Service Cancellation Predictor

Team info

ID	**Name**
20201700185	**بسمة احمد سيد حنفي**
20201700861	**منة الله محمد عبد البديع احمد**
20201700928	**نورالهدى أشرف عبدالحليم عبدالوهاب**
20201700950	**هاجر خالد عبد المنعم على مصطفى**
20201700927	**نور الهدى أحمد مصطفى كامل**
20201700942	**نورهان عادل عبد الراضي زكى**
20201700943	**نورهان محمد ابراهيم انور**

Preprocessing

we use:

from matplotlib. Figure import Figure

import matplotlib. pyplot as plt

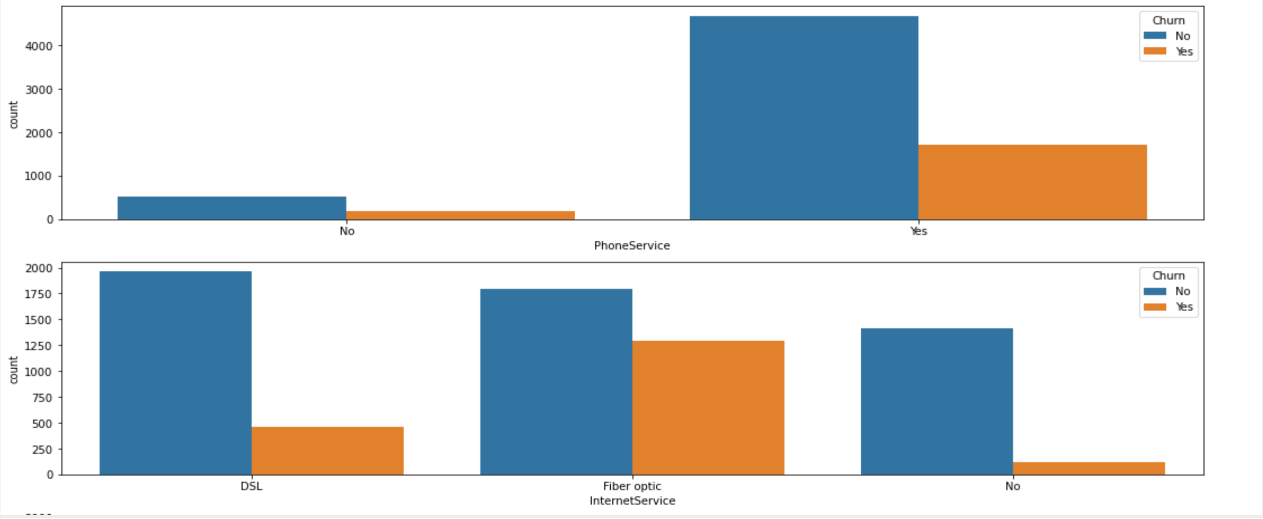
To show the plot and the Figure

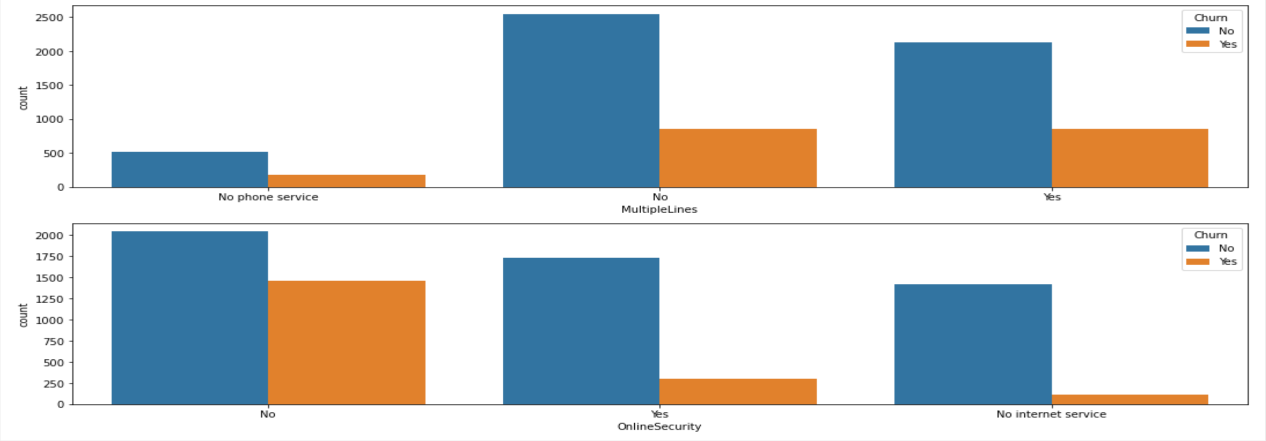
import seaborn as sns

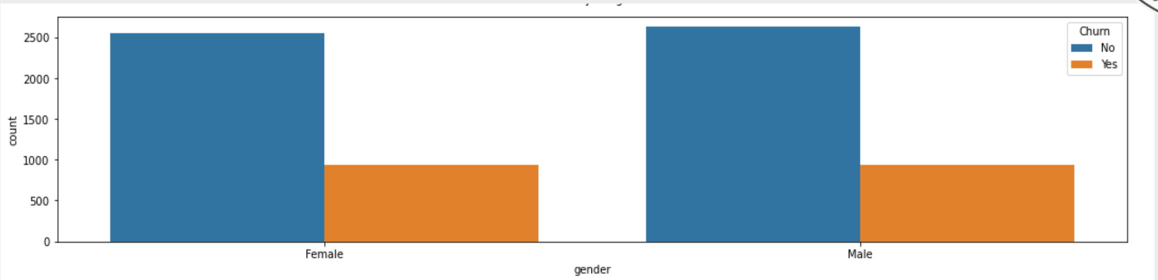
Seaborn based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

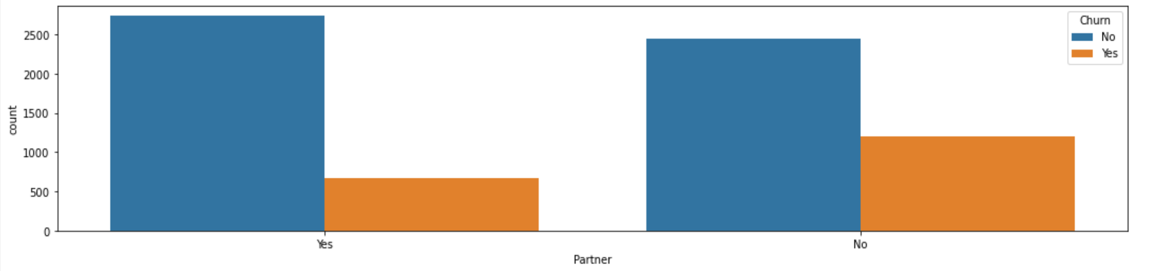
Sns.countplot (for each independent variables , dependent variables “churn”, data=our data set, ax= “Axes object to draw the plot onto”)

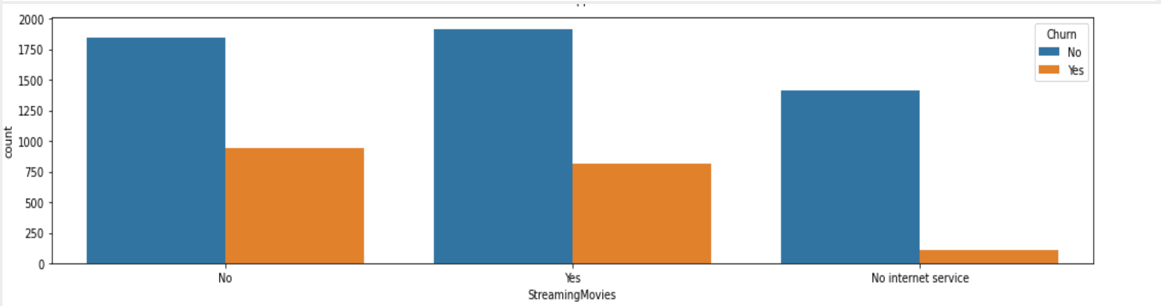
Put all the plots in function so we can call it where we want

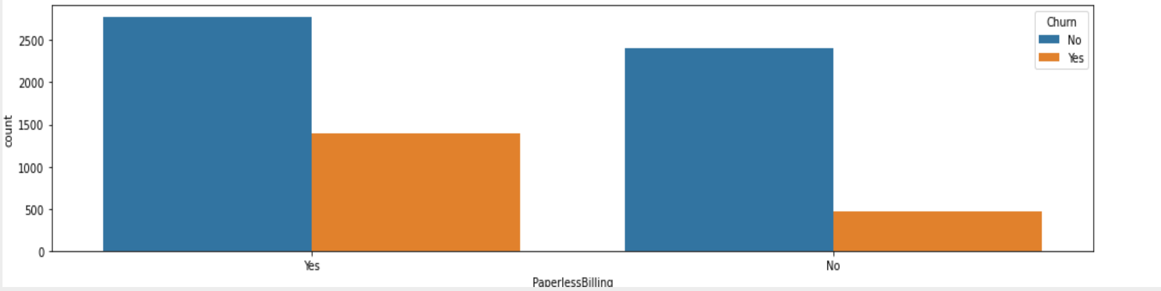


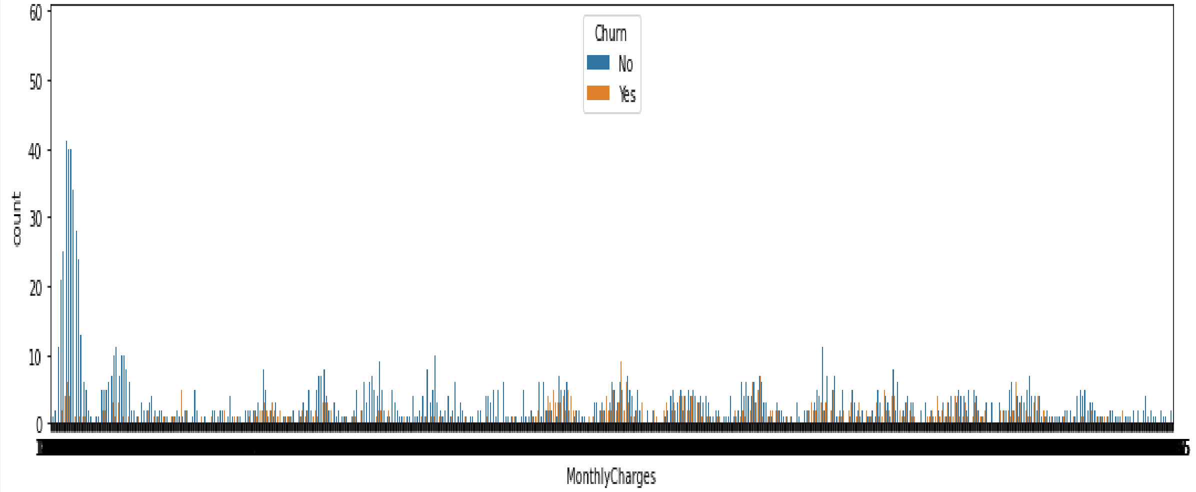


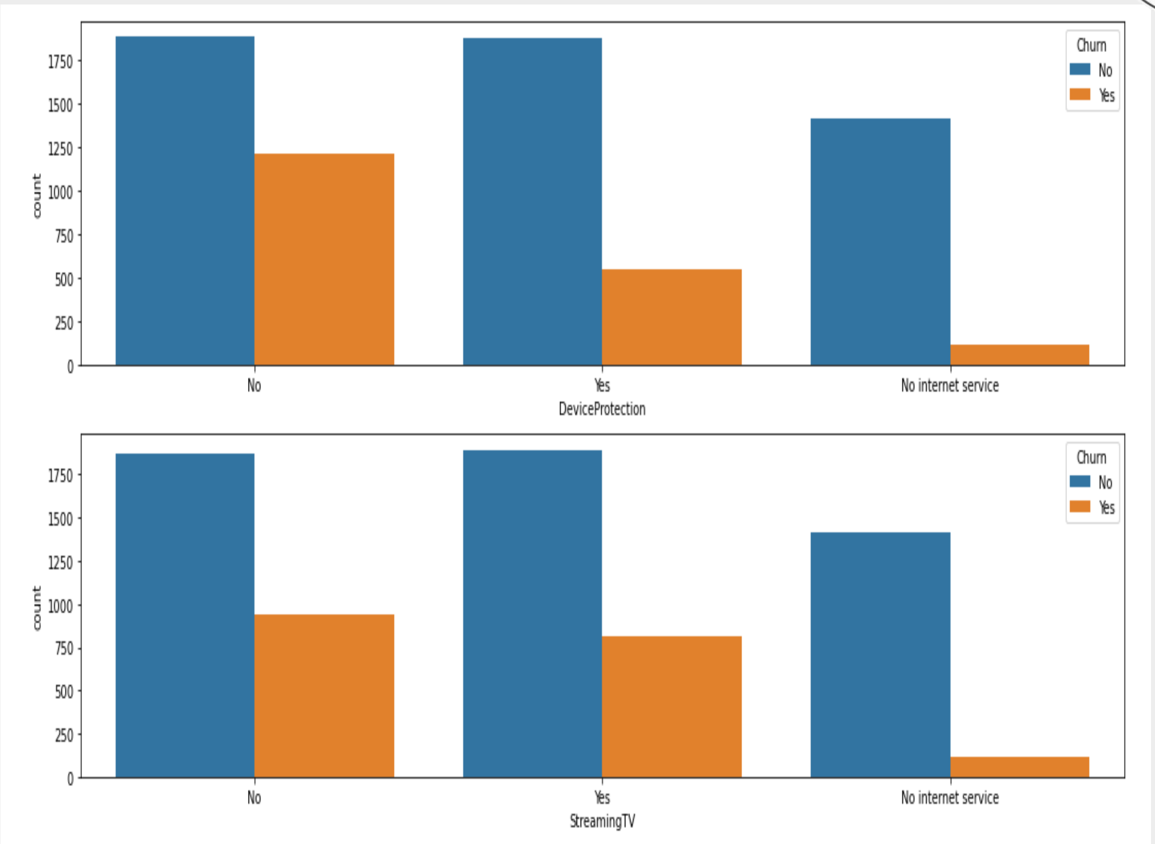

count
2500
2000
1500
1000
500
0
Female
Male
gender
Churn
No
Yes

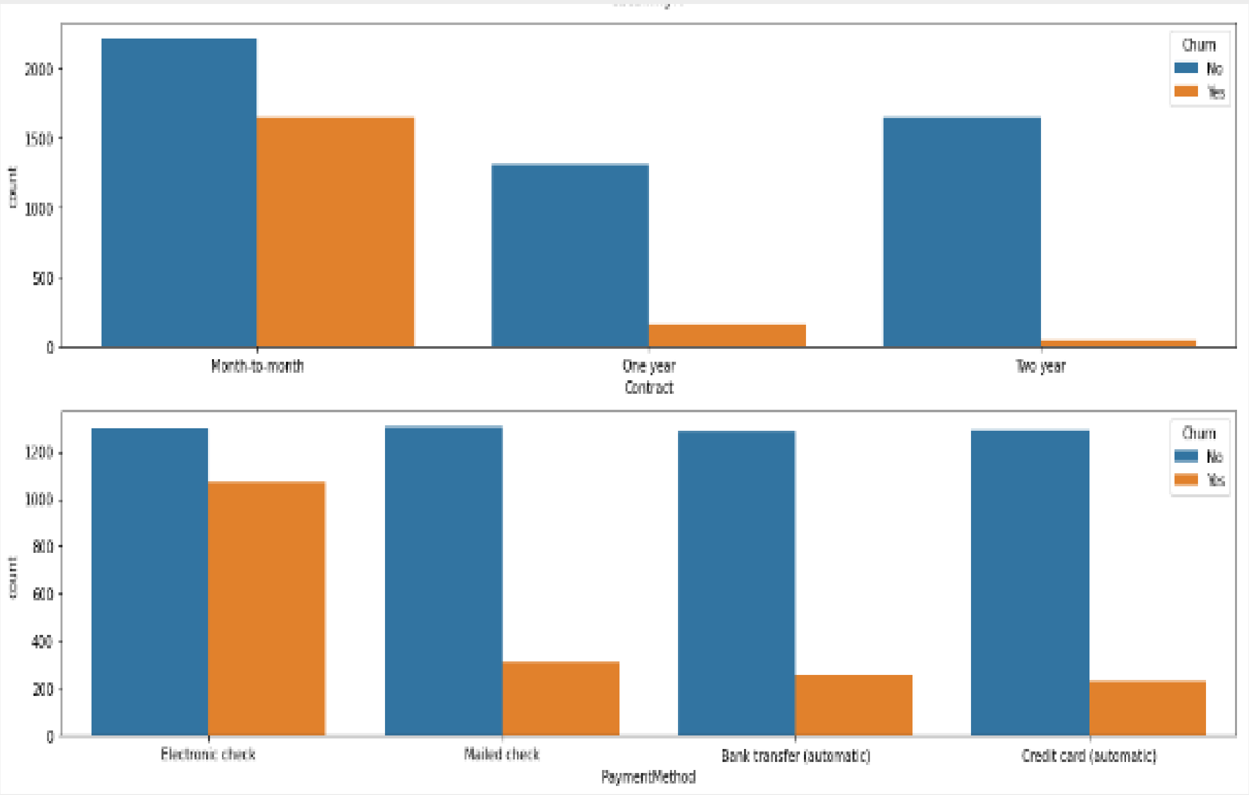

count
2500
2000
1500
1000
500
0
Yes
No
Partner
Churn
No
Yes

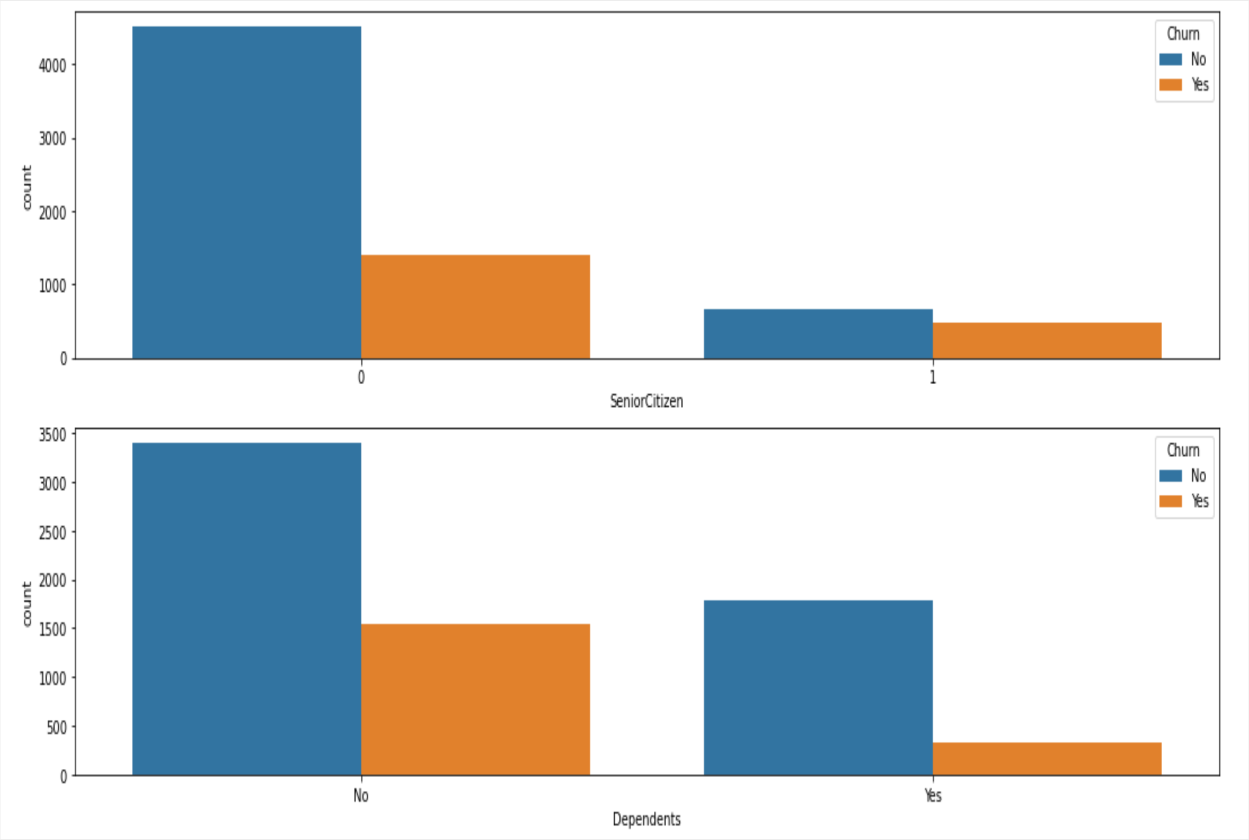

count
2000
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
StreamingMovies
Churn
No
Yes

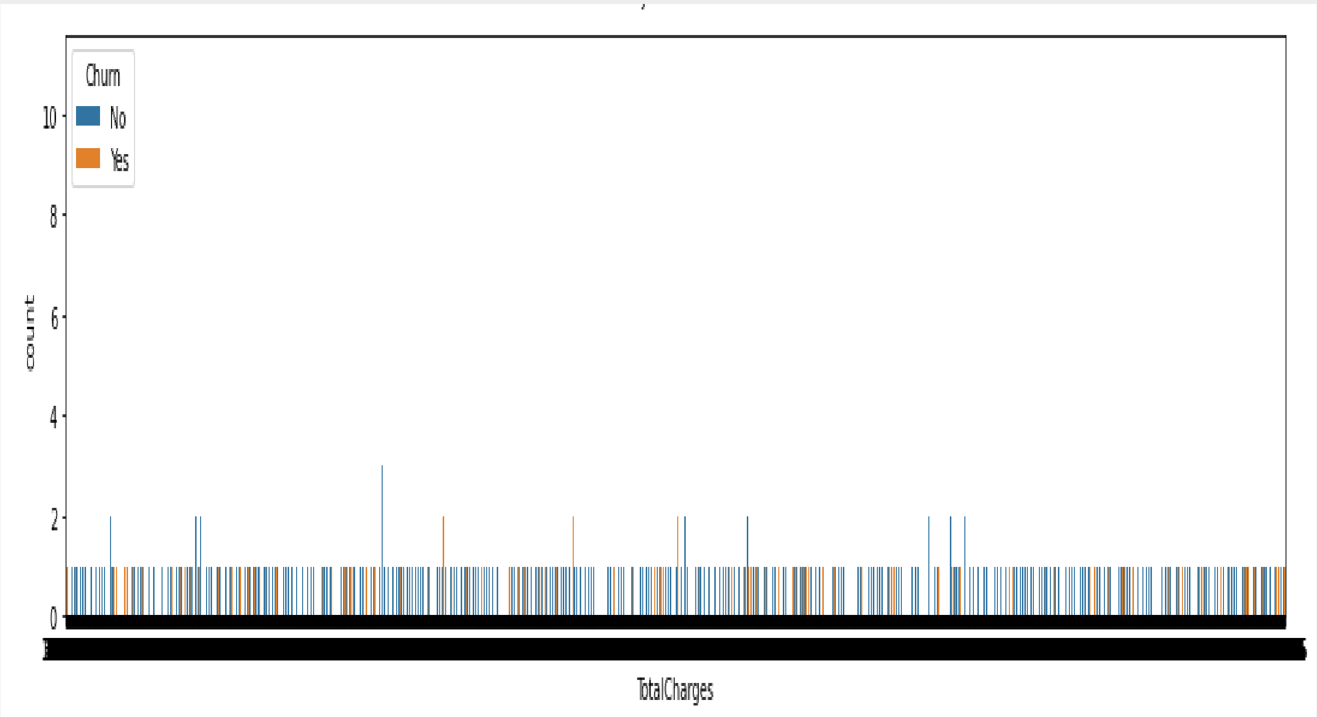

count
2500
2000
1500
1000
500
0
Yes
No
PaperlessBilling
Churn
No
Yes


count
60
50
40
30
20
10
0
MonthlyCharges
Churn
No
Yes


count
0
250
500
750
1000
1250
1500
1750
Churn
No
Yes
No
Yes
No internet service
DeviceProtection
count
0
250
500
750
1000
1250
1500
1750
Churn
No
Yes
No
Yes
No internet service
StreamingTV


count
0
500
1000
1500
2000
Churn
No
Yes
Month-to-month
One year
Two year
Contract
count
0
200
400
600
800
1000
1200
Churn
No
Yes
Electronic check
Mailed check
Bank transfer (automatic)
Credit card (automatic)
PaymentMethod


Churn
No
Yes
count
0
1000
2000
3000
4000
0
1
SeniorCitizen
Churn
No
Yes
count
0
500
1000
1500
2000
2500
3000
3500
No
Yes
Dependents

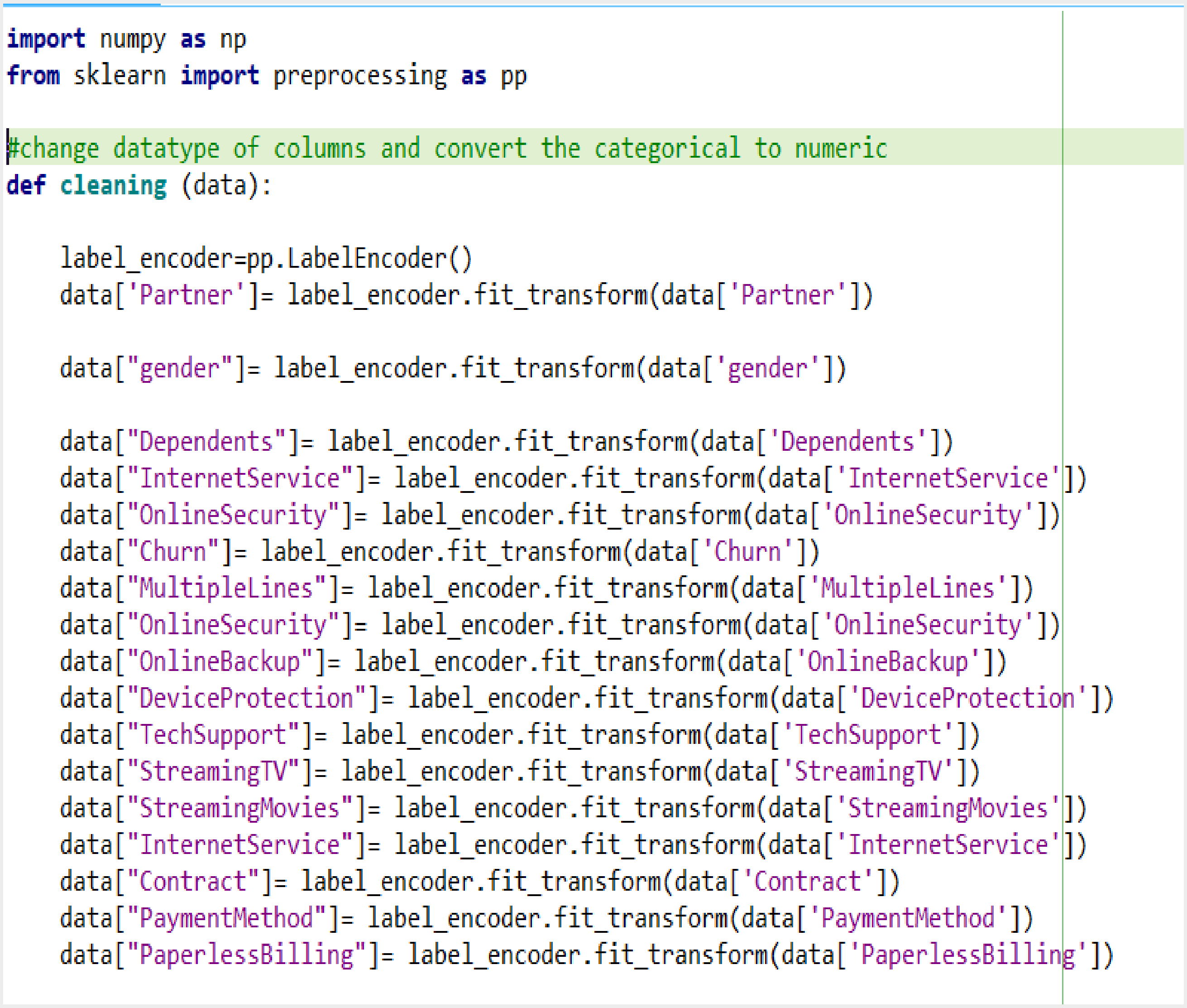

Churn
No
Yes
count
0
2
4
6
8
10
TotalCharges

Data Cleaning

To do that we have 4 steps

***** create a function to take our data to do cleaning and return it after cleaning**

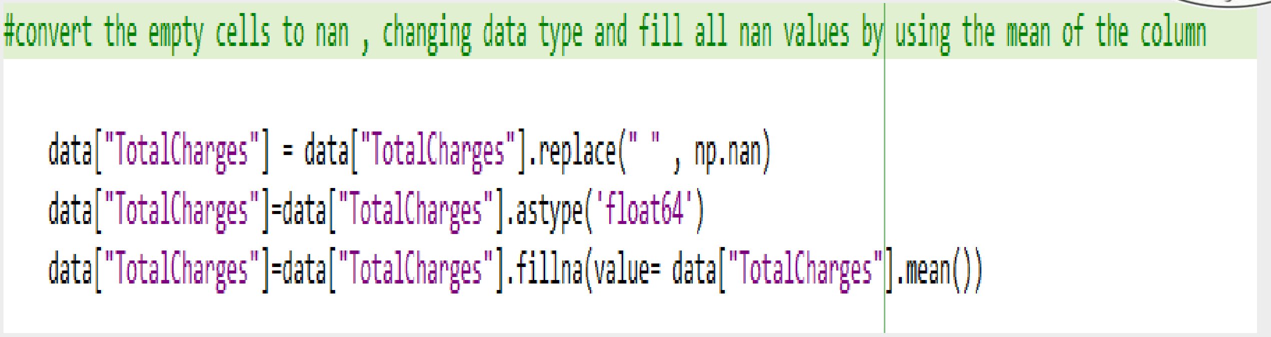
**First step:



```
import numpy as np
from sklearn import preprocessing as pp

#change datatype of columns and convert the categorical to numeric
def cleaning (data):

    label_encoder=pp.LabelEncoder()
    data['Partner']= label_encoder.fit_transform(data['Partner'])

    data["gender"]= label_encoder.fit_transform(data['gender'])

    data["Dependents"]= label_encoder.fit_transform(data['Dependents'])
    data["InternetService"]= label_encoder.fit_transform(data['InternetService'])
    data["OnlineSecurity"]= label_encoder.fit_transform(data['OnlineSecurity'])
    data["Churn"]= label_encoder.fit_transform(data['Churn'])
    data["MultipleLines"]= label_encoder.fit_transform(data['MultipleLines'])
    data["OnlineSecurity"]= label_encoder.fit_transform(data['OnlineSecurity'])
    data["OnlineBackup"]= label_encoder.fit_transform(data['OnlineBackup'])
    data["DeviceProtection"]= label_encoder.fit_transform(data['DeviceProtection'])
    data["TechSupport"]= label_encoder.fit_transform(data['TechSupport'])
    data["StreamingTV"]= label_encoder.fit_transform(data['StreamingTV'])
    data["StreamingMovies"]= label_encoder.fit_transform(data['StreamingMovies'])
    data["InternetService"]= label_encoder.fit_transform(data['InternetService'])
    data["Contract"]= label_encoder.fit_transform(data['Contract'])
    data["PaymentMethod"]= label_encoder.fit_transform(data['PaymentMethod'])
    data["PaperlessBilling"]= label_encoder.fit_transform(data['PaperlessBilling'])
```

- **We change our data to numeric by use (label_Encoder : to change yes &no &… To 0 & 1&…)&(fit_transform : to change strings to numeric by alphabetical order)**

**Second step:

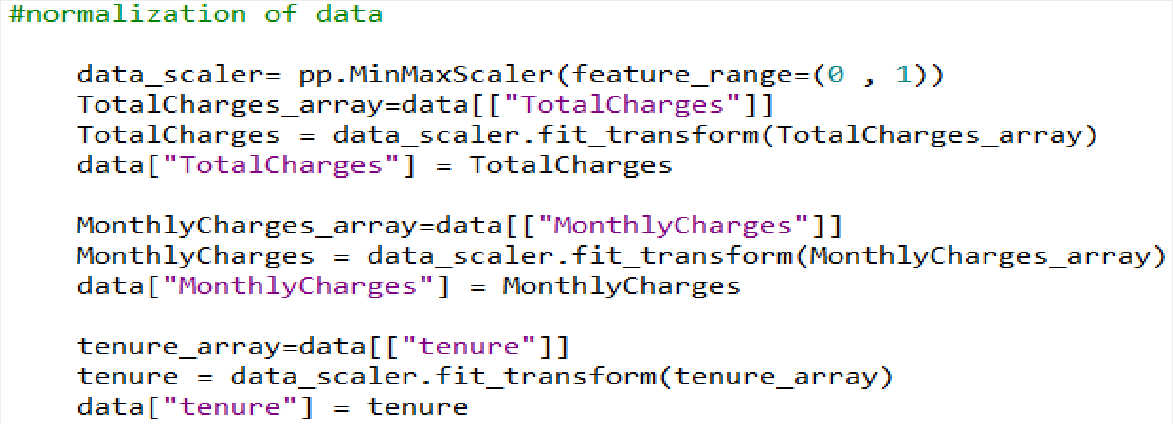


```
#convert the empty cells to nan , changing data type and fill all nan values by using the mean of the column


   data["TotalCharges"] = data["TotalCharges"].replace(" " , np.nan)
   data["TotalCharges"]=data["TotalCharges"].astype('float64')
   data["TotalCharges"]=data["TotalCharges"].fillna(value= data["TotalCharges"].mean())
```

- We have an empty cells in Totalcharges column so we handling that by turn empty cells to Null and replace Null to the mean of data[TotalCharges]
- We change data type of column form object to float64 to be numeric

----

**Third step: “Data Scalling”

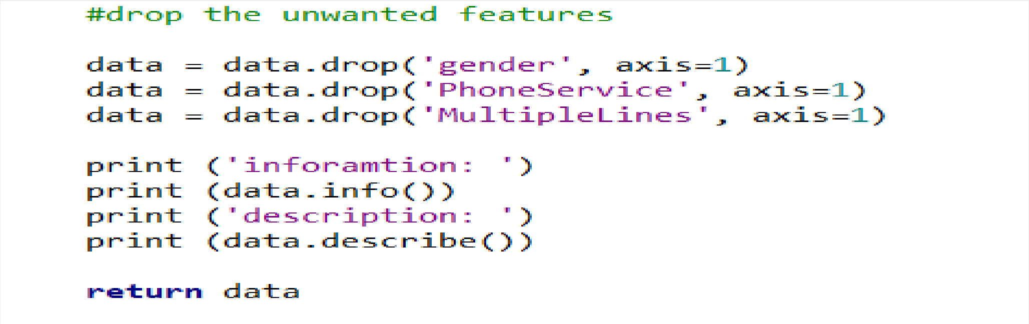


```
#normalization of data

    data_scaler= pp.MinMaxScaler(feature_range=(0 , 1))
    TotalCharges_array=data[["TotalCharges"]]
    TotalCharges = data_scaler.fit_transform(TotalCharges_array)
    data["TotalCharges"] = TotalCharges

    MonthlyCharges_array=data[["MonthlyCharges"]]
    MonthlyCharges = data_scaler.fit_transform(MonthlyCharges_array)
    data["MonthlyCharges"] = MonthlyCharges

    tenure_array=data[["tenure"]]
    tenure = data_scaler.fit_transform(tenure_array)
    data["tenure"] = tenure
```

- We have 3 columns (TotalCharges , MonthlyCharges,Tenure) numeric but its very height and different so that we need to normalize this column to predict correct

- We make min &max range to numbers between (0,1) by using MinMaxScaler and put it in data_scaler
- Puting data of column in array in order to have the appility to make scaling
- Makeing scaling in array and fit it by fit_tranform and put it in object
- Put that object after make scaling in its column in data

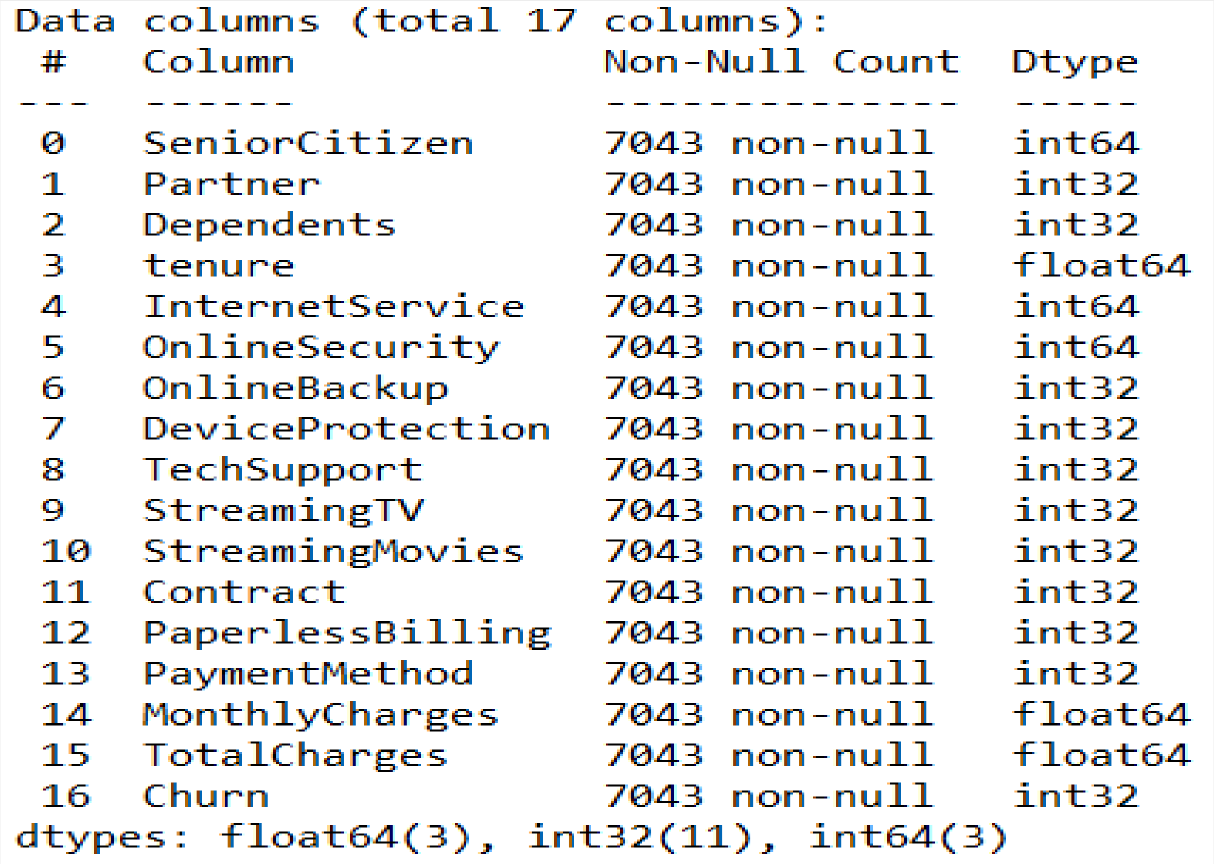
**Forth step:



```
    #drop the unwanted features

    data = data.drop('gender', axis=1)
    data = data.drop('PhoneService', axis=1)
    data = data.drop('MultipleLines', axis=1)

    print ('inforamtion: ')
    print (data.info())
    print ('description: ')
    print (data.describe())

    return data
```

- After cleaning and pre-processing we will drop unwanted features that doesn't affect when we predict (gender ,Phone Service ,Multiple lines)
- print information to see our data types and number of Nulls
- print data description to see our first 5 rows information to see data after cleaning and number of rows and columns.

Data after Cleaning



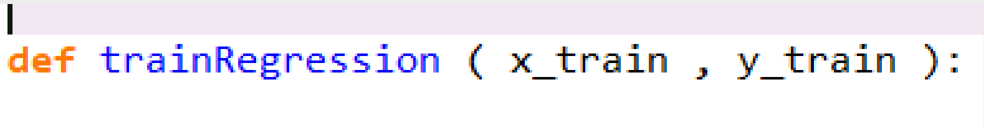
```
Data columns (total 17 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   SeniorCitizen     7043 non-null   int64
 1   Partner           7043 non-null   int32
 2   Dependents        7043 non-null   int32
 3   tenure            7043 non-null   float64
 4   InternetService   7043 non-null   int64
 5   OnlineSecurity    7043 non-null   int64
 6   OnlineBackup      7043 non-null   int32
 7   DeviceProtection  7043 non-null   int32
 8   TechSupport       7043 non-null   int32
 9   StreamingTV       7043 non-null   int32
 10  StreamingMovies   7043 non-null   int32
 11  Contract          7043 non-null   int32
 12  PaperlessBilling  7043 non-null   int32
 13  PaymentMethod     7043 non-null   int32
 14  MonthlyCharges    7043 non-null   float64
 15  TotalCharges      7043 non-null   float64
 16  Churn             7043 non-null   int32
dtypes: float64(3), int32(11), int64(3)
```

Algorithms

Logistic Regression

For train data:

- We import from sklearn. linear_model import LogisticRegression.
- We made function to train data take two parameters x_train and y_train and return LR (object of LogisticRegression class).

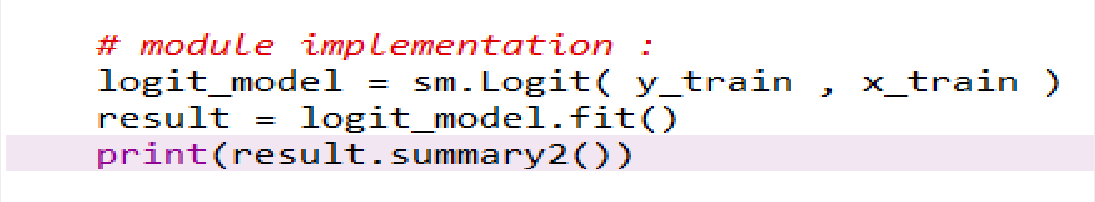


```
def trainRegression ( x_train , y_train ):
```

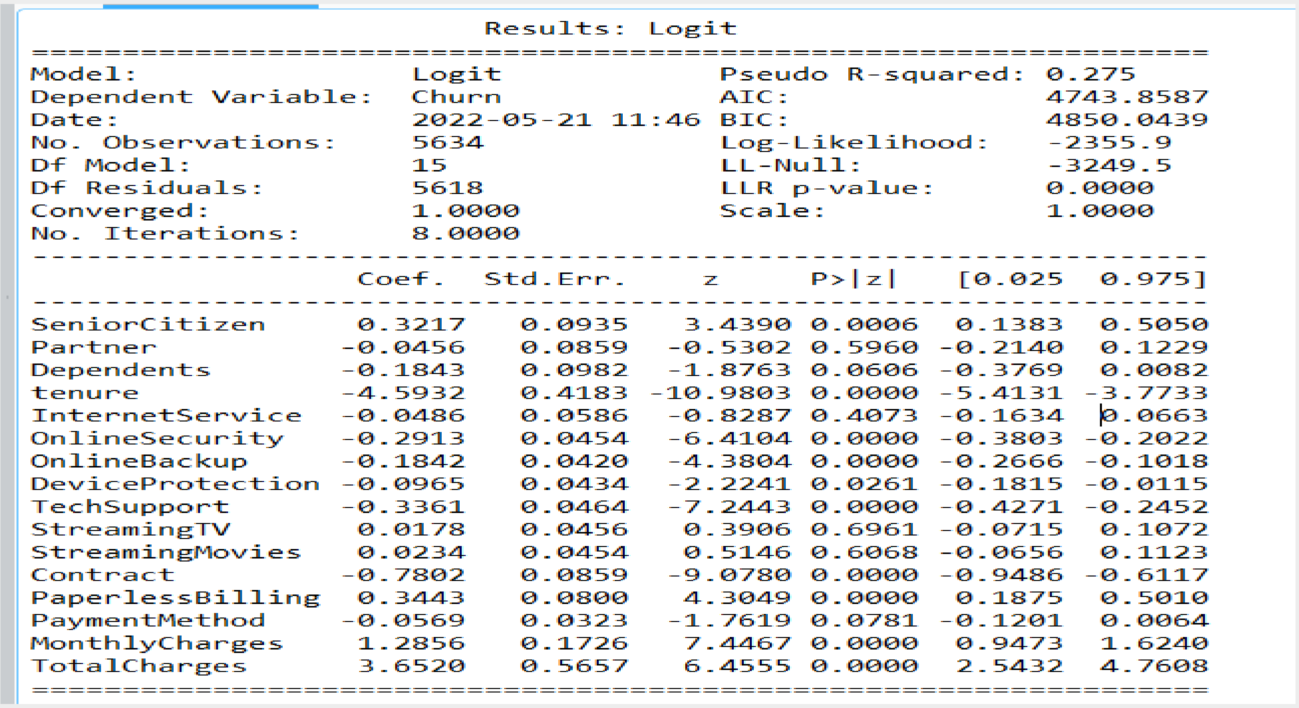
-

For module implementation:

- **We import statsmodels.api as sm to print the result summary.**

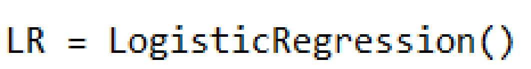


```
# module implementation :
logit_model = sm.Logit( y_train , x_train )
result = logit_model.fit()
print(result.summary2())
```



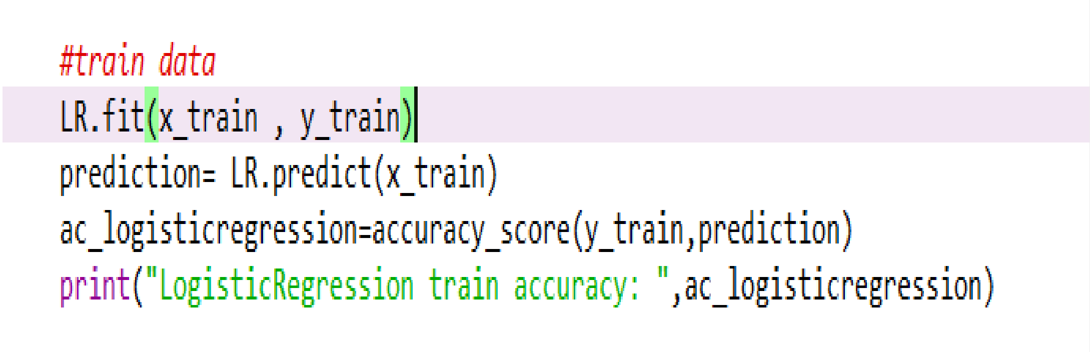
```
                         Results: Logit
=================================================================
Model:              Logit            Pseudo R-squared: 0.275
Dependent Variable: Churn            AIC:              4743.8587
Date:               2022-05-21 11:46 BIC:              4850.0439
No. Observations:   5634             Log-Likelihood:   -2355.9
Df Model:           15               LL-Null:          -3249.5
Df Residuals:       5618             LLR p-value:      0.0000
Converged:          1.0000           Scale:            1.0000
No. Iterations:     8.0000
-----------------------------------------------------------------
                  Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
-----------------------------------------------------------------
SeniorCitizen     0.3217   0.0935   3.4390 0.0006  0.1383  0.5050
Partner          -0.0456   0.0859  -0.5302 0.5960 -0.2140  0.1229
Dependents       -0.1843   0.0982  -1.8763 0.0606 -0.3769  0.0082
tenure           -4.5932   0.4183 -10.9803 0.0000 -5.4131 -3.7733
InternetService  -0.0486   0.0586  -0.8287 0.4073 -0.1634  0.0663
OnlineSecurity   -0.2913   0.0454  -6.4104 0.0000 -0.3803 -0.2022
OnlineBackup     -0.1842   0.0420  -4.3804 0.0000 -0.2666 -0.1018
DeviceProtection -0.0965   0.0434  -2.2241 0.0261 -0.1815 -0.0115
TechSupport      -0.3361   0.0464  -7.2443 0.0000 -0.4271 -0.2452
StreamingTV       0.0178   0.0456   0.3906 0.6961 -0.0715  0.1072
StreamingMovies   0.0234   0.0454   0.5146 0.6068 -0.0656  0.1123
Contract         -0.7802   0.0859  -9.0780 0.0000 -0.9486 -0.6117
PaperlessBilling  0.3443   0.0800   4.3049 0.0000  0.1875  0.5010
PaymentMethod    -0.0569   0.0323  -1.7619 0.0781 -0.1201  0.0064
MonthlyCharges    1.2856   0.1726   7.4467 0.0000  0.9473  1.6240
TotalCharges      3.6520   0.5657   6.4555 0.0000  2.5432  4.7608
=================================================================
```

- doing an object form Logistic Regression class so it can be used for train data.



```
LR = LogisticRegression()
```

- we use from sklearn. metrics import accuracy_score to calculate accuracy.
- We use fit() function to train data.

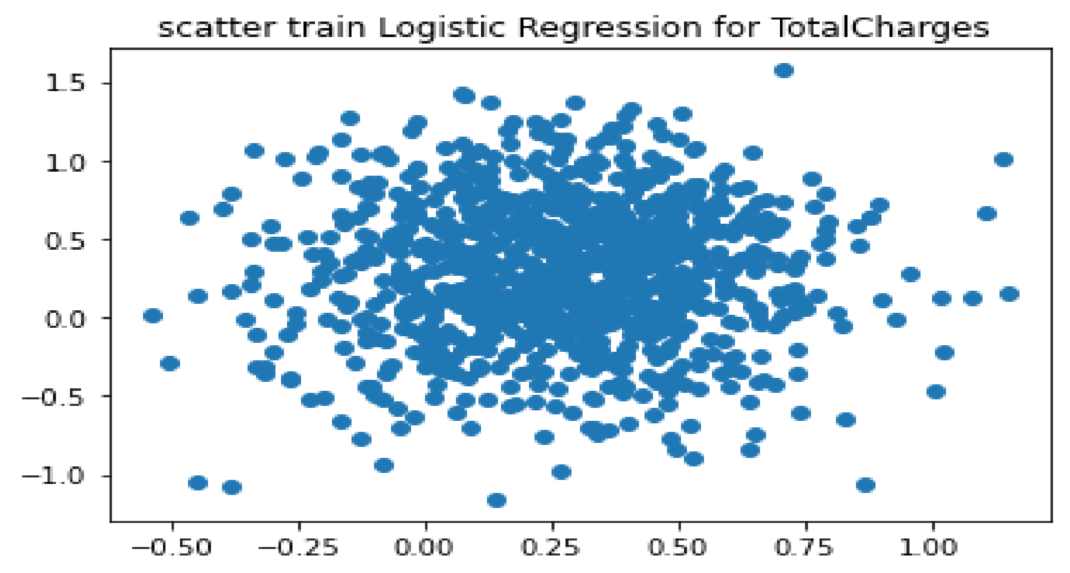


```
#train data
LR.fit(x_train , y_train)
prediction= LR.predict(x_train)
ac_logisticregression=accuracy_score(y_train,prediction)
print("LogisticRegression train accuracy: ",ac_logisticregression)
```

**Logistic Regression train accuracy: 0.8004969826056088

For data scatter:

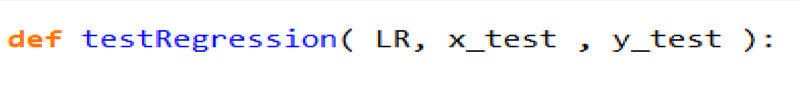
- Scatter train between Total Charges and churn:



- And in the end, we return LR (object of LogisticRegression class).

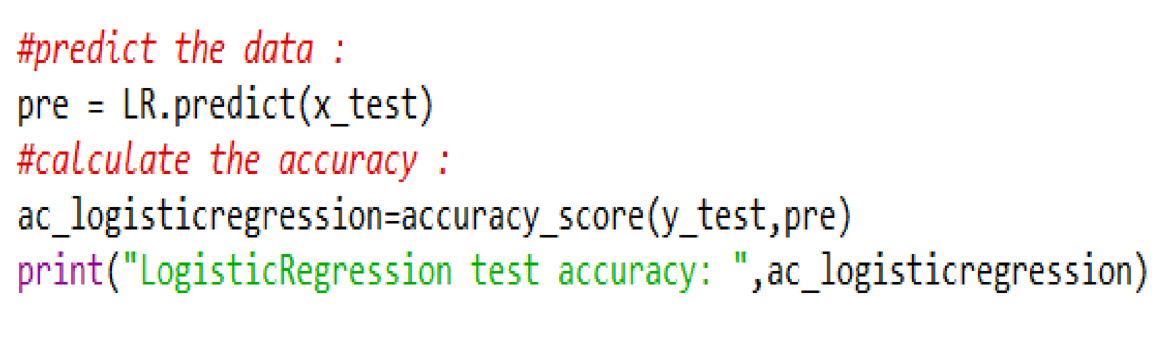
For test data:

- We made function to test data take three parameters x_test, y_test and LR (object of LogisticRegression class made fit () for the data) .



```
def testRegression( LR, x_test , y_test ):
```

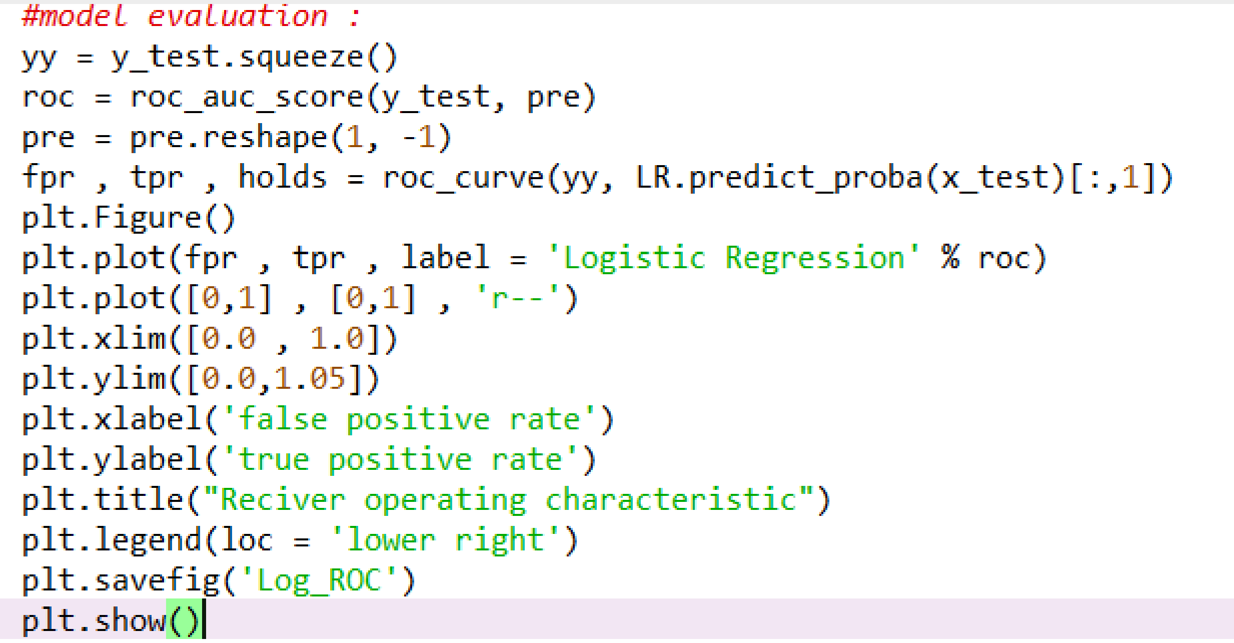
- We use predict () function to predict the target then calculate the accuracy.



```
#predict the data :
pre = LR.predict(x_test)
#calculate the accuracy :
ac_logisticregression=accuracy_score(y_test,pre)
print("LogisticRegression test accuracy: ",ac_logisticregression)
```

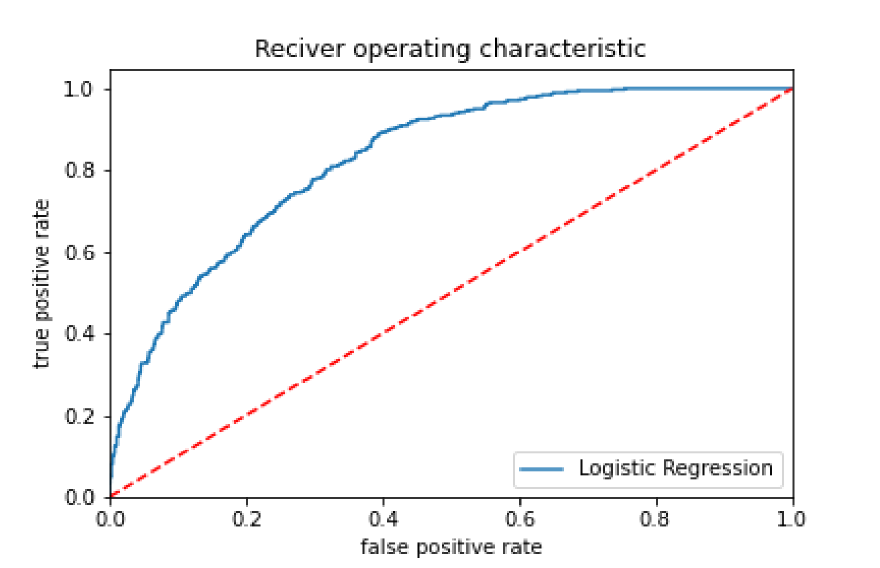
For module implementation:

- We use from sklearn. metrics import roc_curve to calculate the logistic curve.



```
#model evaluation :
yy = y_test.squeeze()
roc = roc_auc_score(y_test, pre)
pre = pre.reshape(1, -1)
fpr , tpr , holds = roc_curve(yy, LR.predict_proba(x_test)[:,1])
plt.Figure()
plt.plot(fpr , tpr , label = 'Logistic Regression' % roc)
plt.plot([0,1] , [0,1] , 'r--')
plt.xlim([0.0 , 1.0])
plt.ylim([0.0,1.05])
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.title("Reciver operating characteristic")
plt.legend(loc = 'lower right')
plt.savefig('Log_ROC')
plt.show()
```

- **We use import matplotlib. Pyplot as plt**

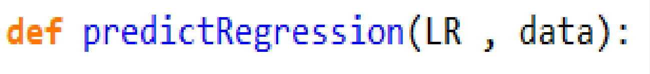


****Logistic Regression test accuracy: 0.801277501774308**

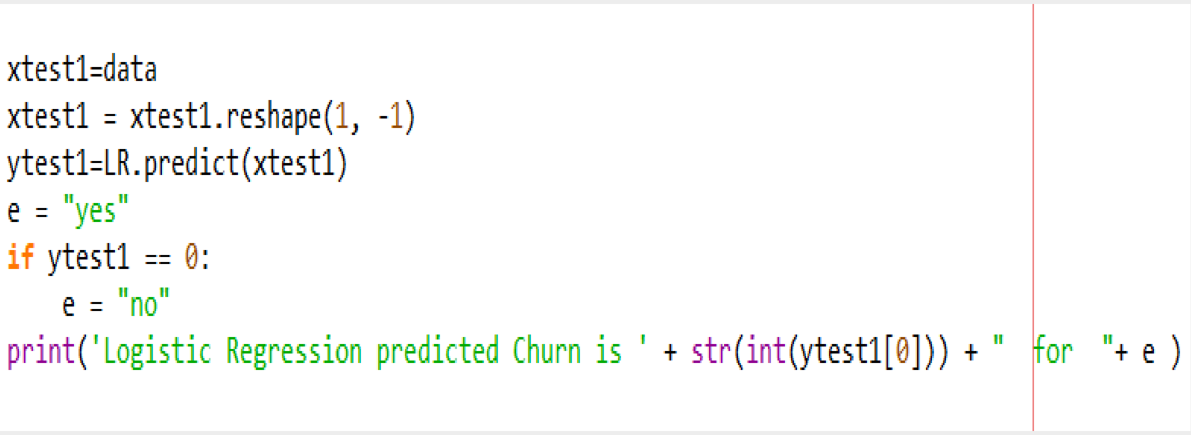
And in the end, we return LR (object of LogisticRegression class).

For predict data:

- We made function to predict data take two parameter data (1D array) and LR.



```
def predictRegression(LR , data):
```



```
xtest1=data
xtest1 = xtest1.reshape(1, -1)
ytest1=LR.predict(xtest1)
e = "yes"
if ytest1 == 0:
    e = "no"
print('Logistic Regression predicted Churn is ' + str(int(ytest1[0])) + "  for  "+ e )
```

We convert the 1D array to 2D array using reshape () function.

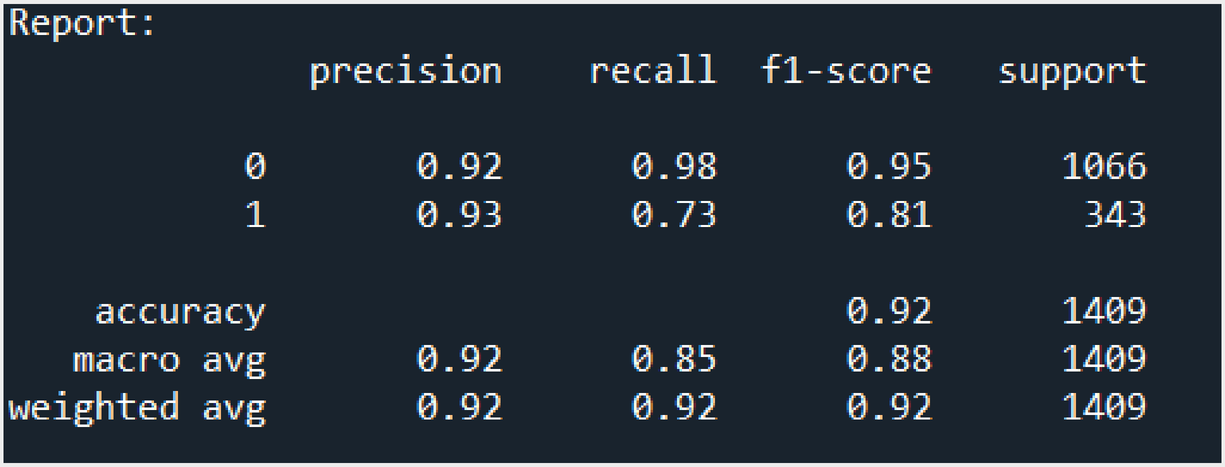
SVM

We use:

from sklearn.svm import SVC

from sklearn. metrics import classification_report

for example → **svm_test_report:**

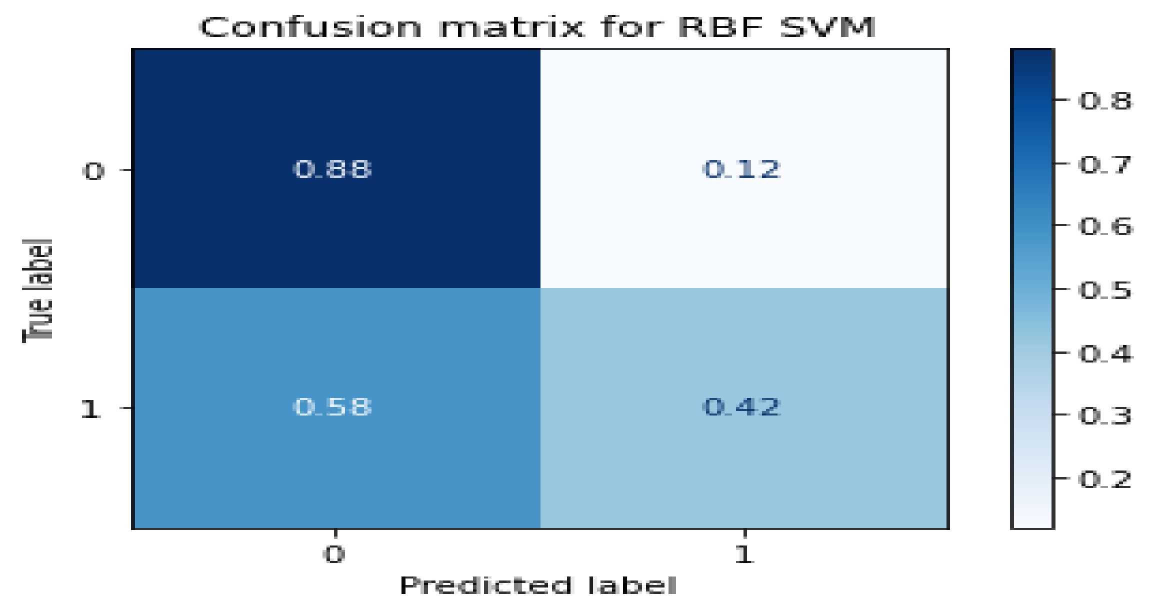


```
Report:
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      1066
           1       0.93      0.73      0.81       343

    accuracy                           0.92      1409
   macro avg       0.92      0.85      0.88      1409
weighted avg       0.92      0.92      0.92      1409
```

from matplotlib import pyplot as plt

from sklearn. metrics import plot_confusion_matrix

for confusion_matrix:

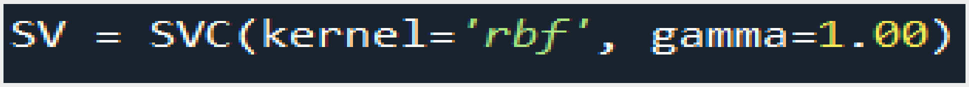


from sklearn. metrics import accuracy_score

To show accuracy.

For function train:

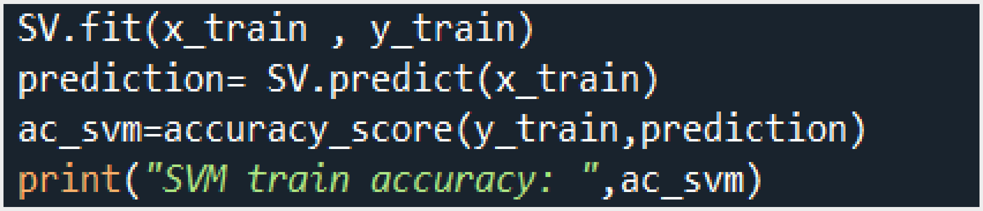
doing an object form svc so it can be used for train data.



```
SV = SVC(kernel='rbf', gamma=1.00)
```

- Doing fitting for x_train, y_train by using function fit()
- Doing predict by using function predict(x_train)
- calculate accuracy

SVM Accuracy for train: **0.8757543485977991.**

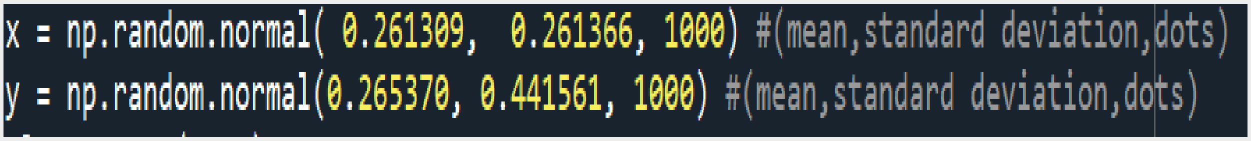


```
SV.fit(x_train , y_train)
prediction= SV.predict(x_train)
ac_svm=accuracy_score(y_train,prediction)
print("SVM train accuracy: ",ac_svm)
```

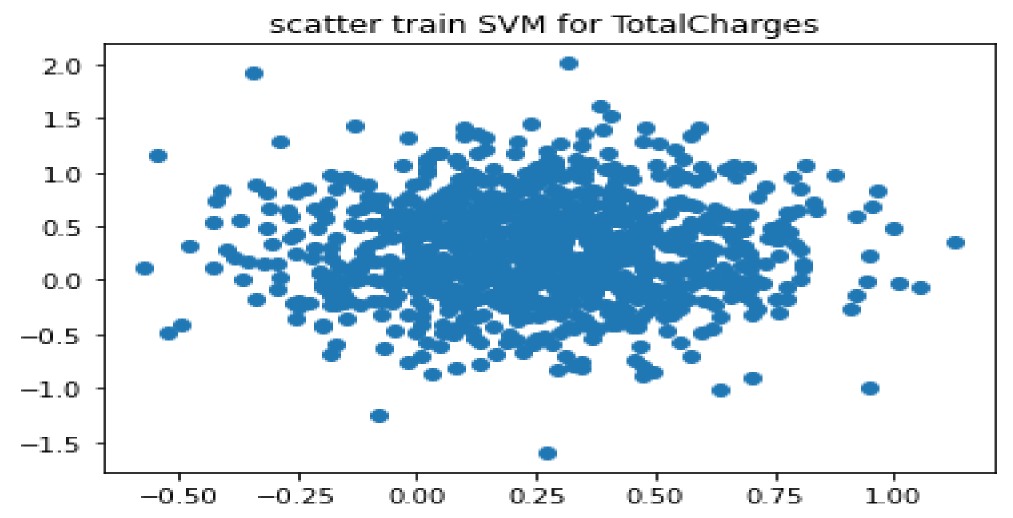
- And in the end, we return SV (object of SVC class).

For data scatter :

Scatter train between Total Charges and churn:

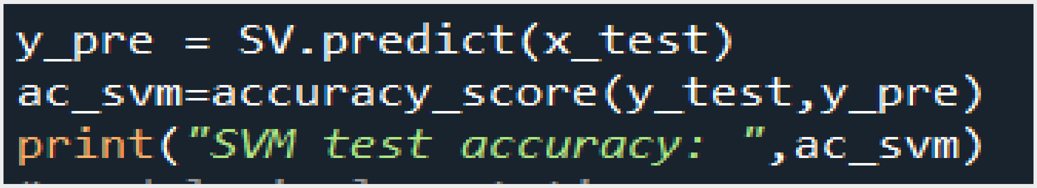


```
x = np.random.normal( 0.261309,  0.261366, 1000) #(mean,standard deviation,dots)
y = np.random.normal(0.265370, 0.441561, 1000) #(mean,standard deviation,dots)
```



For function test:

- We made function to test data take three parameters x_test, y_test and SV (object of SVC class made fit () for the data) .
- We use predict () function to predict the target then calculate the accuracy.



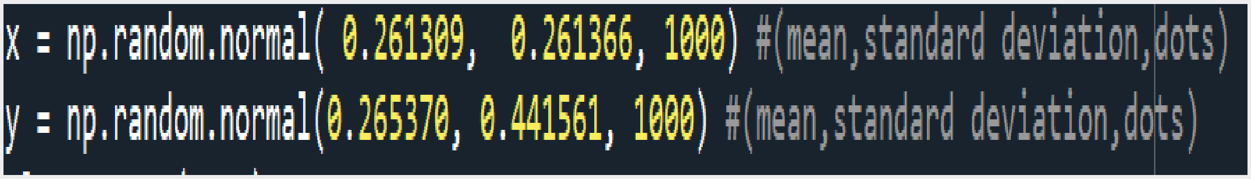
```
y_pre = SV.predict(x_test)
ac_svm=accuracy_score(y_test,y_pre)
print("SVM test accuracy: ",ac_svm)
```

SVM Accuracy for test: 0.7665010645848119.

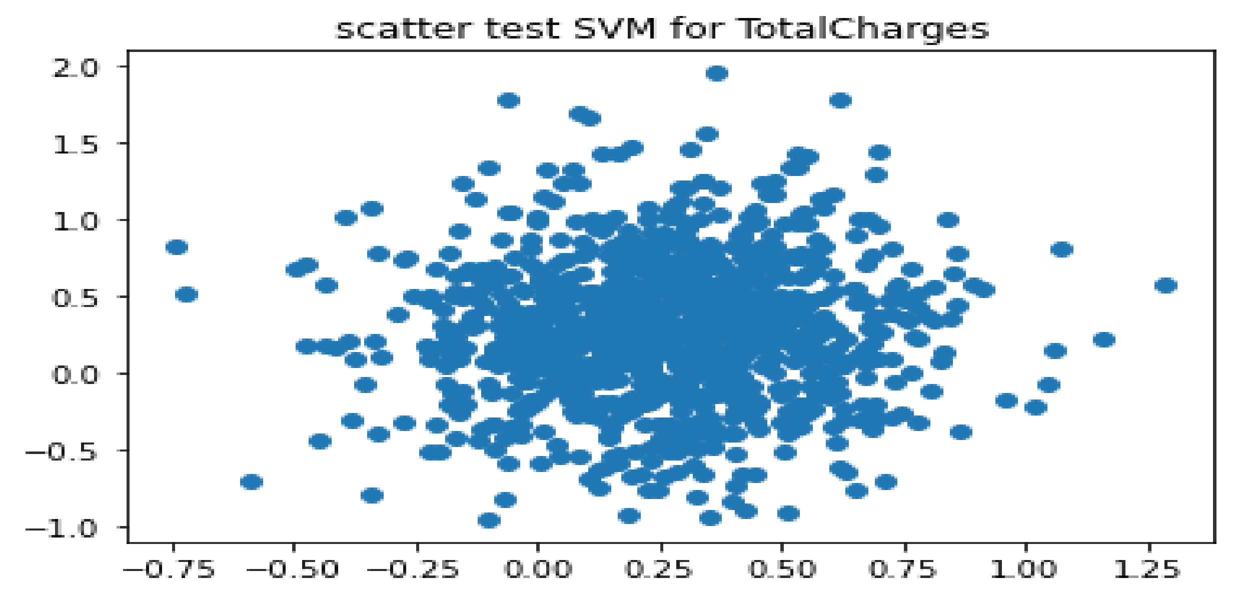
- And in the end of the test function, we return SV(object of SVC class) .

For data scatter :

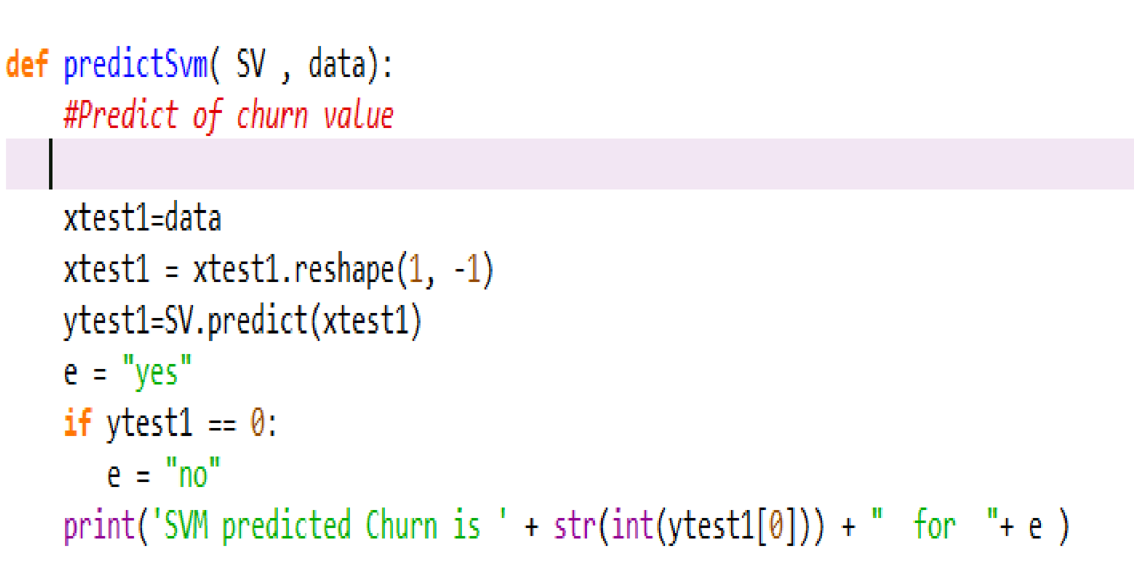
Scatter test between Total Charges and churn:



```
x = np.random.normal( 0.261309,  0.261366, 1000) #(mean,standard deviation,dots)
y = np.random.normal(0.265370, 0.441561, 1000) #(mean,standard deviation,dots)
```



For function predict:



```
def predictSvm( SV , data):
    #Predict of churn value

    xtest1=data
    xtest1 = xtest1.reshape(1, -1)
    ytest1=SV.predict(xtest1)
    e = "yes"
    if ytest1 == 0:
       e = "no"
    print('SVM predicted Churn is ' + str(int(ytest1[0])) + "  for  "+ e )
```

Data variable is the input from the user and reshape it then doing predict and SV (an object of SVC) that made fit () for the data in train function

We convert the 1D array to 2D array using reshape() function.

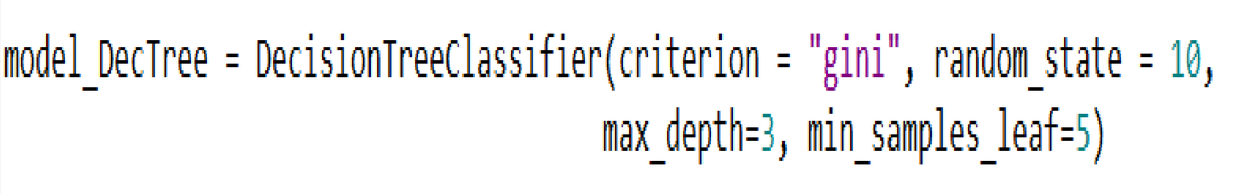
And doing predict to data , If predict is equal 1 then churn is YES, else the churn is NO

Decision Tree

We use:

from sklearn. tree import DecisionTreeClassifier

To make an object from DecisionTreeClassifier, so it can be used for train data.



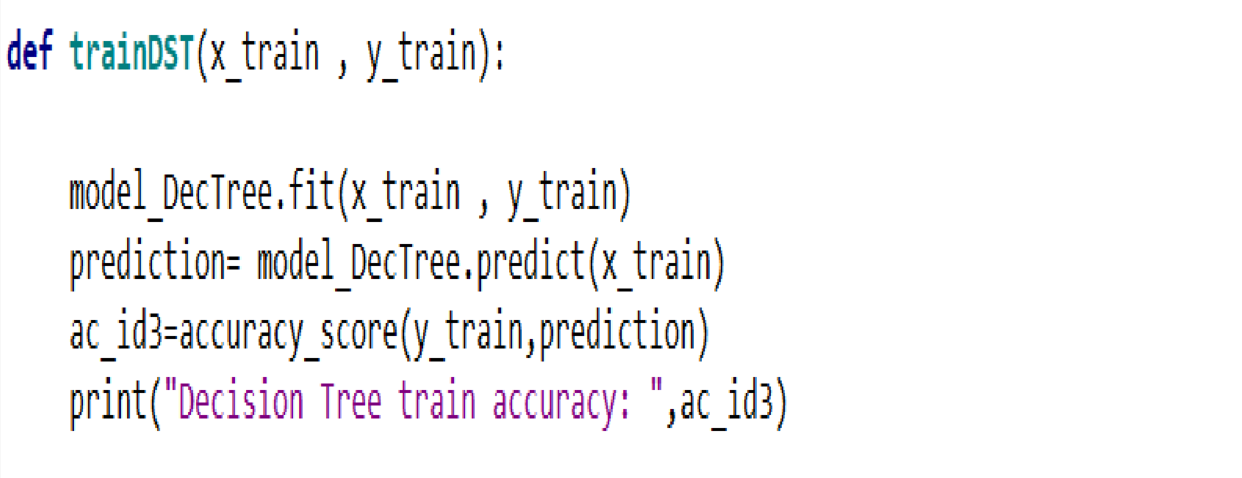
```
model_DecTree = DecisionTreeClassifier(criterion = "gini", random_state = 10,
                                       max_depth=3, min_samples_leaf=5)
```

from sklearn. metrics import accuracy_score

to show accuracy of Decision Tree.

For function Train:

Doing fitting for (x_train, y_train for Training)



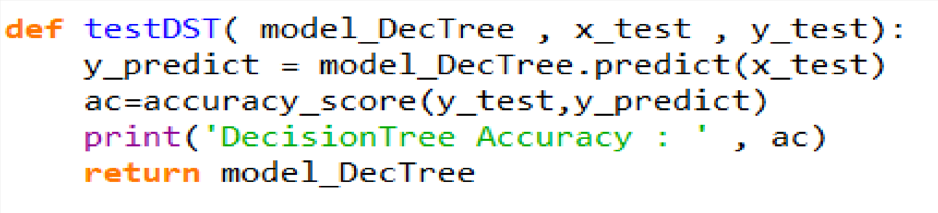
```
def trainDST(x_train , y_train):


    model_DecTree.fit(x_train , y_train)
    prediction= model_DecTree.predict(x_train)
    ac_id3=accuracy_score(y_train,prediction)
    print("Decision Tree train accuracy: ",ac_id3)
```

Decision Tree Accuracy for Training: 0.7825701100461484

For function Test:

Doing predict predict(x_test), and finally calculate accuracy for ytest and y_predict

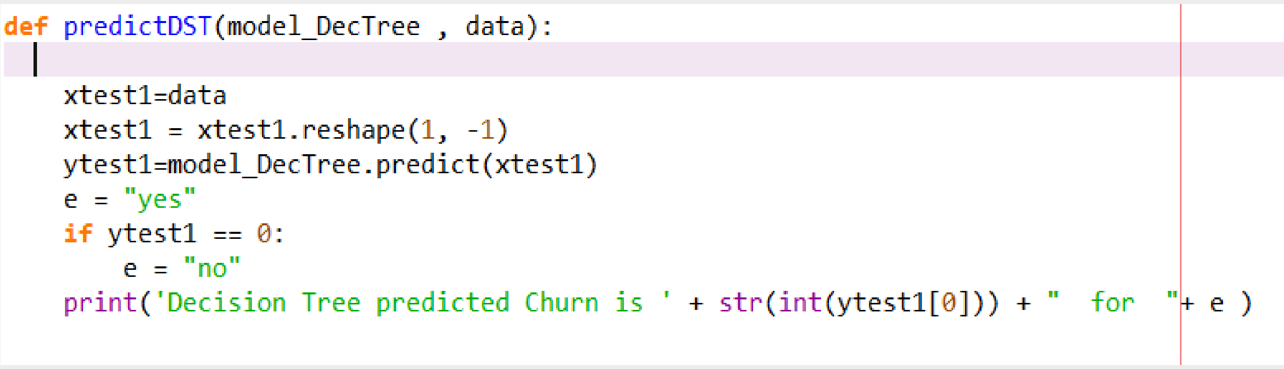
Taking a model_DecTree (an object of DecisionTreeClassifier) that made fit for the data in train function.



```
def testDST( model_DecTree , x_test , y_test):
    y_predict = model_DecTree.predict(x_test)
    ac=accuracy_score(y_test,y_predict)
    print('DecisionTree Accuracy : ' , ac)
    return model_DecTree
```

Decision Tree Accuracy for Testing: 0.7821149751596878

For function Predict:



```
def predictDST(model_DecTree , data):

    xtest1=data
    xtest1 = xtest1.reshape(1, -1)
    ytest1=model_DecTree.predict(xtest1)
    e = "yes"
    if ytest1 == 0:
        e = "no"
    print('Decision Tree predicted Churn is ' + str(int(ytest1[0])) + "  for  "+ e )
```

Data variable is the input from the user and reshape it then doing predict

And model_DecTree (an object of DecisionTreeClassifier) that made fit () for the data in train function and predict () for data in test function.

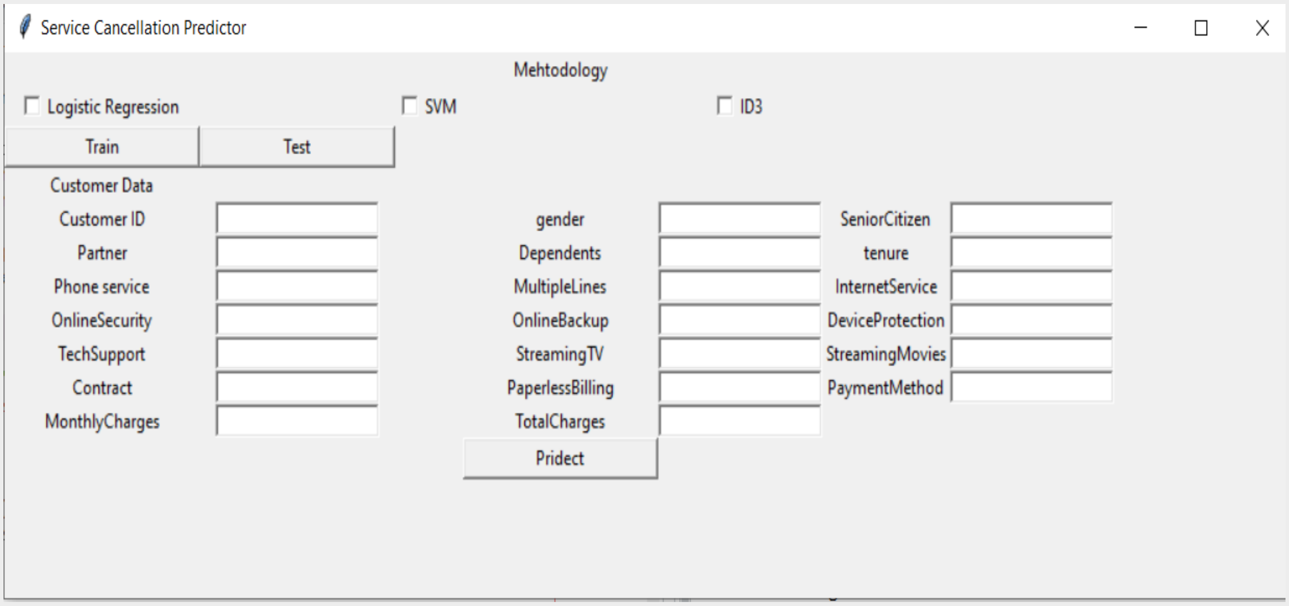
If predict is equal 1 then churn is YES, else the churn is NO

Comparing among 3 Algorithms

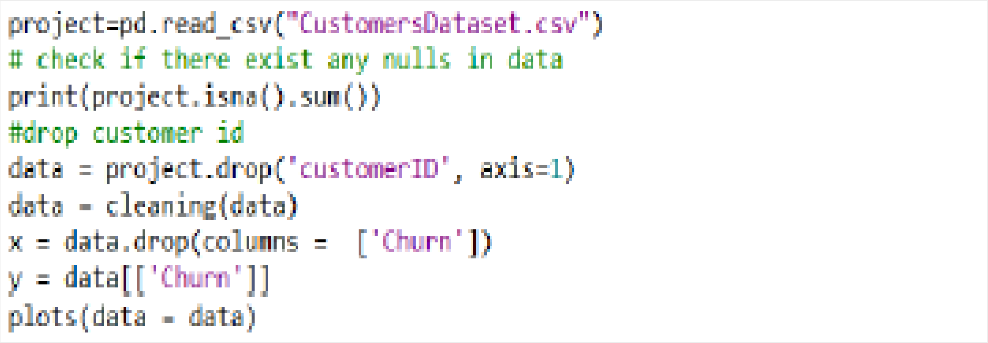
Algorithm Accuracy	Logistic Regression	SVM	Decision Tree
train	0.8004969826056088	0.8757543485977991	0.7825701100461484
test	0.801277501774308	0.7665010645848119	0.7821149751596878

GUI

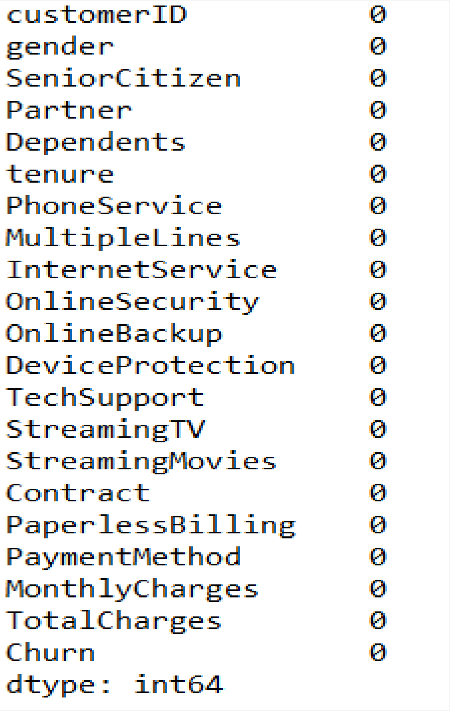
GUI module:



- We use import tkinter as tk for GUI.
- We use from sklearn. model_selection import train_test_split to split data.
- First , we must read file (CustomersDataset.csv) , and use function (isna().sum()) to count null values in data , drop the column "customerID" , and then clean the data , and then show plots

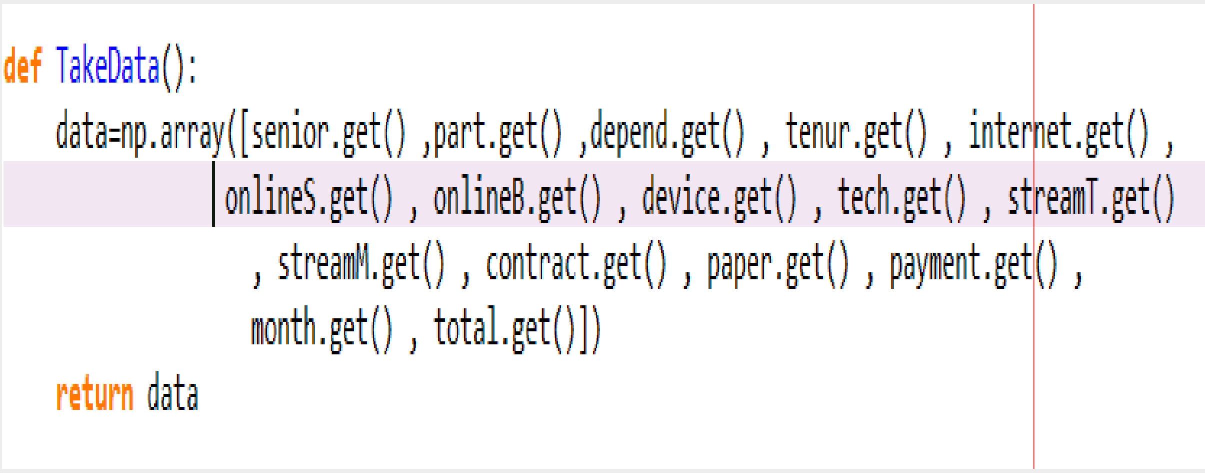


```
project=pd.read_csv("CustomersDataset.csv")
# check if there exist any nulls in data
print(project.isna().sum())
#drop customer id
data = project.drop('customerID', axis=1)
data = cleaning(data)
x = data.drop(columns =  ['Churn'])
y = data[['Churn']]
plots(data = data)
```



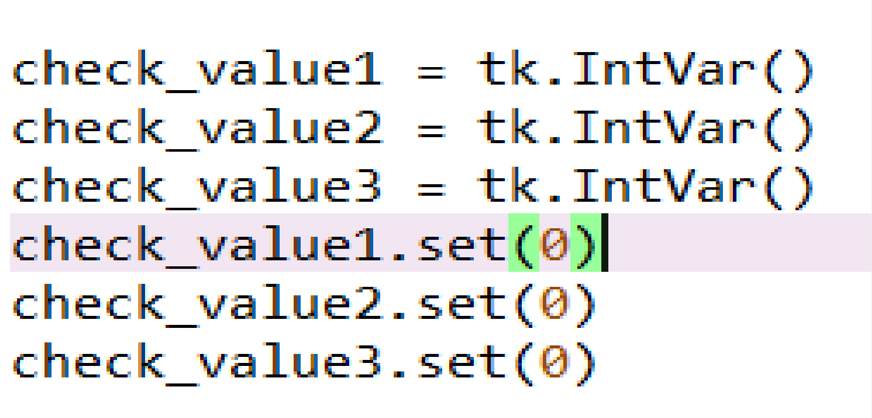
```
customerID          0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

- x_train, x_test, y_train, y_test =train_test_split (x, y, test_size=0.2, random_state=10)
- **We made function to take data from Entries to make a 1D array and return it.**



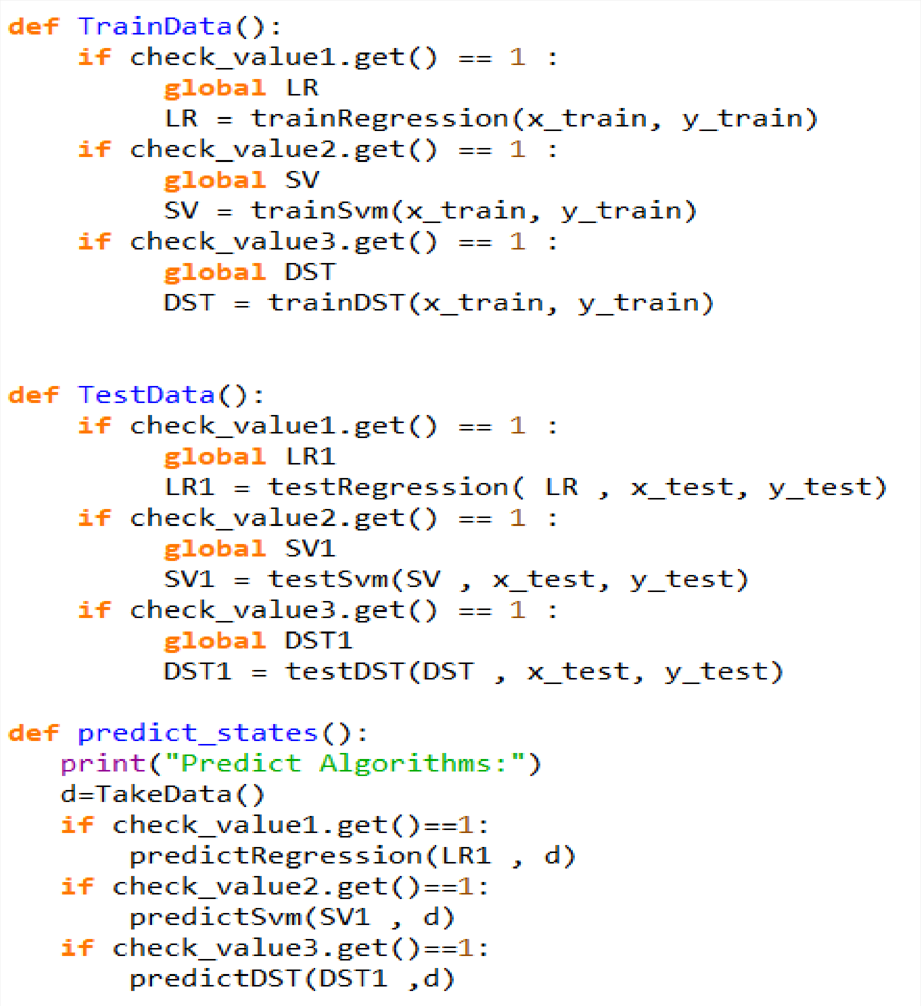
```
def TakeData():
    data=np.array([senior.get() ,part.get() ,depend.get() , tenur.get() , internet.get() ,
                 onlineS.get() , onlineB.get() , device.get() , tech.get() , streamT.get()
                   , streamM.get() , contract.get() , paper.get() , payment.get() ,
                   month.get() , total.get()])
    return data
```

- we made three IntVar () variables for every check button, set it = 0 by default, as when button selected the variable = 1 other variable = 0.



```
check_value1 = tk.IntVar()
check_value2 = tk.IntVar()
check_value3 = tk.IntVar()
check_value1.set(0)
check_value2.set(0)
check_value3.set(0)
```

- we made three functions for (train data, test data, predict state) for buttons (train, test, predict) in command attribute.



```
def TrainData():
    if check_value1.get() == 1 :
         global LR
         LR = trainRegression(x_train, y_train)
    if check_value2.get() == 1 :
         global SV
         SV = trainSvm(x_train, y_train)
    if check_value3.get() == 1 :
         global DST
         DST = trainDST(x_train, y_train)


def TestData():
    if check_value1.get() == 1 :
         global LR1
         LR1 = testRegression( LR , x_test, y_test)
    if check_value2.get() == 1 :
         global SV1
         SV1 = testSvm(SV , x_test, y_test)
    if check_value3.get() == 1 :
         global DST1
         DST1 = testDST(DST , x_test, y_test)

def predict_states():
   print("Predict Algorithms:")
   d=TakeData()
   if check_value1.get()==1:
       predictRegression(LR1 , d)
   if check_value2.get()==1:
       predictSvm(SV1 , d)
   if check_value3.get()==1:
       predictDST(DST1 ,d)
```

Thanks For Your Time 😇