# MACHINE LEARNING WITH PYTHON

AchyuthReddy.k

M.TECH DataScience

| 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='bla
ck',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)

```python
# 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()
```