**DATASET**

#import necessary libraries

import pandas as pd

import numpy as np

import pickle

import matplotlib.pyplot as plt

%matplotlib inline

import seaborn as sns

import sklearn

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.model\_selection import RandomizedSearchCV

import imblearn

from imblearn.over\_sampling import SMOTE

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

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, f1\_score

#import dataset

data = pd.read\_csv(r"/content/Telecom\_customer churn.csv")

data

**Activity 2.1: Handling missing values**

data.info()

#checking for ‘null values’

data.Customer\_ID = pd.to\_numeric(data.Customer\_ID, errors= 'coerce')

data.isnull().any()

data["Customer\_ID"].fillna(data["Customer\_ID"].median() , inplace =False)

data.isnull().sum()

**Activity 2.2: Handling Categorical Values**

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

data["gender"] = le.fit\_transform(data["gender"])

data["Partner"] = le.fit\_transform(data["Partner"])

data["Dependents"] = le.fit\_transform(data["Dependents"])

data["Phoneservice"] = le.fit\_transform(data["Phoneservice"])

data["MultipleLines"] = le.fit\_transform(data["MultipleLines"])

data["InternetService"] = le.fit\_transform(data["InternetService"])

data["OnlineBackup"] = le.fit\_transform(data["OnlineBackup"])

data["DeviceProtection"] = le.fit\_transform(data["DeviceProtection"])

data["TechSupport"] = le.fit\_transform(data["TechSupport"])

data["StreamingTV"] = le.fit\_transform(data["StreamingTV"])

data["StreamingMovies"] = le.fit\_transform(data["StreamingMovies"])

data["Contract"] = le.fit\_transform(data["Contract"])

data["PaperlessBilling"] = le.fit\_transform(data["PaperlessBilling"])

data["PaymentMethod"] = le.fit\_transform(data["PaymentMethod"])

data["chrun"] = le.fit\_transform(data["chrun"])

data.head()

x= data.iloc[:,0:19].values

y= data.iloc[:,19:20].values

from sklearn.preprocessing import OneHotEncoder

one = OneHotEncoder()

a= one.fit\_transform(x[:,6:7]).toarray()

b= one.fit\_transform(x[:,7:8]).toarray()

c= one.fit\_transform(x[:,8:9]).toarray()

d= one.fit\_transform(x[:,9:10]).toarray()

e= one.fit\_transform(x[:,10:11]).toarray()

f= one.fit\_transform(x[:,11:12]).toarray()

g= one.fit\_transform(x[:,12:13]).toarray()

a= one.fit\_transform(x[:,13:14]).toarray()

a= one.fit\_transform(x[:,14:15]).toarray()

a= one.fit\_transform(x[:,16:17]).toarray()

x=np.delete(x,[6,7,8,9,10,11,12,13,14,16],axis=1)

x=np.concatenate((a,b,c,d,e,f,g,h,i,j,x),axis=1)

**Activity 2.3: Handling Imbalance Data**

from imblearn.over\_sampling import SMOTE

smt = SMOTE()

x\_resample, y\_resample = smt.fit\_resample(x,y)

x\_resample

**Milestone 3: Exploratory Data Analysis**

**Activity 1: Descriptive statistical**

data.describe()

**Activity 2.1: Univariate analysis**

p1t.figure(figsize=(12,5))

p1t.subplot(1,2,1)

sns.distplot(data["tenure"])

p1t.subplot(1,2,2)

sns.distplot(data["MonthCharges"])

p1t.figure(figsize=(12,5))

p1t.subplot(1,2,1)

sns.distplot(data["gender"])

p1t.subplot(1,2,2)

sns.distplot(data["Dependents"])

**Activity 2.2: Bivariate analysis**

sns.barplot(x="Churn", y="MonthlyCharges", data=data)

**Activity 2.3: Multivariate analysis**

sns.headmap(data.corr(), annot=Irue)

sns.pairplot(data data, markers=["^","v".],palette="inferno")

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

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x\_resample,y\_resample,test\_size = 0.2, random\_state =0)

from sklearn.preprocessing import Standardscaler

sc = Standardscaler()

x\_train = sc.fit\_transform(x\_train)

x\_test = sc.fit\_transform(x\_test)

x.test.shape

**Milestone 4: Model Building**

**Activity 1.2: Logistic Regression Model**

#importing and building the Decision tree model

def logreg(x\_train,x\_test,y\_train,y\_test):

  lr = LogisticRegression(random\_state=0)

  lr.fit(x\_train,y\_train)

  y\_lr\_tr = lr.predict(x\_train)

  print(accuracy\_score(y\_lr\_tr,y\_train))

  yPred\_lr = lr.predict(x\_test)

  print(accuracy\_score(yPred\_lr,y\_test))

  print("\*\*\*Logistic Regression\*\*\*")

  print("Confusion\_Matrix")

  print(confusion\_matrix(y\_test,yPred\_lr))

  print("Classification Report")

  print(classification\_report(y\_test,yPred\_lr))

  #printing the train accuracy and test accuracy respectively

logreg(x\_train,x\_test,y\_train,y\_test)

**Activity 1.2: Decision tree model**

#importing and building the Decision tree model

def decisionTree(x\_train,x\_test,y\_train,y\_test):

  dtc = DecisionTreeClassifier(criterion= "entropy",random\_state=0)

  dtc.fit(x\_train,y\_train)

  y\_dt\_tr = dtc.predict(x\_train)

  print(accuracy\_score(y\_dt\_tr,y\_train))

  yPred\_dt = dtc.predict(x\_test)

  print(accuracy\_score(yPred\_dt,y\_test))

  print("\*\*\*Decision Tree\*\*\*”)

  print("Confusion\_Matrix")

  print(confusion\_matrix(y\_test,yPred\_dt))

  print("Classification Report")

  print(classification\_report(y\_test,yPred\_dt))

#printing the train accuracy and test accuracy respectively

  decisionTree(x\_train,x\_test,y\_train,y\_test)

**Activity 1.3: Random forest model**

#importing and building the Decision tree model

def RandomForest(x\_train,x\_test,y\_train,y\_test):

  rf = RandomForestClassifier(criterion= "entropy",n\_estimators=10,random\_state=0)

  rf.fit(x\_train,y\_train)

  y\_rf\_tr = rf.predict(x\_train)

  print(accuracy\_score(y\_rf\_tr,y\_train))

  yPred\_rf = rf.predict(x\_test)

  print(accuracy\_score(yPred\_rf,y\_test))

  print("\*\*\*Random Forest\*\*\*")

  print("Confusion\_Matrix")

  print(confusion\_matrix(y\_test,yPred\_rf))

  print("Classification Report")

  print(classification\_report(y\_test,yPred\_rf))

#printing the train accuracy and test accuracy respectively

  RandomForest(x\_train,x\_test,y\_train,y\_test)

**Activity 1.3: KNN model**

#importing and building the Decision tree model

def KNN(x\_train,x\_test,y\_train,y\_test):

  knn = KNeighborsClassifier()

  knn.fit(x\_train,y\_train)

  y\_knn\_tr = knn.predict(x\_train)

  print(accuracy\_score(y\_rf\_knn,y\_train))

  yPred\_knn = knn.predict(x\_test)

  print(accuracy\_score(yPred\_knn,y\_test))

  print("\*\*\*KNN\*\*\*")

  print("Confusion\_Matrix")

  print(confusion\_matrix(y\_test,yPred\_knn))

  print("Classification Report")

  print(classification\_report(y\_test,yPred\_knn))

#printing the train accuracy and test accuracy respectively

 KNN(x\_train,x\_test,y\_train,y\_test)

**Activity 1.4: SVM model**

#importing and building the Decision tree model

def svm(x\_train,x\_test,y\_train,y\_test):

    svm = SVC(kernel = "linear")

    svm.fit(x\_train,y\_train)

    y\_svm\_tr = svm.predict(x\_train)

    print(accuracy\_score(y\_svm\_tr,y\_train))

    yPred\_svm = svm.predict(x\_test)

    print(accuracy\_score(yPred\_svm,y\_test))

    print("\*\*\*Support Vector Machine\*\*\*")

    print("Confusion\_Matrix")

    print(confusion\_matrix(y\_test,yPred\_svm))

    print("Classification Report")

    print(classification\_report(y\_test,yPred\_svm))

#printing the train accuracy and test accuracy respectively

svm(x\_train,x\_test,y\_train,y\_test)

**Activity 1.5: ANN model**

# Importing the keras libraries and packages

import keras

from keras.model import sequential

from keras.layers import Dense

# Initialising the ANN

  classifier = sequential()

# Adding the input layer and the first hidden layer

  classifire.add(Dense(units=30, activation='relu', input\_dim=40))

# Adding the second hidden layer

  classifire.add(Dense(units=30, activation='relu'))

# Adding the output layer

  classifire.add(Dense(units=1, activation='sigmoid'))

# Compiling the ANN

  classifire.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Fitting the ANN Training set

model\_history = classifier.fit(x\_train,y\_train, batch\_size=10, validation\_split=0.33, epochs=200)

print(accuracy\_score(ann\_pred,y\_test))

print("\*\*\*\*ANN Model\*\*\*")

print("Confusion\_Matrix")

print(confusion\_matrix(y\_test,ann\_pred))

print("Classification Report")

print(classification\_report(y\_test,ann\_pred))

**Activity 2: Testing the model**

#testing on random input values

lr = LogisticRegression(random\_state=0)

lr.fit(x\_train,y\_train)

print("Predicting on random input")

lr\_pred\_own = lr.predict(sc.transform([[0,0,1,1,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,1,0,0,1,1,0,0,456,1,0,3245,4567]]))

print("output is:",lr\_pred\_own)

#testing on random input values

dtc = DesicionTreeClassifier(criterion="entropy",random\_state=0)

dtc.fit(x\_train,y\_train)

print("Predicting on random input")

dtc\_pred\_own = dtc.predict(sc.transform([[0,0,1,1,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,1,0,0,1,1,0,0,456,1,0,3245,4567]]))

print("output is:" dtc\_pred\_own)

#testing on random input values

rf = Random orsetclassifier(criterion="entropy",n\_estimators=10,random\_state=0)

rf.fit(x\_train,y\_train)

print("Predicting on random input")

rf\_pred\_own = rf.predict(sc.transform([[0,0,1,1,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,1,0,0,1,1,0,0,456,1,0,3245,4567]]))

print("output is: ",rf\_pred\_own)

#testing on random input values

svc = SVC("kernel = linear")

svc.fit(x\_train,y\_train)

print("Predicting on random input")

svm\_pred\_own = svc.predict(sc.transform([[0,0,1,1,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,1,0,0,1,1,0,0,456,1,0,3245,4567]]))

print("output is: ",svm\_pred\_own)

#testing on random input values

knn = KNeighborsClassifier()

knn.fit(x\_train,y\_train)

print("Predicting on random input")

knn\_pred\_own = knn.predict(sc.transform([[0,0,1,1,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,1,0,0,1,1,0,0,456,1,0,3245,4567]]))

print("output is:",knn\_pred\_own)

#testing on random input values

print("Predicting on random input")

ann\_pred\_own = classifier.predict(sc.transform([[0,0,1,1,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,1,0,0,1,1,0,0,456,1,0,3245,4567]]))

print(ann\_pred\_own)

ann\_pred\_own = (ann\_pred\_own>0.5)

print("output is:" ann\_pred\_own)

**Milestone 5: Performance Testing & Hyperparameter Tuning**

**Activity 1.1: Compare the model**

def compareModel(X\_train,X\_test,y\_train,y\_test):

    logreg(x\_train,x\_test,y\_train,y\_test)

    print('-'\*100)

    decisionTree(X\_train,X\_test,y\_train,y\_test)

    print('-'\*100)

    RandomForest(X\_train,X\_test,y\_train,y\_test)

    print('-'\*100)

    svm(X\_train,X\_test,y\_train,y\_test)

    print('-'\*100)

    KNN(X\_train,X\_test,y\_train,y\_test)

    print("-"\*100)

compareModel(x\_train,x\_test,y\_train,y\_test)

print(accuracy\_score(ann\_pred,y\_test))

print(\*\*\*\*ANN Model\*\*\*\*)

print("Confusion\_Matrix")

print(confusion\_matrix(y\_test,ann\_pred))

print("Classification Report")

print(classification\_report(y\_test,ann\_pred))

**Activity 2: Comparing model accuracy before & after applying**

**hyperparameter tuning**

y\_rf = model.predict(x\_train)

print(accuracy\_score(y\_rf,y\_train))

yPred\_rfcv= model.predict(x\_test)

print(accuracy\_score(yPred\_rfcv,y\_test))

print("Random Forest after Hyperparameter Luning")

print("Confusion\_Matrix")

print(confusion\_matrix)(y\_test,yPred\_rfcv)

print("Clasification Report")

printf(classification\_report(y\_test,yPred\_rfcv))

print("Predicting on random input")

rfcv\_pred\_own = model.predict(sc.transform([[0,0,1,1,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,1,0,0,1,1,0,0,456,1,0,3245,4567]]))

print("output is:",rfcv\_pred\_own)

**Milestone 6: Model Deployment**

**Activity 1:Save the best model**

classifier.save("telcom\_churn.15")

**Activity 2.2: Build Python code:**

from flask import Flask, render\_template, request

import keras

from keras.models import load\_model

app = Flask(\_\_\_\_name\_\_\_)

model = load\_model("telcom\_churn.h5")

@app.route('/') # rendering the html template

def home():

    return render\_template('home.html')

@app.roude('/')

def helloworld():

    return render\_template("base.html")

@app.route('/assesment')

def prediction():

    return render\_template("index.html")

@app.route('/predict', methods = ['POST'])

def admin():

    a= request.form["gender"]

    if (a == 'f'):

        a=0

    if (a == 'm'):

        a=1

    b= request.form["srcitizen"]

    if (b == 'n'):

        b=0

    if (b == 'y'):

        b=1

    c= request.form["partner"]

    if (c == 'n'):

        c=0

    if (c == 'y'):

        c=1

    d= request.form["dependents"]

    if (d == 'n'):

        d=0

    if (d == 'y'):

        d=1

    e= request.form["Lenure"]

    f= request.form["phservices"]

    if (f == 'n'):

        f=0

    if (f == 'y'):

        f=1

    g= request.form["multi"]

    if (g == 'n'):

        g1,g2,g3=1,0,0

    if (g == 'nps'):

        g1,g2,g3=0,1,0

    if (g == 'y'):

        g1,g2,g3=0,0,1

    h= request.form["is"]

    if (h == 'dsl'):

        h1,h2,h3=1,0,0

    if (h == 'fo'):

        h1,h2,h3=0,1,0

    if (h == 'n'):

        h1,h2,h3=0,0,1

    i= request.form["os"]

    if (i == 'n'):

        i1,i2,i3=1,0,0

    if (i == 'nis'):

        i1,i2,i3=0,1,0

    if (i == 'y'):

        i1,i2,i3=0,0,1

    j= request.form["ob"]

    if (j == 'n'):

        j1,j2,j3=1,0,0

    if (j == 'nis'):

        j1,j2,j3=0,1,0

    if (j =j 'y'):

        j1,j2,j3=0,0,1

    k= request.form["dp"]

    if (k == 'n'):

        k1,k2,k3=1,0,0

    if (k == 'nis'):

        k1,k2,k3=0,1,0

    if (k == 'y'):

        k1,k2,k3=0,0,1

    l= request.form["ts"]

    if (l == 'n'):

        l1,l2,l3=1,0,0

    if (l == 'nis'):

        l1,l2,l3=0,1,0

    if (l == 'y'):

        l1,l2,l3=0,0,1

    g= request.form["stv"]

    if (m == 'n'):

        m1,m2,m3=1,0,0

    if (m == 'nis'):

        m1,m2,m3=0,1,0

    if (m == 'y'):

        m1,m2,m3=0,0,1

    n= request.form["smv"]

    if (n == 'n'):

        n1,n2,n3=1,0,0

    if (n == 'nis'):

        n1,n2,n3=0,1,0

    if (n == 'y'):

        n1,n2,n3=0,0,1

    o= request.form["contract"]

    if (o == 'mtm'):

        o1,o2,o3=1,0,0

    if (o == 'oyr'):

        o1,o2,o3=0,1,0

    if (o == 'tyrs'):

        o1,o2,o3=0,0,1

    p= request.form["pmt"]

    if (p == 'ec'):

        p1,p2,p3=1,0,0,0

    if (p == 'mail'):

        p1,p2,p3,p4=0,1,0,0

    if (p == 'bt'):

        p1,p2,p3,p4=0,0,1,0

    if (p == 'cc'):

        p1,p2,p3,p4=0,0,0,1

    q= request.form["plb"]

    if (q == 'n'):

        q=0

    if (q == 'y'):

        q=1

    r= request.form["mcharges"]

    s =request.form["tcharges"]

t=[[int(g1),int(g2),int(g3),int(h1),int(h2),int(h3),int(i1),int(i2),int(i3),int(j1),int(j2),int(j3),int(k1),int(k2),int(k3),int(l1),int(l2),int(l3),int(m1),int(m2),int(m3),int(n1),int(n2),int(n3),int(o1),int(o2),int(o3),int(p1),int(p2),int(p3),int(p4)]]

print(t)

x = model.predict(t)

print(x[0])

if (x[[0]]) <=0.5):

    y ="NO"

    return render\_template("predno.html", z = y)

if (x[[0]]) >= 0.5):

    y ="Yes"

    return render\_template("predyes.html", z = y)