```
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()
h= one.fit_transform(x[:,13:14]).toarray()
i= one.fit_transform(x[:,14:15]).toarray()
j= 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)
```

```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
sns.distplot(data["tenure"])
plt.subplot(1,2,2)
sns.distplot(data["MonthlyCharges"])
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.fit_transform(x_test)
```

```
x_train.shape
```

```
(8239, 10)
```

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

```
if (l == 'y'):
    l1,l2,l3=0,0,1
m= 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,p4=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'):
```

```
#importing and building the KNN 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_knn_tr,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)
```

```
#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,
print("output is: ",lr_pred_own)
```

```
Predicting on random input
output is:  [0]
```

```
#testing on random input values
dtc = DecisionTreeClassifier(criterion="entropy",random_state=0)
dtc.fit(x_train,y_train)
```

```
#importing and building the random forest model
def svm(x_tarin,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)
```

```python
#importing and building the random forest model
def RandomForest(x_tarin,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))
```

```python
#printing the train accuracy and test accuracy respectively
RandomForest(x_train,x_test,y_train,y_test)
```

```
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 == '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'):
```

```python
@app.route('/')
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["tenure"]
    f= request.form["phservices"]
    if (f == 'n'):
        f=0
    if (f == 'y'):
        f=1
```

```
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)
```

```
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
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)
```

```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
sns.countplot(data["gender"])
plt.subplot(1,2,2)
sns.countplot(data["Dependents"])
```

```
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["OnlineSecurity"] = le.fit_transform(data["OnlineSecurity"])
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["Churn"] = le.fit_transform(data["Churn"])
```

```
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))
```

```
    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)
```

```
[ ] # Importing the Keras libraries and packages
    import keras
    from keras.models import Sequential
    from keras.layers import Dense
```

```
[ ] # Initialising the ANN
    classifier = Sequential()
```

```
[ ] # Adding the input layer and the first hidden layer
    classifier.add(Dense(units=30, activation='relu', input_dim=40))
```

```
[ ] # Adding the second hidden layer
    classifier.add(Dense(units=30, activation='relu'))
```

```
[ ] # Adding the output layer
    classifier.add(Dense(units=1, activation='sigmoid'))
```

```
[ ] # Compiling the ANN
    classifier.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
#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)
```

```
from imblearn.over_sampling import SMOTE


smt = SMOTE()

x_resample, y_resample = smt.fit_resample(x,y)
```

```
#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
```

```
x= data.iloc[:,0:19].values
y= data.iloc[:,19:20].values
```