**AchyuthReddy.k**

**M.tech**  **DataScience**

**MACHINE LEARNING WITH PYTHON**

|  |  |  |
| --- | --- | --- |
| **Model** | **Package** | **Class** |
| LinearRegression | sklearn.linear\_model | LinearRegression |
| Multiple Linear Regression | sklearn.linear\_model | LinearRegression |
| Polynomial Regression | sklearn.linear\_model | LinearRegression |
| Support Vector Regression | sklearn.svm | SVR |
| Decision Tree Regression | sklearn.tree | DecisionTreeRegression |
| RandomForest Regression | sklearn.ensemble | RandomForestRegression |
| Logistic Regression | sklearn.linear\_model | LogisticRegression |
| KNN | sklearn.n neighbors | KNeighborsClassifier |
| Support vector Machine(SVM) | sklearn.svm | SVC |
| Kernel svm | sklearn.svm | SVC |
| Decision Tree Classification | sklearn.tree | DecisionTreeClassification |
| Random Forest Classification | sklearn.ensemble | RandomForestClassification |
| Naive Bayes | sklearn.naive\_bayes | GaussianNB |
| K-Means Cluster | sklearn.cluster | KMeans |
| Hierarchical\_Clustering | sklearn.cluster | AgglomerativeClustering |
| Apriori-ARM | apyori | apriori |
| Upper Confidence Bound(UCB) | NaN | NaN |
| Thompson Sampling | NaN | NaN |
| NLP | nltk | -- |
| Principal Component Analysis | sklearn.decomposition | PCA |
| LinearDiscriminant  Analysis | sklearn.  discriminant\_analysis | LinearDiscriminantAnalysis |
| Kernel PCA | sklearn.decomposition | KernelPCA |
| K-Fold Cross Validation | sklearn.model\_selection | cross\_val\_score |
| Grid Search | sklearn.model\_selection | GridSearchCV |
| XGBoost | xgboost | XGBClassifier |

TP = # True Positives,

TN = # True Negatives,

FP = # False Positives,

FN = # False Negatives)

**To test Model Performance::**

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Precision = TP / (TP + FP)

Recall = TP / (TP + FN)

F1 Score = 2 \* Precision \* Recall / (Precision + Recall)

**Data Preprocessing in Python**

#importing lib  
import pandas as pd  
import numpy as ny  
import matplotlib.pyplot as plt  
  
#importing dataset  
data=pd.read\_csv("Data.csv")  
X=data.iloc[:,:-1]  
y=data.iloc[:,-1:]  
  
#taking care of missing data  
from sklearn.preprocessing import Imputer  
imputer=Imputer(missing\_values=ny.nan,strategy="mean",axis=0)  
imputer=imputer.fit(X.iloc[:,1:3])  
X.iloc[:,1:3]=imputer.transform(X.iloc[:,1:3])  
  
#categorical data  
from sklearn.preprocessing import LabelEncoder,OneHotEncoder  
lab\_X=LabelEncoder()  
X.iloc[:,0]=lab\_X.fit\_transform(X.iloc[:,0])  
onehot=OneHotEncoder(categorical\_features=[0])  
X=onehot.fit\_transform(X).toarray()  
  
#splitting data into train and test sets  
from sklearn.model\_selection import train\_test\_split  
X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state=0)  
  
#feature scaling  
from sklearn.preprocessing import StandardScaler  
sc=StandardScaler()  
X\_train=sc.fit\_transform(X\_train)  
X\_test=sc.transform(X\_test)

**Regression Algorithms**

**Simple Linear Regression**

#importing Lib  
import pandas as pd  
import numpy as ny  
import matplotlib.pyplot as plt  
  
#importing data   
data=pd.read\_csv("Salary\_Data.csv")  
X=data.iloc[:,:-1]  
y=data.iloc[:,1:]  
  
#splitting train and testset  
from sklearn.model\_selection import train\_test\_split  
X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state=0)  
  
#model building  
from sklearn.linear\_model import LinearRegression  
reg=LinearRegression()  
reg.fit(X\_train,y\_train)  
  
#prediction on test set  
y\_pred=reg.predict(X\_test)  
  
#visualising the training data  
plt.scatter(X\_train,y\_train,color='red')  
plt.plot(X\_train,reg.predict(X\_train),color="blue")  
plt.title("Sal vs Yrs Exp(trainset)")  
plt.ylabel("Salary")  
plt.xlabel("Years Of Exp")  
plt.show()  
  
#visualising the test data  
plt.scatter(X\_test,y\_test,color='red')  
plt.plot(X\_test,reg.predict(X\_test),color="blue")  
plt.title("Sal vs Yrs Exp(testset)")  
plt.ylabel("Salary")  
plt.xlabel("Years Of Exp")  
plt.show()

**Multiple Linear Regression**

#import libraries  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
  
#importing data   
data=pd.read\_csv("50\_Startups.csv")  
X=data.iloc[:,:-1]  
y=data.iloc[:,-1]  
  
#categorical data  
from sklearn.preprocessing import LabelEncoder,OneHotEncoder  
lab\_enc=LabelEncoder()  
X.iloc[:,3]=lab\_enc.fit\_transform(X.iloc[:,3])  
one\_hot=OneHotEncoder(categorical\_features=[3])  
X=one\_hot.fit\_transform(X).toarray()  
  
#dummy variable trap  
X=X[:,1:]  
  
#feature scaling  
from sklearn.preprocessing import StandardScaler  
sc=StandardScaler()  
X=sc.fit\_transform(X)  
  
#splitting data into train and test set  
from sklearn.model\_selection import train\_test\_split  
X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25)  
  
#model building  
from sklearn.linear\_model import LinearRegression  
reg=LinearRegression()  
reg.fit(X\_train,y\_train)  
  
#prediction on test set  
y\_pred=reg.predict(X\_test)  
  
  
#back elimination  
import statsmodels.formula.api as sf  
X=np.append(arr=np.ones((50,1)).astype(int),values=X,axis=1)  
X\_opt=X[:,[0,1,2,3,4,5]]  
reg\_ols=sf.OLS(endog=y,exog=X\_opt).fit()  
reg\_ols.summary()  
X\_opt=X[:,[0,1,2,3,5]]  
reg\_ols=sf.OLS(endog=y,exog=X\_opt).fit()  
reg\_ols.summary()  
X\_opt=X[:,[0,1,2,5]]  
reg\_ols=sf.OLS(endog=y,exog=X\_opt).fit()  
reg\_ols.summary()  
  
#model build after elimination  
X=X[:,[1,2,5]]  
  
#feature scaling  
from sklearn.preprocessing import StandardScaler  
sc=StandardScaler()  
X=sc.fit\_transform(X)  
  
#splitting data into train and test set  
from sklearn.model\_selection import train\_test\_split  
X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25)  
  
#model building  
from sklearn.linear\_model import LinearRegression  
reg=LinearRegression()  
reg.fit(X\_train,y\_train)  
  
#prediction on test set  
y\_pred\_elim=reg.predict(X\_test)

**Polynomial Regression**

#importing libraries  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
  
#importing dataset  
data=pd.read\_csv("Position\_Salaries.csv")  
X=data.iloc[:,1:2].values  
y=data.iloc[:,2].values  
  
  
#LinerReg model building  
from sklearn.linear\_model import LinearRegression  
lin\_reg=LinearRegression()  
lin\_reg.fit(X,y)  
  
#BUilding a polynomial Reg model  
from sklearn.preprocessing import PolynomialFeatures  
ploy\_fea=PolynomialFeatures(degree=4)  
X\_ply=ploy\_fea.fit\_transform(X)  
poly\_reg=LinearRegression()  
poly\_reg.fit(X\_ply,y)  
  
#visualising the data in both liner and polynomial  
X\_grid=np.arange(min(X),max(X),0.1)  
X\_grid=X\_grid.reshape(len(X\_grid),1)  
  
plt.scatter(X,y,color="red")  
plt.plot(X,lin\_reg.predict(X),color="blue")  
plt.plot(X\_grid,poly\_reg.predict(ploy\_fea.fit\_transform(X\_grid)),color="black")  
plt.title("Salary Detect")  
plt.xlabel("Years Of Exp")  
plt.ylabel("Salary")

**Support Vector Regression**

#importing librar  
import pandas as pd  
import numpy as ny  
import matplotlib.pyplot as plt  
  
#importing dataset  
data=pd.read\_csv("Position\_Salaries.csv")  
X=data.iloc[:,1:2]  
y=data.iloc[:,2:]  
  
#feature scaling  
from sklearn.preprocessing import StandardScaler  
sc\_X=StandardScaler()  
sc\_y=StandardScaler()  
X=sc\_X.fit\_transform(X)  
y=sc\_y.fit\_transform(y)  
  
#build the model  
from sklearn.svm import SVR  
reg=SVR(kernel="rbf")  
reg.fit(X,y)  
  
#prediction on particular value  
y\_pred=sc\_y.inverse\_transform(reg.predict(sc\_X.transform(ny.array([[6.5]]))))  
  
#visualising the model  
plt.scatter(X,y,color="black")  
plt.plot(X,reg.predict(X),color="red")  
plt.title("Salary Detect(SVR)")  
plt.xlabel("Exper")  
plt.ylabel("Salary")  
plt.show()

**Decision Tree Regression**

#import packages  
import pandas as pd  
import numpy as ny  
import matplotlib.pyplot as plt  
  
#import data set  
data=pd.read\_csv("Position\_Salaries.csv")  
X=data.iloc[:, 1:2].values  
y=data.iloc[:, 2].values  
  
#model devlop  
from sklearn.tree import DecisionTreeRegressor  
reg=DecisionTreeRegressor(random\_state=0)  
reg.fit(X,y)

#model prediction  
y\_pred=reg.predict(6.5)  
  
#visu  
X\_grid=ny.arange(min(X),max(X),0.1)  
X\_grid=X\_grid.reshape((len(X\_grid),1))  
plt.scatter(X,y,color="red")  
plt.plot(X\_grid,reg.predict(X\_grid),color='green')  
plt.title("D.T.Regressor")  
plt.xlabel("Position")  
plt.ylabel("Salary")  
plt.show()

**Random Forest Regression**

#import packages  
import pandas as pd  
import numpy as ny  
import matplotlib.pyplot as plt  
  
#import data set  
data=pd.read\_csv("Position\_Salaries.csv")  
X=data.iloc[:, 1:2].values  
y=data.iloc[:, 2].values  
  
#model devlop  
from sklearn.tree import DecisionTreeRegressor  
reg=DecisionTreeRegressor(random\_state=0)  
reg.fit(X,y)  
  
#model prediction  
y\_pred=reg.predict(6.5)  
#visu  
X\_grid=ny.arange(min(X),max(X),0.1)  
X\_grid=X\_grid.reshape((len(X\_grid),1))  
plt.scatter(X,y,color="red")  
plt.plot(X\_grid,reg.predict(X\_grid),color='green')  
plt.title("D.T.Regressor")  
plt.xlabel("Position")  
plt.ylabel("Salary")  
plt.show()

**Classification Algorithms**

**Logistic Regression**

#importing packages  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
  
#importing datasets  
data=pd.read\_csv("E:\\MY\_GOAL\\Machine Learning\\Part 3 - Classification\\Section 14 - Logistic Regression\\Social\_Network\_Ads.csv")  
X=data.iloc[:,2 : 4].values  
y=data.iloc[:,-1].values  
  
#splitting data into train and test sets  
from sklearn.model\_selection import train\_test\_split  
X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state=0)  
  
#Feature scaling  
from sklearn.preprocessing import StandardScaler  
sc\_X=StandardScaler()  
X\_train=sc\_X.fit\_transform(X\_train)  
X\_test=sc\_X.transform(X\_test)  
  
#Build the model  
from sklearn.linear\_model import LogisticRegression  
reg=LogisticRegression()  
reg.fit(X\_train,y\_train)  
  
#test the model  
y\_pred=reg.predict(X\_test)  
  
  
#model\_performance  
from sklearn.metrics import confusion\_matrix  
cm=confusion\_matrix(y\_test,y\_pred)  
  
count=len(X\_test)  
crt\_pre=0  
for i in range(count):  
 if y\_test[i]==y\_pred[i]:  
 crt\_pre=crt\_pre+1;  
Accu=(crt\_pre/count) \* 100  
print("Model Accure:",Accu)  
  
# Visualising the Training set results  
from matplotlib.colors import ListedColormap  
X\_set, y\_set = X\_train, y\_train  
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  
 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))  
plt.contourf(X1, X2, reg.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  
 alpha = 0.75, cmap = ListedColormap(('red', 'green')))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i, j in enumerate(np.unique(y\_set)):  
 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  
 c = ListedColormap(('red', 'green'))(i), label = j)  
plt.title('Logistic Regression (Training set)')  
plt.xlabel('Age')  
plt.ylabel('Estimated Salary')  
plt.legend()  
plt.show()  
  
# Visualising the Test set results  
from matplotlib.colors import ListedColormap  
X\_set, y\_set = X\_test, y\_test  
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  
 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))  
plt.contourf(X1, X2, reg.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  
 alpha = 0.75, cmap = ListedColormap(('red', 'green')))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i, j in enumerate(np.unique(y\_set)):  
 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  
 c = ListedColormap(('red', 'green'))(i), label = j)  
plt.title('Logistic Regression (Test set)')  
plt.xlabel('Age')  
plt.ylabel('Estimated Salary')  
plt.legend()  
plt.show()

**K-Nearest Neighbor(KNN)**

#importing lib  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
  
#importing data  
data=pd.read\_csv("E:\\MY\_GOAL\\Machine Learning\\Part 3 - Classification\\Section 14 - Logistic Regression\\Social\_Network\_Ads.csv")  
X=data.iloc[:,2 : 4].values  
y=data.iloc[:,-1].values  
  
#Splittting data  
from sklearn.model\_selection import train\_test\_split  
X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state=0)  
  
#Feature scaling  
from sklearn.preprocessing import StandardScaler  
sc\_X=StandardScaler()  
X\_train=sc\_X.fit\_transform(X\_train)  
X\_test=sc\_X.transform(X\_test)

#Build the model KNN  
from sklearn.neighbors import KNeighborsClassifier  
classifier=KNeighborsClassifier(n\_neighbors=5,metric='minkowski',p=2)  
classifier.fit(X\_train,y\_train)  
  
#Test the model  
y\_pred=classifier.predict(X\_test)  
  
#confusion matrix  
from sklearn.metrics import confusion\_matrix  
cm=confusion\_matrix(y\_test,y\_pred)  
  
count=len(X\_test)  
crt\_pre=0  
for i in range(count):  
 if y\_test[i]==y\_pred[i]:  
 crt\_pre=crt\_pre+1;  
Accu=(crt\_pre/count) \* 100  
print("Model Accure:",Accu)  
  
# Visualising the Training set results  
from matplotlib.colors import ListedColormap  
X\_set, y\_set = X\_train, y\_train  
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  
 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))  
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  
 alpha = 0.75, cmap = ListedColormap(('red', 'green')))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i, j in enumerate(np.unique(y\_set)):  
 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  
 c = ListedColormap(('red', 'green'))(i), label = j)  
plt.title('KNN (Training set)')  
plt.xlabel('Age')  
plt.ylabel('Estimated Salary')  
plt.legend()  
plt.show()

# Visualising the Test set results  
from matplotlib.colors import ListedColormap  
X\_set, y\_set = X\_test, y\_test  
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  
 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))  
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  
 alpha = 0.75, cmap = ListedColormap(('red', 'green')))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i, j in enumerate(np.unique(y\_set)):  
 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  
 c = ListedColormap(('red', 'green'))(i), label = j)  
plt.title('KNN (Test set)')  
plt.xlabel('Age')  
plt.ylabel('Estimated Salary')  
plt.legend()  
plt.show()

**Support Vector Machine (SVM)**

#importing lib  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
  
#importing data  
data=pd.read\_csv("E:\\MY\_GOAL\\Machine Learning\\Part 3 - Classification\\Section 14 - Logistic Regression\\Social\_Network\_Ads.csv")  
X=data.iloc[:,2 : 4].values  
y=data.iloc[:,-1].values  
  
#Splittting data  
from sklearn.model\_selection import train\_test\_split  
X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state=0)  
  
#Feature scaling  
from sklearn.preprocessing import StandardScaler  
sc\_X=StandardScaler()  
X\_train=sc\_X.fit\_transform(X\_train)  
X\_test=sc\_X.transform(X\_test)  
  
#Build the model  
from sklearn.svm import SVC  
classifier=SVC(kernel='linear',random\_state=0)  
classifier.fit(X\_train,y\_train)  
  
#Test the model  
y\_pred=classifier.predict(X\_test)  
  
#confusion matrix  
from sklearn.metrics import confusion\_matrix  
cm=confusion\_matrix(y\_test,y\_pred)  
  
count=len(X\_test)  
crt\_pre=0  
for i in range(count):  
 if y\_test[i]==y\_pred[i]:  
 crt\_pre=crt\_pre+1;  
Accu=(crt\_pre/count) \* 100  
print("Model Accure:",Accu)  
  
# Visualising the Training set results  
from matplotlib.colors import ListedColormap  
X\_set, y\_set = X\_train, y\_train  
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  
 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))  
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  
 alpha = 0.75, cmap = ListedColormap(('red', 'green')))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i, j in enumerate(np.unique(y\_set)):  
 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  
 c = ListedColormap(('red', 'green'))(i), label = j)  
plt.title('SVM(Training set)')  
plt.xlabel('Age')  
plt.ylabel('Estimated Salary')  
plt.legend()  
plt.show()  
  
# Visualising the Test set results  
from matplotlib.colors import ListedColormap  
X\_set, y\_set = X\_test, y\_test  
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  
 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))  
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  
 alpha = 0.75, cmap = ListedColormap(('red', 'green')))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i, j in enumerate(np.unique(y\_set)):  
 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  
 c = ListedColormap(('red', 'green'))(i), label = j)  
plt.title('SVM (Test set)')  
plt.xlabel('Age')  
plt.ylabel('Estimated Salary')  
plt.legend()

**Kernel SVM**

#importing lib  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
  
#importing data  
data=pd.read\_csv("E:\\MY\_GOAL\\Machine Learning\\Part 3 - Classification\\Section 14 - Logistic Regression\\Social\_Network\_Ads.csv")  
X=data.iloc[:,2 : 4].values  
y=data.iloc[:,-1].values  
  
#Splittting data  
from sklearn.model\_selection import train\_test\_split  
X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state=0)  
  
#Feature scaling  
from sklearn.preprocessing import StandardScaler  
sc\_X=StandardScaler()  
X\_train=sc\_X.fit\_transform(X\_train)  
X\_test=sc\_X.transform(X\_test)  
  
#Build the model  
from sklearn.svm import SVC  
classifier=SVC(kernel='rbf',random\_state=0)  
classifier.fit(X\_train,y\_train)  
  
#Test the model  
y\_pred=classifier.predict(X\_test)  
  
#confusion matrix  
from sklearn.metrics import confusion\_matrix  
cm=confusion\_matrix(y\_test,y\_pred)  
  
count=len(X\_test)  
crt\_pre=0  
for i in range(count):  
 if y\_test[i]==y\_pred[i]:  
 crt\_pre=crt\_pre+1;  
Accu=(crt\_pre/count) \* 100  
print("Model Accure:",Accu)  
  
# Visualising the Training set results  
from matplotlib.colors import ListedColormap  
X\_set, y\_set = X\_train, y\_train  
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  
 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))  
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  
 alpha = 0.75, cmap = ListedColormap(('red', 'green')))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i, j in enumerate(np.unique(y\_set)):  
 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  
 c = ListedColormap(('red', 'green'))(i), label = j)  
plt.title('SVM(Training set)')  
plt.xlabel('Age')  
plt.ylabel('Estimated Salary')  
plt.legend()  
plt.show()  
  
# Visualising the Test set results  
from matplotlib.colors import ListedColormap  
X\_set, y\_set = X\_test, y\_test  
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  
 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))  
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  
 alpha = 0.75, cmap = ListedColormap(('red', 'green')))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i, j in enumerate(np.unique(y\_set)):  
 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  
 c = ListedColormap(('red', 'green'))(i), label = j)  
plt.title('SVM (Test set)')  
plt.xlabel('Age')  
plt.ylabel('Estimated Salary')  
plt.legend()

**Decision Tree Classification**

#importing lib  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
  
#importing data  
data=pd.read\_csv("E:\\MY\_GOAL\\Machine Learning\\Part 3 - Classification\\Section 14 - Logistic Regression\\Social\_Network\_Ads.csv")  
X=data.iloc[:,2 : 4].values  
y=data.iloc[:,-1].values  
  
#Splittting data  
from sklearn.model\_selection import train\_test\_split  
X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state=0)  
  
#Feature scaling  
from sklearn.preprocessing import StandardScaler  
sc\_X=StandardScaler()  
X\_train=sc\_X.fit\_transform(X\_train)  
X\_test=sc\_X.transform(X\_test)  
  
#Build the model  
from sklearn.tree import DecisionTreeClassifier  
classifier=DecisionTreeClassifier(criterion='entropy',random\_state=0)  
classifier.fit(X\_train,y\_train)  
  
#Test the model  
y\_pred=classifier.predict(X\_test)  
  
#confusion matrix  
from sklearn.metrics import confusion\_matrix  
cm=confusion\_matrix(y\_test,y\_pred)  
  
count=len(X\_test)  
crt\_pre=0  
for i in range(count):  
 if y\_test[i]==y\_pred[i]:  
 crt\_pre=crt\_pre+1;  
Accu=(crt\_pre/count) \* 100  
print("Model Accure:",Accu)  
  
# Visualising the Training set results  
from matplotlib.colors import ListedColormap  
X\_set, y\_set = X\_train, y\_train  
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  
 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))  
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  
 alpha = 0.75, cmap = ListedColormap(('red', 'green')))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i, j in enumerate(np.unique(y\_set)):  
 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  
 c = ListedColormap(('red', 'green'))(i), label = j)  
plt.title('Decision Tree Classification(Training set)')  
plt.xlabel('Age')  
plt.ylabel('Estimated Salary')  
plt.legend()  
plt.show()  
  
# Visualising the Test set results  
from matplotlib.colors import ListedColormap  
X\_set, y\_set = X\_test, y\_test  
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  
 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))  
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  
 alpha = 0.75, cmap = ListedColormap(('red', 'green')))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i, j in enumerate(np.unique(y\_set)):  
 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  
 c = ListedColormap(('red', 'green'))(i), label = j)  
plt.title('Decision Tree Classification(Test set)')  
plt.xlabel('Age')  
plt.ylabel('Estimated Salary')  
plt.legend()

**Random Forest Classification**

#importing packages  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
  
#importing datasets  
data=pd.read\_csv("E:\\MY\_GOAL\\Machine Learning\\Part 3 - Classification\\Section 14 - Logistic Regression\\Social\_Network\_Ads.csv")  
X=data.iloc[:,2 : 4].values  
y=data.iloc[:,-1].values  
  
#splitting data into train and test sets  
from sklearn.model\_selection import train\_test\_split  
X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state=0)  
  
#Feature scaling  
from sklearn.preprocessing import StandardScaler  
sc\_X=StandardScaler()  
X\_train=sc\_X.fit\_transform(X\_train)  
X\_test=sc\_X.transform(X\_test)  
  
#Build the model  
from sklearn.ensemble import RandomForestClassifier  
classifier=RandomForestClassifier(n\_estimators=10,  
 criterion='entropy',  
 random\_state=0)  
  
#test the model  
y\_pred=classifier.predict(X\_test)  
  
  
#model\_performance  
from sklearn.metrics import confusion\_matrix  
cm=confusion\_matrix(y\_test,y\_pred)  
  
count=len(X\_test)  
crt\_pre=0  
for i in range(count):  
 if y\_test[i]==y\_pred[i]:  
 crt\_pre=crt\_pre+1;  
Accu=(crt\_pre/count) \* 100  
print("Model Accure:",Accu)  
  
# Visualising the Training set results  
from matplotlib.colors import ListedColormap  
X\_set, y\_set = X\_train, y\_train  
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  
 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))  
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  
 alpha = 0.75, cmap = ListedColormap(('red', 'green')))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i, j in enumerate(np.unique(y\_set)):  
 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  
 c = ListedColormap(('red', 'green'))(i), label = j)  
plt.title('Random\_Forest\_Classification (Training set)')  
plt.xlabel('Age')  
plt.ylabel('Estimated Salary')  
plt.legend()  
plt.show()  
  
# Visualising the Test set results  
from matplotlib.colors import ListedColormap  
X\_set, y\_set = X\_test, y\_test  
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  
 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))  
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  
 alpha = 0.75, cmap = ListedColormap(('red', 'green')))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i, j in enumerate(np.unique(y\_set)):  
 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  
 c = ListedColormap(('red', 'green'))(i), label = j)  
plt.title('Random\_Forest\_Classification (Test set)')  
plt.xlabel('Age')  
plt.ylabel('Estimated Salary')  
plt.legend()  
plt.show()

**Naive Bayes**

#importing lib  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
  
#importing data  
data=pd.read\_csv("E:\\MY\_GOAL\\Machine Learning\\Part 3 - Classification\\Section 14 - Logistic Regression\\Social\_Network\_Ads.csv")  
X=data.iloc[:,2 : 4].values  
y=data.iloc[:,-1].values  
  
#Splittting data  
from sklearn.model\_selection import train\_test\_split  
X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state=0)  
  
#Feature scaling  
from sklearn.preprocessing import StandardScaler  
sc\_X=StandardScaler()  
X\_train=sc\_X.fit\_transform(X\_train)  
X\_test=sc\_X.transform(X\_test)  
  
#Build the model  
from sklearn.naive\_bayes import GaussianNB  
classifier=GaussianNB()  
classifier.fit(X\_train,y\_train)  
#Test the model  
y\_pred=classifier.predict(X\_test)  
  
#confusion matrix  
from sklearn.metrics import confusion\_matrix  
cm=confusion\_matrix(y\_test,y\_pred)  
  
count=len(X\_test)  
crt\_pre=0  
for i in range(count):  
 if y\_test[i]==y\_pred[i]:  
 crt\_pre=crt\_pre+1;  
Accu=(crt\_pre/count) \* 100  
print("Model Accure:",Accu)  
  
# Visualising the Training set results  
from matplotlib.colors import ListedColormap  
X\_set, y\_set = X\_train, y\_train  
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  
 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))  
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  
 alpha = 0.75, cmap = ListedColormap(('red', 'green')))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i, j in enumerate(np.unique(y\_set)):  
 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  
 c = ListedColormap(('red', 'green'))(i), label = j)  
plt.title('Naive Bayes(Training set)')  
plt.xlabel('Age')  
plt.ylabel('Estimated Salary')  
plt.legend()  
plt.show()  
  
# Visualising the Test set results  
from matplotlib.colors import ListedColormap  
X\_set, y\_set = X\_test, y\_test  
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  
 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))  
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  
 alpha = 0.75, cmap = ListedColormap(('red', 'green')))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i, j in enumerate(np.unique(y\_set)):  
 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  
 c = ListedColormap(('red', 'green'))(i), label = j)  
plt.title('Naive Bayes(Test set)')  
plt.xlabel('Age')  
plt.ylabel('Estimated Salary')  
plt.legend()

**CLUSTERING**

**K-Means Clustering**

#import lib  
import pandas as pd  
import matplotlib.pyplot as plt  
import numpy as ny  
  
#import dataset  
dataset =pd.read\_csv("E:\\MY\_GOAL\\Machine Learning\\Part 4 - Clustering\\Section 24 - K-Means Clustering\\Mall\_Customers.csv")  
X=dataset.iloc[:,[3,4]].values  
  
#using elbow method  
from sklearn.cluster import KMeans  
wcss=[]  
for i in range(1,11):  
 kmeans=KMeans(n\_clusters=i,init='k-means++',max\_iter=300,n\_init=10,random\_state=0)  
 kmeans.fit(X)  
 wcss.append(kmeans.inertia\_)  
plt.plot(range(1,11),wcss)  
plt.title("Elbow Method")  
plt.xlabel("No Of Clusters")  
plt.ylabel("WCSS")  
plt.show()   
  
  
#Build the kmeans cluster  
kmeans=KMeans(n\_clusters=5,init='k-means++',max\_iter=300,n\_init=10,random\_state=0)  
y\_kmeans=kmeans.fit\_predict(X)  
  
#visualising the clusters  
plt.scatter(X[y\_kmeans ==0,0],X[y\_kmeans==0,1],s=100,c='red',label='C1')  
plt.scatter(X[y\_kmeans ==1,0],X[y\_kmeans==1,1],s=100,c='green',label='C2')  
plt.scatter(X[y\_kmeans ==2,0],X[y\_kmeans==2,1],s=100,c='blue',label='C3')  
plt.scatter(X[y\_kmeans ==3,0],X[y\_kmeans==3,1],s=100,c='orange',label='C4')  
plt.scatter(X[y\_kmeans ==4,0],X[y\_kmeans==4,1],s=100,c='yellow',label='C5')  
plt.scatter(kmeans.cluster\_centers\_[:,0],kmeans.cluster\_centers\_[:,1],s=300,c='black',label='Centroid')  
plt.xlabel("Annual Income")  
plt.ylabel("Age")  
plt.title("Cluster Of Customers")  
plt.show()

**Hierarchical\_Clustering**

#importing packages  
import pandas as pd  
import matplotlib.pyplot as plt  
  
#importing dataset  
data=pd.read\_csv("E:/MY\_GOAL/Machine Learning/Part 4 - Clustering/Section 25 - Hierarchical Clustering/Mall\_Customers.csv")  
X=data.iloc[:,[3,4]].values  
  
  
#using dendrograms to find number of clusters  
import scipy.cluster.hierarchy as sch  
dendro=sch.dendrogram(sch.linkage(X,method='ward'))  
plt.title("DendroGrams")  
plt.xlabel("Customers")  
plt.ylabel("Euclidean Dist")  
plt.show()  
  
#BUild HC model  
from sklearn.cluster import AgglomerativeClustering  
A\_hc=AgglomerativeClustering(n\_clusters=5,affinity="euclidean",linkage="ward")  
y\_hc=A\_hc.fit\_predict(X)  
  
#visualising the clusters  
plt.scatter(X[y\_hc ==0,0],X[y\_hc==0,1],s=100,c='red',label='C1')  
plt.scatter(X[y\_hc ==1,0],X[y\_hc==1,1],s=100,c='green',label='C2')  
plt.scatter(X[y\_hc ==2,0],X[y\_hc==2,1],s=100,c='blue',label='C3')  
plt.scatter(X[y\_hc ==3,0],X[y\_hc==3,1],s=100,c='orange',label='C4')  
plt.scatter(X[y\_hc ==4,0],X[y\_hc==4,1],s=100,c='yellow',label='C5')  
plt.xlabel("Annual Income")  
plt.ylabel("Age")  
plt.title("Cluster Of Customers")  
plt.show()

**APRIORI-Association Rule Mapping**

#import lib  
import pandas as pd  
import numpy as ny  
import matplotlib.pyplot as plt  
  
#importing dataset  
dataset=pd.read\_csv("E:\\MY\_GOAL\\Machine Learning\\Part 5 - Association Rule Learning\\Section 28 - Apriori\\Market\_Basket\_Optimisation.csv",header=None)  
trans=[]  
for i in range(0,7501):  
 trans.append([str(dataset.values[i,j])for j in range(0,20)])   
#train the model  
from apyori import apriori   
rules=apriori(trans,min\_support=0.005,min\_confidence=0.2,min\_lift=3,min\_length=2)  
  
#visualize  
res=list(rules)

**Upper Confidence Bound(UCB)**

# Upper Confidence Bound  
  
# Importing the libraries  
import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
  
# Importing the dataset  
dataset = pd.read\_csv('Ads\_CTR\_Optimisation.csv')  
  
# Implementing UCB  
import math  
N = 10000  
d = 10  
ads\_selected = []  
numbers\_of\_selections = [0] \* d  
sums\_of\_rewards = [0] \* d  
total\_reward = 0  
for n in range(0, N):  
 ad = 0  
 max\_upper\_bound = 0  
 for i in range(0, d):  
 if (numbers\_of\_selections[i] > 0):  
 #print("if:",numbers\_of\_selections[i],ad)  
 average\_reward = sums\_of\_rewards[i] / numbers\_of\_selections[i]  
 delta\_i = math.sqrt(3/2 \* math.log(n + 1) / numbers\_of\_selections[i])  
 upper\_bound = average\_reward + delta\_i  
 else:  
 print("else:",numbers\_of\_selections[i])  
 upper\_bound = 1e400  
 if upper\_bound > max\_upper\_bound:  
 max\_upper\_bound = upper\_bound  
 ad = i  
 ads\_selected.append(ad)  
 numbers\_of\_selections[ad] = numbers\_of\_selections[ad] + 1  
 reward = dataset.values[n, ad]  
 sums\_of\_rewards[ad] = sums\_of\_rewards[ad] + reward  
 total\_reward = total\_reward + reward  
  
# Visualising the results  
plt.hist(ads\_selected)  
plt.title('Histogram of ads selections')  
plt.xlabel('Ads')  
plt.ylabel('Number of times each ad was selected')  
plt.show()

**Thompson Sampling**

# Importing the libraries  
import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
  
# Importing the dataset  
dataset = pd.read\_csv('Ads\_CTR\_Optimisation.csv')  
  
# Implementing Thompson Sampling  
import random  
N = 10000  
d = 10  
ads\_selected = []  
numbers\_of\_rewards\_1 = [0] \* d  
numbers\_of\_rewards\_0 = [0] \* d  
total\_reward = 0  
for n in range(0, N):  
 ad = 0  
 max\_random = 0  
 for i in range(0, d):  
 random\_beta = random.betavariate(numbers\_of\_rewards\_1[i] + 1, numbers\_of\_rewards\_0[i] + 1)  
 if random\_beta > max\_random:  
 max\_random = random\_beta  
 ad = i  
 ads\_selected.append(ad)  
 reward = dataset.values[n, ad]  
 if reward == 1:  
 numbers\_of\_rewards\_1[ad] = numbers\_of\_rewards\_1[ad] + 1  
 else:  
 numbers\_of\_rewards\_0[ad] = numbers\_of\_rewards\_0[ad] + 1  
 total\_reward = total\_reward + reward  
  
# Visualising the results - Histogram  
plt.hist(ads\_selected)  
plt.title('Histogram of ads selections')  
plt.xlabel('Ads')  
plt.ylabel('Number of times each ad was selected')  
plt.show()

**Natural language processing (NLP)**

#importing lib  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import re  
import nltk  
from sklearn.feature\_extraction.text import CountVectorizer  
#nltk.download("stopwords")  
from nltk.corpus import stopwords  
from nltk.stem import PorterStemmer  
from sklearn.metrics import confusion\_matrix  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.model\_selection import train\_test\_split  
  
#importing dataset  
dataset=pd.read\_csv('Restaurant\_Reviews.tsv',delimiter="\t",quoting=3)  
  
#cleaning  
corpus=[]  
ps=PorterStemmer()  
for i in range (1000):  
 review = re.sub("[^a-zA-Z]", " ", dataset["Review"][i])  
 review=review.lower()  
 review=review.split()  
 review= [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]  
 review=" ".join(review)  
 corpus.append(review)  
  
#create bag of words  
cv=CountVectorizer()  
X=cv.fit\_transform(corpus).toarray()  
y = dataset.iloc[:, 1].values  
  
# Splitting the dataset into the Training set and Test set  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20, random\_state = 0)  
  
# Fitting Naive Bayes to the Training set  
classifier = GaussianNB()  
classifier.fit(X\_train, y\_train)  
  
# Predicting the Test set results  
y\_pred = classifier.predict(X\_test)  
  
# Making the Confusion Matrix  
cm = confusion\_matrix(y\_test, y\_pred)  
print(cm)

**Principal Component Analysis**

# Importing the libraries  
import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
  
# Importing the dataset  
dataset = pd.read\_csv('Wine.csv')  
X = dataset.iloc[:, 0:13].values  
y = dataset.iloc[:, 13].values  
  
# Splitting the dataset into the Training set and Test set  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0)  
  
# Feature Scaling  
from sklearn.preprocessing import StandardScaler  
sc = StandardScaler()  
X\_train = sc.fit\_transform(X\_train)  
X\_test = sc.transform(X\_test)  
  
# Applying PCA  
from sklearn.decomposition import PCA  
pca = PCA(n\_components = 2)  
X\_train = pca.fit\_transform(X\_train)  
X\_test = pca.transform(X\_test)  
explained\_variance = pca.explained\_variance\_ratio\_  
  
# Fitting Logistic Regression to the Training set  
from sklearn.linear\_model import LogisticRegression  
classifier = LogisticRegression(random\_state = 0)  
classifier.fit(X\_train, y\_train)  
  
# Predicting the Test set results  
y\_pred = classifier.predict(X\_test)  
  
# Making the Confusion Matrix  
from sklearn.metrics import confusion\_matrix  
cm = confusion\_matrix(y\_test, y\_pred)  
  
# Visualising the Training set results  
from matplotlib.colors import ListedColormap  
X\_set, y\_set = X\_train, y\_train  
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  
 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))  
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  
 alpha = 0.75, cmap = ListedColormap(('red', 'green', 'blue')))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i, j in enumerate(np.unique(y\_set)):  
 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  
 c = ListedColormap(('red', 'green', 'blue'))(i), label = j)  
plt.title('Logistic Regression (Training set)')  
plt.xlabel('PC1')  
plt.ylabel('PC2')  
plt.legend()  
plt.show()  
  
# Visualising the Test set results  
from matplotlib.colors import ListedColormap  
X\_set, y\_set = X\_test, y\_test  
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  
 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))  
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  
 alpha = 0.75, cmap = ListedColormap(('red', 'green', 'blue')))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i, j in enumerate(np.unique(y\_set)):  
 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  
 c = ListedColormap(('red', 'green', 'blue'))(i), label = j)  
plt.title('Logistic Regression (Test set)')  
plt.xlabel('PC1')  
plt.ylabel('PC2')  
plt.legend()  
plt.show()

**LinearDiscriminantAnalysis**

# LDA  
  
# Importing the libraries  
import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
  
# Importing the dataset  
dataset = pd.read\_csv('Wine.csv')  
X = dataset.iloc[:, 0:13].values  
y = dataset.iloc[:, 13].values  
  
# Splitting the dataset into the Training set and Test set  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0)  
  
# Feature Scaling  
from sklearn.preprocessing import StandardScaler  
sc = StandardScaler()  
X\_train = sc.fit\_transform(X\_train)  
X\_test = sc.transform(X\_test)  
  
# Applying LDA  
from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis as LDA  
lda = LDA(n\_components = 2)  
X\_train = lda.fit\_transform(X\_train, y\_train)  
X\_test = lda.transform(X\_test)  
  
# Fitting Logistic Regression to the Training set  
from sklearn.linear\_model import LogisticRegression  
classifier = LogisticRegression(random\_state = 0)  
classifier.fit(X\_train, y\_train)  
  
# Predicting the Test set results  
y\_pred = classifier.predict(X\_test)  
  
# Making the Confusion Matrix  
from sklearn.metrics import confusion\_matrix  
cm = confusion\_matrix(y\_test, y\_pred)  
  
# Visualising the Training set results  
from matplotlib.colors import ListedColormap  
X\_set, y\_set = X\_train, y\_train  
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  
 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))  
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  
 alpha = 0.75, cmap = ListedColormap(('red', 'green', 'blue')))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i, j in enumerate(np.unique(y\_set)):  
 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  
 c = ListedColormap(('red', 'green', 'blue'))(i), label = j)  
plt.title('Logistic Regression (Training set)')  
plt.xlabel('LD1')  
plt.ylabel('LD2')  
plt.legend()  
plt.show()  
  
# Visualising the Test set results  
from matplotlib.colors import ListedColormap  
X\_set, y\_set = X\_test, y\_test  
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  
 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))  
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  
 alpha = 0.75, cmap = ListedColormap(('red', 'green', 'blue')))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i, j in enumerate(np.unique(y\_set)):  
 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  
 c = ListedColormap(('red', 'green', 'blue'))(i), label = j)  
plt.title('Logistic Regression (Test set)')  
plt.xlabel('LD1')  
plt.ylabel('LD2')  
plt.legend()  
plt.show()

**Kernel PCA**

# Kernel PCA  
  
# Importing the libraries  
import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
  
# Importing the dataset  
dataset = pd.read\_csv('Social\_Network\_Ads.csv')  
X = dataset.iloc[:, [2, 3]].values  
y = dataset.iloc[:, 4].values  
  
# Splitting the dataset into the Training set and Test set  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)  
  
# Feature Scaling  
from sklearn.preprocessing import StandardScaler  
sc = StandardScaler()  
X\_train = sc.fit\_transform(X\_train)  
X\_test = sc.transform(X\_test)  
  
# Applying Kernel PCA  
from sklearn.decomposition import KernelPCA  
kpca = KernelPCA(n\_components = 2, kernel = 'rbf')  
X\_train = kpca.fit\_transform(X\_train)  
X\_test = kpca.transform(X\_test)  
  
# Fitting Logistic Regression to the Training set  
from sklearn.linear\_model import LogisticRegression  
classifier = LogisticRegression(random\_state = 0)  
classifier.fit(X\_train, y\_train)  
  
# Predicting the Test set results  
y\_pred = classifier.predict(X\_test)  
  
# Making the Confusion Matrix  
from sklearn.metrics import confusion\_matrix  
cm = confusion\_matrix(y\_test, y\_pred)  
  
# Visualising the Training set results  
from matplotlib.colors import ListedColormap  
X\_set, y\_set = X\_train, y\_train  
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  
 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))  
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  
 alpha = 0.75, cmap = ListedColormap(('red', 'green')))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i, j in enumerate(np.unique(y\_set)):  
 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  
 c = ListedColormap(('red', 'green'))(i), label = j)  
plt.title('Logistic Regression (Training set)')  
plt.xlabel('Age')  
plt.ylabel('Estimated Salary')  
plt.legend()  
plt.show()  
  
# Visualising the Test set results  
from matplotlib.colors import ListedColormap  
X\_set, y\_set = X\_test, y\_test  
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  
 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))  
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  
 alpha = 0.75, cmap = ListedColormap(('red', 'green')))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i, j in enumerate(np.unique(y\_set)):  
 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  
 c = ListedColormap(('red', 'green'))(i), label = j)  
plt.title('Logistic Regression (Test set)')  
plt.xlabel('Age')  
plt.ylabel('Estimated Salary')  
plt.legend()  
plt.show()

**K-Fold Cross Validation**

# Importing the libraries  
import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
  
# Importing the dataset  
dataset = pd.read\_csv('Social\_Network\_Ads.csv')  
X = dataset.iloc[:, [2, 3]].values  
y = dataset.iloc[:, 4].values  
  
# Splitting the dataset into the Training set and Test set  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)  
  
# Feature Scaling  
from sklearn.preprocessing import StandardScaler  
sc = StandardScaler()  
X\_train = sc.fit\_transform(X\_train)  
X\_test = sc.transform(X\_test)  
  
# Fitting Kernel SVM to the Training set  
from sklearn.svm import SVC  
classifier = SVC(kernel = 'rbf', random\_state = 0)  
classifier.fit(X\_train, y\_train)  
  
# Predicting the Test set results  
y\_pred = classifier.predict(X\_test)  
  
# Making the Confusion Matrix  
from sklearn.metrics import confusion\_matrix  
cm = confusion\_matrix(y\_test, y\_pred)  
  
# Applying k-Fold Cross Validation  
from sklearn.model\_selection import cross\_val\_score  
accuracies = cross\_val\_score(estimator = classifier, X = X\_train, y = y\_train, cv = 10)  
m=accuracies.mean()  
s=accuracies.std()  
print("Mean:",m)  
print("Std:",s)

**Grid Search**

# Importing the libraries  
import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
  
# Importing the dataset  
dataset = pd.read\_csv('Social\_Network\_Ads.csv')  
X = dataset.iloc[:, [2, 3]].values  
y = dataset.iloc[:, 4].values  
  
# Splitting the dataset into the Training set and Test set  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)  
  
# Feature Scaling  
from sklearn.preprocessing import StandardScaler  
sc = StandardScaler()  
X\_train = sc.fit\_transform(X\_train)  
X\_test = sc.transform(X\_test)  
  
# Fitting Kernel SVM to the Training set  
from sklearn.svm import SVC  
classifier = SVC(kernel = 'rbf', random\_state = 0)  
classifier.fit(X\_train, y\_train)  
  
# Predicting the Test set results  
y\_pred = classifier.predict(X\_test)  
  
# Making the Confusion Matrix  
from sklearn.metrics import confusion\_matrix  
cm = confusion\_matrix(y\_test, y\_pred)  
  
# Applying k-Fold Cross Validation  
from sklearn.model\_selection import cross\_val\_score  
accuracies = cross\_val\_score(estimator = classifier, X = X\_train, y = y\_train, cv = 10)  
accuracies.mean()  
accuracies.std()  
  
# Applying Grid Search to find the best model and the best parameters  
from sklearn.model\_selection import GridSearchCV  
parameters = [{'C': [1, 10, 100, 1000], 'kernel': ['linear']},  
 {'C': [1, 10, 100, 1000], 'kernel': ['rbf'], 'gamma': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]}]  
grid\_search = GridSearchCV(estimator = classifier,  
 param\_grid = parameters,  
 scoring = 'accuracy',  
 cv = 10,  
 n\_jobs = -1)  
grid\_search = grid\_search.fit(X\_train, y\_train)  
best\_accuracy = grid\_search.best\_score\_  
best\_parameters = grid\_search.best\_params\_  
print("ACC:",best\_accuracy)  
print("Parameters:",best\_parameters)

**XGBOOST**

# Importing the libraries  
import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
  
# Importing the dataset  
dataset = pd.read\_csv('Churn\_Modelling.csv')  
X = dataset.iloc[:, 3:13].values  
y = dataset.iloc[:, 13].values  
# Encoding categorical data  
from sklearn.preprocessing import LabelEncoder, OneHotEncoder  
labelencoder\_X\_1 = LabelEncoder()  
X[:, 1] = labelencoder\_X\_1.fit\_transform(X[:, 1])  
labelencoder\_X\_2 = LabelEncoder()  
X[:, 2] = labelencoder\_X\_2.fit\_transform(X[:, 2])  
onehotencoder = OneHotEncoder(categorical\_features = [1])  
X = onehotencoder.fit\_transform(X).toarray()  
X = X[:, 1:]  
# Splitting the dataset into the Training set and Test set  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0)  
# Fitting XGBoost to the Training set  
from xgboost import XGBClassifier  
classifier = XGBClassifier()  
classifier.fit(X\_train, y\_train)  
# Predicting the Test set results  
y\_pred = classifier.predict(X\_test)  
# Making the Confusion Matrix  
from sklearn.metrics import confusion\_matrix  
cm = confusion\_matrix(y\_test, y\_pred)  
# Applying k-Fold Cross Validation  
from sklearn.model\_selection import cross\_val\_score  
accuracies = cross\_val\_score(estimator = classifier, X = X\_train, y = y\_train, cv = 10)  
accuracies.mean()  
accuracies.std()