axes[0].set\_xlabel('Predicted labels')

axes[0].set\_ylabel('True labels')

axes[0].set\_title('Confusion Matrix');

axes[0].xaxis.set\_ticklabels(['Responsive', 'Non Responsive']);

axes[0].yaxis.set\_ticklabels(['Responsive', 'Non Responsive']);

print("\n Classification report : \n {}".format(classification\_report(yTest,predictions)))

roc\_auc = roc\_auc\_score(yTest,predictions)

print ("Area under ROC curve : ",roc\_auc,"\n")

fpr, tpr,\_ = roc\_curve(yTest, predictions)

axes[1].plot(fpr,tpr,label= "auc="+str(roc\_auc));

axes[1].plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic')

plt.legend(loc="lower right")

logistic\_reg = LogisticRegression(random\_state = 42)

apply\_classifier2(logistic\_reg,X\_train, X\_test, y\_train, y\_test)

// HYPER PARAMETER TUNING OF LOGISTIC REGRESSION

LogReg\_parameters = {

"C":[0.25,0.5,0.75,1.0,1.5,2.0,2.5,3.0,4.0,10.0],

"solver":["newton-cg", "lbfgs", "sag", "saga"],

"tol":[0.01,0.001,0.0001,0.00001],

"warm\_start":["True","False"]}

logReg\_grid = grid\_search(logistic\_reg,LogReg\_parameters,X\_train,y\_train);

apply\_classifier2(logReg\_grid,X\_train, X\_test, y\_train, y\_test)

//Visualizing Logistic Regression Algorithm

clf = linear\_model.LogisticRegression(C=1e5)

clf.fit(X, y)

plt.figure(1, figsize=(4, 3))

plt.clf()

plt.scatter(X.ravel(), y, color='black', zorder=20)

X\_test = np.linspace(-5, 10, 300)

loss = expit(X\_test \* clf.coef\_ + clf.intercept\_).ravel()

plt.plot(X\_test, loss, color='red', linewidth=3)

ols = linear\_model.LinearRegression()

ols.fit(X, y)

plt.plot(X\_test, ols.coef\_ \* X\_test + ols.intercept\_, linewidth=1)

plt.axhline(.5, color='.5')

plt.ylabel('Responsiveness')

plt.xlabel('Non Responsiveness')

plt.xticks(range(-5, 10))

plt.yticks([0, 0.5, 1])

plt.ylim(-.25, 1.25)

plt.xlim(-4, 10)

plt.legend(('Logistic Regression Model', 'Linear Regression Model'),

loc="lower right", fontsize='small')

plt.tight\_layout()

plt.show()

//2. DECISION TREE ALGORITHM

def apply\_classifier1(clf,xTrain,xTest,yTrain,yTest):

clf.fit(xTrain, yTrain)

predictions = clf.predict(xTest)

conf\_mtx = confusion\_matrix(yTest,predictions)

f, axes = plt.subplots(ncols=2, figsize=(15, 5))

sns.heatmap(conf\_mtx,annot=True,cmap='tab20c',cbar = False,fmt = "g",ax = axes[0])

axes[0].set\_xlabel('Predicted labels')

axes[0].set\_ylabel('True labels')

axes[0].set\_title('Confusion Matrix');

axes[0].xaxis.set\_ticklabels(['SOLD', 'NOT SOLD']);

axes[0].yaxis.set\_ticklabels(['SOLD', 'NOT SOLD']);

print("\n Classification report : \n {}".format(classification\_report(yTest,predictions)))

roc\_auc = roc\_auc\_score(yTest,predictions)

print ("Area under ROC curve : ",roc\_auc,"\n")

fpr, tpr,\_ = roc\_curve(yTest, predictions)

axes[1].plot(fpr,tpr,label= "auc="+str(roc\_auc));

axes[1].plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic')

plt.legend(loc="lower right")

decision\_tree = DecisionTreeClassifier(random\_state = 42);

apply\_classifier1(decision\_tree,X\_train, X\_test, y\_train, y\_test)

// HYPER PARAMETER TUNING OF DECISION TREE

Tree\_parameters = {"max\_depth": [20,21,22,23],

"min\_samples\_leaf":[19,20,21,22]}

tree\_grid = grid\_search(decision\_tree,Tree\_parameters,X\_train,y\_train);

apply\_classifier(tree\_grid,X\_train, X\_test, y\_train, y\_test)

//Visualizing DECISION TREE ALGORITHM

from sklearn import tree

Tree\_parameters = {"max\_depth": [2],

"min\_samples\_leaf":[1]}

tree\_grid = grid\_search(decision\_tree,Tree\_parameters,X\_train,y\_train);

decision\_tree = DecisionTreeClassifier(max\_depth = 2,

random\_state = 0)

tree.plot\_tree(tree\_grid);

//3. RANDOM FOREST ALGORITHM

def apply\_classifier3(clf,xTrain,xTest,yTrain,yTest):

clf.fit(xTrain, yTrain)

predictions = clf.predict(xTest)

conf\_mtx = confusion\_matrix(yTest,predictions)

f, axes = plt.subplots(ncols=2, figsize=(15, 5))

sns.heatmap(conf\_mtx,annot=True,cmap='tab20c',cbar = False,fmt = "g",ax = axes[0])

axes[0].set\_xlabel('Predicted labels')

axes[0].set\_ylabel('True labels')

axes[0].set\_title('Confusion Matrix');

axes[0].xaxis.set\_ticklabels(['ORDERED', 'NOT ORDERED']);

axes[0].yaxis.set\_ticklabels(['ORDERED', 'NOT ORDERED']);

print("\n Classification report : \n {}".format(classification\_report(yTest,predictions)))

roc\_auc = roc\_auc\_score(yTest,predictions)

print ("Area under ROC curve : ",roc\_auc,"\n")

fpr, tpr,\_ = roc\_curve(yTest, predictions)

axes[1].plot(fpr,tpr,label= "auc="+str(roc\_auc));

axes[1].plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic')

plt.legend(loc="lower right")

random\_forest = RandomForestClassifier(random\_state = 42)

apply\_classifier3(random\_forest,X\_train, X\_test, y\_train, y\_test)

// HYPER PARAMETER TUNING OF RANDOM FOREST

RandomForest\_parameters = {

"n\_estimators" :[10,15,20,25,30],

"criterion": ["entropy","gini"],

"max\_depth" : [5,10,15],

"min\_samples\_split":[2,4,8,16],

"max\_features":["sqrt","auto","log2"],

"class\_weight" : ["balanced\_subsample","balanced"]}

randomForest\_grid = grid\_search(random\_forest,RandomForest\_parameters,X\_train,y\_train);

apply\_classifier3(randomForest\_grid,X\_train, X\_test, y\_train, y\_test)

// USING ADABOOST TO INCREASE THE ACCURACY of Random Forest

model = AdaBoostClassifier(base\_estimator = randomForest\_grid, n\_estimators = 4)

apply\_classifier3(model,X\_train, X\_test, y\_train, y\_test)

// RESAMPLING TO IMPROVE ACCURACY of Random Forest

upsample\_data = data\_original

majority = upsample\_data[upsample\_data["Churn"]==0]

minority = upsample\_data[upsample\_data["Churn"]==1]

minority\_upsampled = resample(minority, replace=True, n\_samples=5163,random\_state=42)

del(upsample\_data)

upsample\_data = pd.concat([majority,minority\_upsampled])

id\_customer\_upsample = upsample\_data["customerID"]

label\_upsample = upsample\_data["Churn"]

upsample\_data.drop("Churn",inplace = True, axis = 1)

upsample\_data.drop("customerID",inplace = True, axis = 1)

from sklearn.model\_selection import train\_test\_split

X\_train\_upS, X\_test\_upS, y\_train\_upS, y\_test\_upS = train\_test\_split(upsample\_data, label\_upsample, test\_size = 0.3, random\_state = 42)

print("Training set has {} samples.".format(X\_train\_upS.shape[0]))

print("Testing set has {} samples.".format(X\_test\_upS.shape[0]))

model = AdaBoostClassifier(base\_estimator = random\_forest, n\_estimators = 4)

apply\_classifier(model,X\_train\_upS, X\_test\_upS, y\_train\_upS, y\_test\_upS)

//4.SVM ALGORITHM

def apply\_classifier4(clf,xTrain,xTest,yTrain,yTest):

clf.fit(xTrain, yTrain)

predictions = clf.predict(xTest)

conf\_mtx = confusion\_matrix(yTest,predictions)

f, axes = plt.subplots(ncols=2, figsize=(15, 5))

sns.heatmap(conf\_mtx,annot=True,cmap='tab20c',cbar = False,fmt = "g",ax = axes[0])

axes[0].set\_xlabel('Predicted labels')

axes[0].set\_ylabel('True labels')

axes[0].set\_title('Confusion Matrix');

axes[0].xaxis.set\_ticklabels(['Best HOur', 'Bad Hour']);

axes[0].yaxis.set\_ticklabels(['Best Hour', 'Bad Hour']);

print("\n Classification report : \n {}".format(classification\_report(yTest,predictions)))

roc\_auc = roc\_auc\_score(yTest,predictions)

print ("Area under ROC curve : ",roc\_auc,"\n")

fpr, tpr,\_ = roc\_curve(yTest, predictions)

axes[1].plot(fpr,tpr,label= "auc="+str(roc\_auc));

axes[1].plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic')

plt.legend(loc="lower right")

svm\_model = SVC(random\_state = 42)

apply\_classifier4(svm\_model,X\_train, X\_test, y\_train, y\_test)

// HYPER PARAMETER TUNING OF SVM

SVM\_parameters = {

"C":[1.0,2.0,3.0],

"cache\_size":[100,200],

"decision\_function\_shape":['ovo','ovr'],

"kernel":['sigmoid',"linear"],

"tol":[0.001,0.0001]}

svm\_grid = grid\_search(svm\_model,SVM\_parameters,X\_train,y\_train);

apply\_classifier4(svm\_grid,X\_train, X\_test, y\_train, y\_test)

//Visualizing SVM Algorithm to plot results

import numpy as np

import matplotlib.pyplot as plt

from sklearn import svm

np.random.seed(2)

X = np.r\_[np.random.randn(20, 2) - [2, 2], np.random.randn(20, 2) + [2, 2]]

Y = [0] \* 20 + [1] \* 20

# fit the model

clf = svm.SVC(kernel='linear', C=1)

clf.fit(X, Y)

w = clf.coef\_[0]

a = -w[0] / w[1]

xx = np.linspace(-5, 5)

yy = a \* xx - (clf.intercept\_[0]) / w[1]

margin = 1 / np.sqrt(np.sum(clf.coef\_ \*\* 2))

yy\_down = yy - np.sqrt(1 + a \*\* 2) \* margin

yy\_up = yy + np.sqrt(1 + a \*\* 2) \* margin

plt.figure(1, figsize=(4, 3))

plt.clf()

plt.plot(xx, yy, "k-")

plt.plot(xx, yy\_down, "k-")

plt.plot(xx, yy\_up, "k-")

plt.scatter(clf.support\_vectors\_[:, 0], clf.support\_vectors\_[:, 1], s=80,

facecolors="none", zorder=10, edgecolors="k")

plt.scatter(X[:, 0], X[:, 1], c=Y, zorder=10, cmap=plt.cm.Paired,

edgecolors="k")

plt.xlabel("Best Hour")

plt.ylabel("Bad Hour")

plt.show()

//5. NAIVE BAYES ALGORITHM

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data, label, test\_size = 0.3, random\_state = 42)

print("Training set has {} samples.".format(X\_train.shape[0]))

print("Testing set has {} samples.".format(X\_test.shape[0]))

m sklearn.naive\_bayes import GaussianNB

gnb = GaussianNB()

gnb.fit(X\_train, y\_train)

y\_pred = gnb.predict(X\_test)

apply\_classifier(gnbb,X\_train\_upS, X\_test\_upS, y\_train\_upS, y\_test\_upS)

//Result of Naïve Bayes

from sklearn.datasets import make\_blobs

X, y = make\_blobs(100, 2, centers=2, random\_state=2, cluster\_std=1.5)

fig, ax = plt.subplots()

f, axes = plt.subplots(ncols=2, figsize=(15, 5))

sns.heatmap(conf\_mtx,annot=True,cmap='tab20c',cbar = False,fmt = "g",ax = axes[0])

axes[0].set\_xlabel('Predicted labels')

axes[0].set\_ylabel('True labels')

axes[0].set\_title('Confusion Matrix');

axes[0].xaxis.set\_ticklabels(['Best Month', 'Bad Month']);

axes[0].yaxis.set\_ticklabels(['Best Month', 'Bad Month']);

for label, color in enumerate(['red', 'blue']):

mask = (y == label)

mu, std = X[mask].mean(0), X[mask].std(0)

P = np.exp(-0.5 \* (Xgrid - mu) \*\* 2 / std \*\* 2).prod(1)

Pm = np.ma.masked\_array(P, P < 0.03)

ax.pcolorfast(xg, yg, Pm.reshape(xx.shape), alpha=0.5,

cmap=color.title() + 's')

ax.contour(xx, yy, P.reshape(xx.shape),

levels=[0.01, 0.1, 0.5, 0.9],

colors=color, alpha=0.2)