INTELLIGENT CUSTOMER RETENTION : USING MACHINE LEARNING FOR ENHANCED PREDICTION OF TELECOM CUSTOMER CHURN

SUBMITTED BY

SATHIYA PRIYA.M(20326ER064)

SELVI.S(20326ER065)

SHOBHANA.S(20326ER066)

SHRUTHI.S(20326ER067)

ABSTRACT

Customer churn is often referred to as customer attrition, or customer defection which is the rate at which the customers are lost. Customer churn is a major problem and one of the most important concerns for large companies. Due to the direct effect on the revenues of the companies, especially in the telecom field, companies are seeking to develop means to predict potential customer to churn.

Looking at churn, different reasons trigger customers to terminate their contracts, for example better price offers, more interesting packages, bad service experiences or change of customers' personal situations.

Customer churn has become highly important for companies because of increasing competition among companies, increased importance of marketing strategies and conscious behaviour of customers in the recent years.

INTRODUCTION

1.1 Overview

Customers can easily trend toward alternative services. Companies must develop various strategies to prevent these possible trends, depending on the services they provide. During the estimation of possible churns, data from the previous churns might be used. An efficient churn predictive model benefits companies in many ways.

Early identification of customers likely to leave may help to build cost effective ways in marketing strategies. Customer retention campaigns might be limited to selected customers but it should cover most of the customer. Incorrect predictions could result in a company losing profits because of the discounts offered to continuous subscribers.

Telecommunication industry always suffers from a very high churn rates when one industry offers a better plan than the previous there is a high possibility of the customer churning from the present due to a better plan in such a scenario it is very difficult to avoid losses but through prediction we can keep it to a minimal level.

1.2 Purpose

Using these predictions, telecom operators can take proactive steps to retain customers, such as offering incentives or targeted marketing campaigns. This can help reduce churn rates and improve customer satisfaction, leading to increased revenue and market share.

Overall, intelligent customer retention using machine learning is a powerful tool for telecom operators looking to improve their customer retention strategies and stay competitive in an increasingly crowded market

Machine learning algorithms can be trained on large datasets of customer behavior, including usage patterns, call history, and demographics, to identify the factors that are most predictive of churn.

These factors can include things like call drop rates, data usage, and customer complaints. By analyzing these factors, machine learning models can identify customers who are at risk of churn and predict when they are likely to leave.

2. PROBLEM DEFINITION &DESIGN THINKING

2.1 Empathy Map

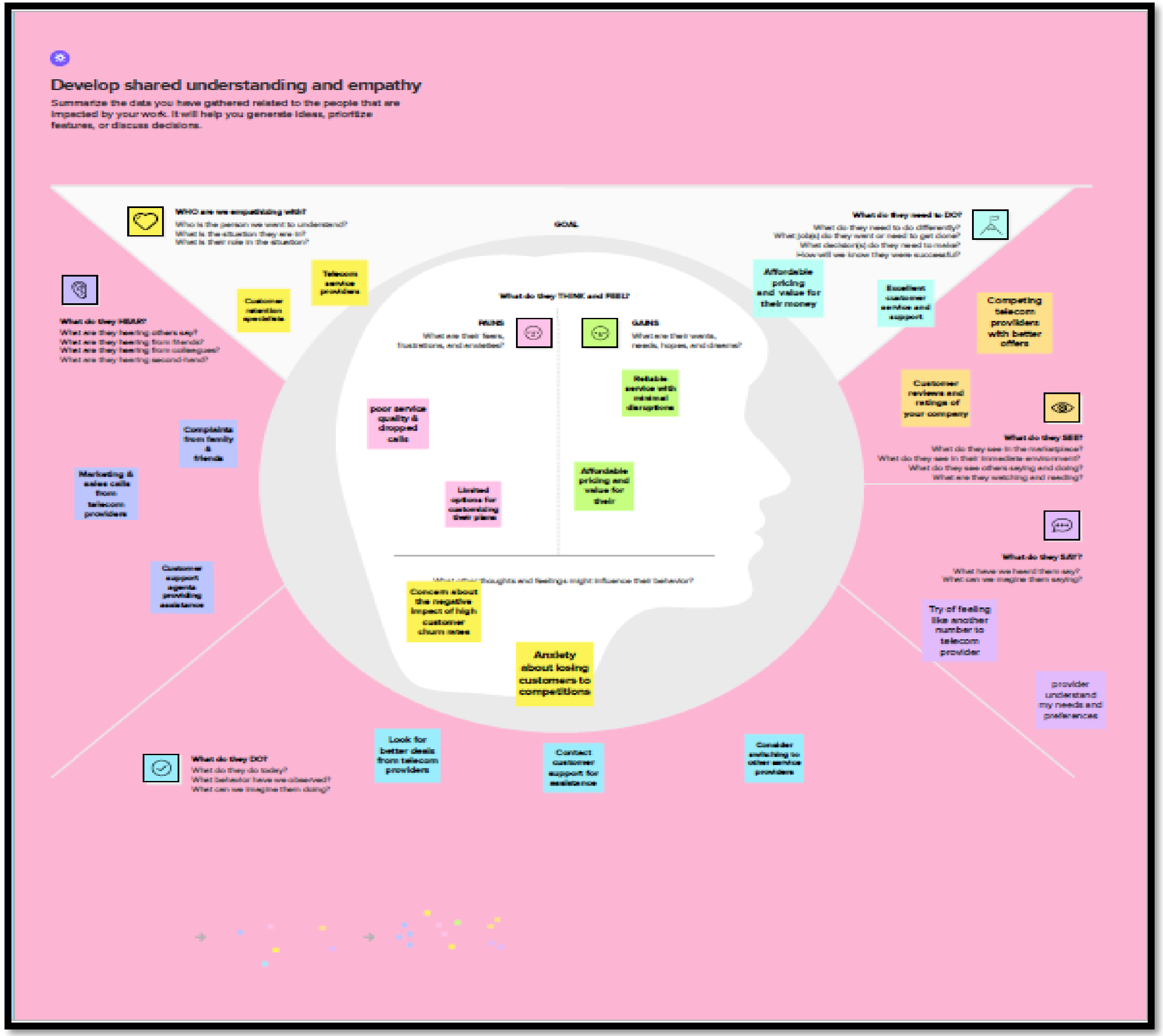
An empathy map is a visual tool that helps individuals or teams understand and empathize with a particular group of people they are designing a product, service or solution for. It is used to create a deeper understanding of customers' needs, wants, and behaviors by capturing their perspectives, emotions, and experiences. The empathy map is usually divided into four quadrants: Think, Feel, See, and Do, and each quadrant captures different aspects of the customer's experience. The purpose of the empathy map is to help the design team develop a more customer-centric solution that meets the customers' needs and expectations.


Template

Empathy map canvas

Use this framework to empathize with a customer, user, or any person who is affected by a team's work. Document and discuss your observations and note your assumptions to gain more empathy for the people you serve.

Originally created by Dave Gray at



Develop shared understanding and empathy
Summarize the data you have gathered related to the people that are impacted by your work. It will help you generate ideas, prioritize features, or discuss decisions.
WHO are we empathizing with?
Who is the person we want to understand?
What is the situation they are in?
What is their role in the situation?
GOAL
What do they need to DO?
What do they need to do differently?
What job(s) do they want or need to get done?
What decision(s) do they need to make?
How will we know they were successful?
Telecom service providers
Customer retention specialists
Affordable pricing and value for their money
Excellent customer service and support
Competing telecom providers with better offers
What do they HEAR?
What are they hearing others say?
What are they hearing from friends?
What are they hearing from colleagues?
What are they hearing second-hand?
What do they THINK and FEEL?
PAINS
What are their fears, frustrations, and anxieties?
GAINS
What are their wants, needs, hopes, and dreams?
poor service quality & dropped calls
Limited options for customizing their plans
Reliable service with minimal disruptions
Affordable pricing and value for their
Complaints from family & friends
Marketing & sales calls from telecom providers
Customer support agents providing assistance
Customer reviews and ratings of your company
What do they SEE?
What do they see in the marketplace?
What do they see in their immediate environment?
What do they see others saying and doing?
What are they watching and reading?
What other thoughts and feelings might influence their behavior?
Concern about the negative impact of high customer churn rates
Anxiety about losing customers to competitions
What do they SAY?
What have we heard them say?
What can we imagine them saying?
Try of feeling like another number to telecom provider
provider understand my needs and preferences
What do they DO?
What do they do today?
What behavior have we observed?
What can we imagine them doing?
Look for better deals from telecom providers
Contact customer support for assistance
Consider switching to other service providers

2.2 Ideation & Brainstorming Map

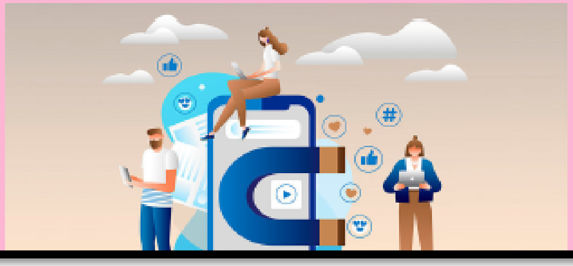
Brainstorming is a creative problem-solving technique that involves generating a large number of ideas and potential solutions to a problem or challenge in a short period of time. It is typically done in a group setting, where participants are encouraged to share their ideas freely and without criticism or judgment from others. The goal of brainstorming is to generate a diverse range of ideas, no matter how unconventional or unrealistic they may seem, and then to evaluate and refine them later on. Brainstorming is often used in fields such as business, marketing, and product development to generate innovative and effective solutions to complex problems.

Template

Brainstorm & idea prioritization

Intelligent Customer Retention:Using Machine Learning For Enhanced Prediction of Telecom Customer Churn

10 minutes to prepare
1 hour to collaborate
2-8 people recommended



Before collaborate

The main object of intelligent customer retention problem is to predict the potential telecom customer churn.

10 minutes

A **Team gathering**
Totally four participation are their in this session.we invite members through mural link and gathered in this session.

B **Set the goal**
The main object of intelligent customer retention problem is to predict the potential telecom customer churn.

C **Learn how to use the facilitation tools**
Facilitation tools can be very helpful for guiding discussions ,brainstorming sessions or decision-making processes

Open article →

1

Problem Statement

1.Customer churn is often referred to as customer attribute or customer defection which is the Rate at which the customers are lost.

5 minutes

2.Customer churn has become highly Important for companies because of increasing competition among companies.

3.The main object of intelligent customer retention problem is to predict the potential telecom customer churn.

4.This project will help the telecom companies to predict the number of customer that will leave a telecom service provide.

5.To identify probable churn customer machine learning algorithm will be applied and the result will be predict.

2

Brainstorm

Here some ideas.

10 minutes

Person 1

- KNN algorithm is used
- A fast internet connection needed
- COLAB is used
- Dataset is needed

Person 3

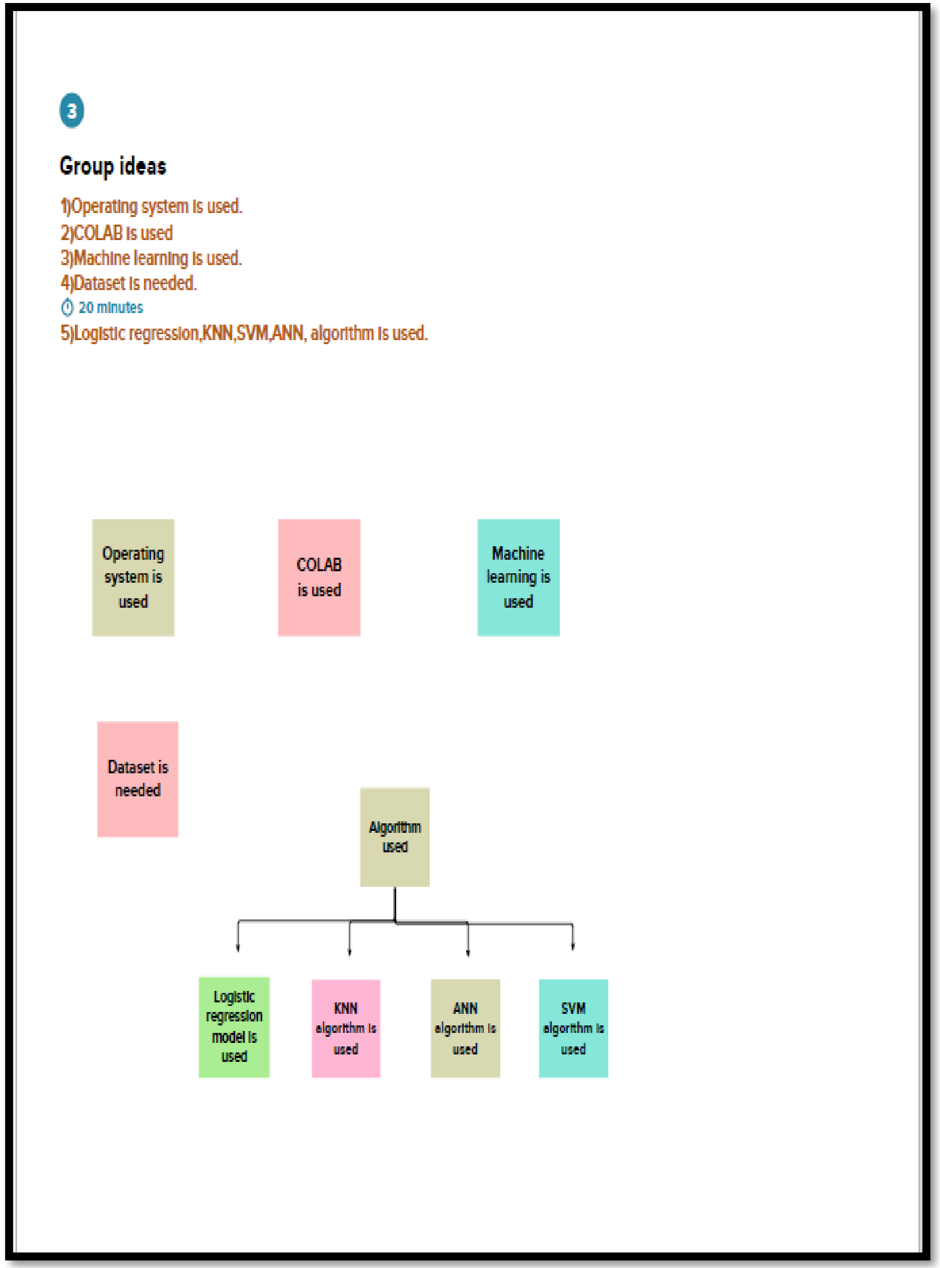
- SVM algorithm is used
- Storage is store large dataset
- Machine learning is used
- Libraries are Imported

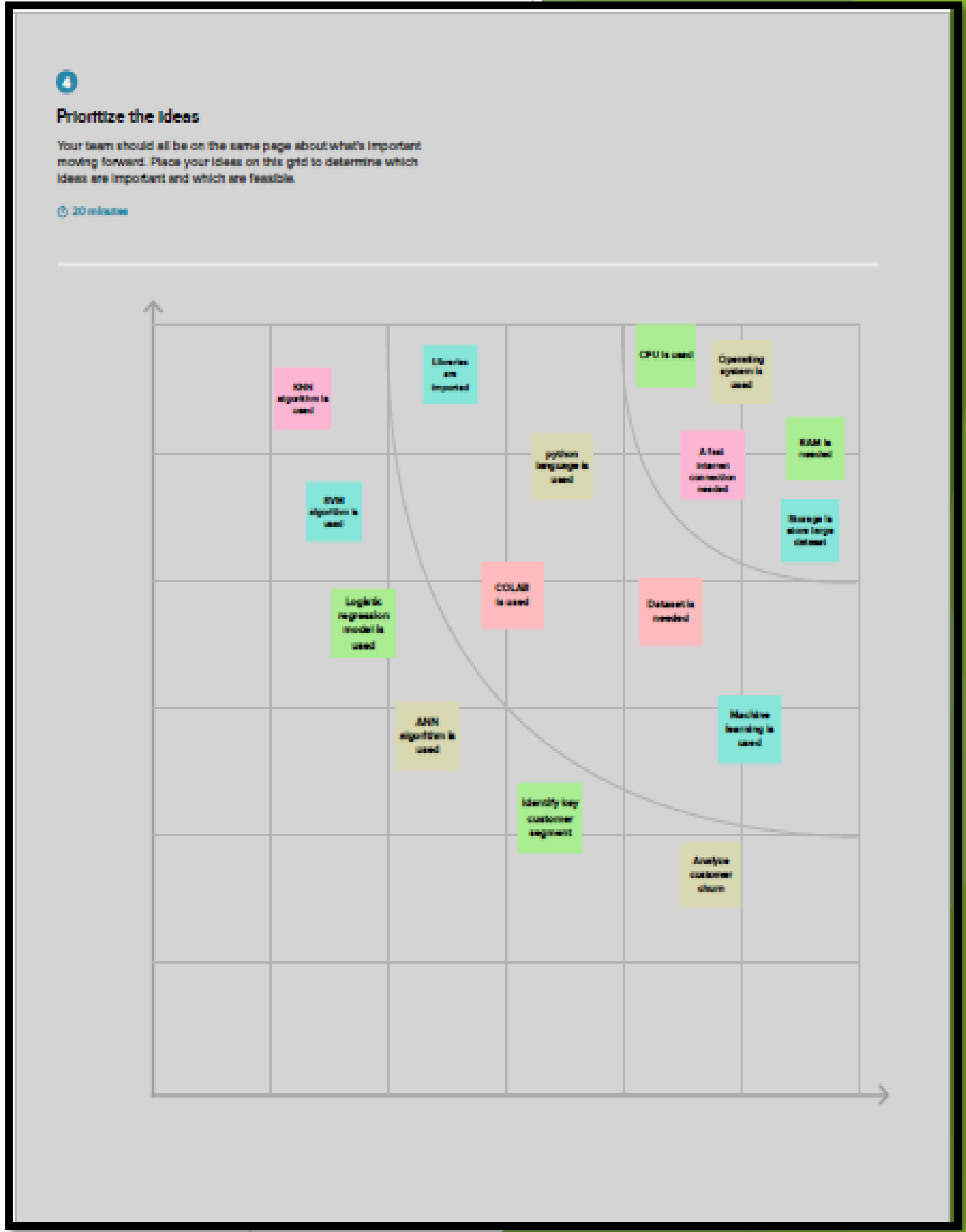
Person 2

- Logistic regression model is used
- CPU is used
- RAM is needed
- Identify key customer segment

Person 4

- ANN algorithm is used
- Operating system is used
- python language is used
- Analyze customer churn


3
Group ideas
1)Operating system is used.
2)COLAB is used
3)Machine learning is used.
4)Dataset is needed.
20 minutes
5)Logistic regression,KNN,SVM,ANN, algorithm is used.
Operating system is used
COLAB is used
Machine learning is used
Dataset is needed
Algorithm used
Logistic regression model is used
KNN algorithm is used
ANN algorithm is used
SVM algorithm is used


4
Prioritize the ideas
Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.
20 minutes
KNN algorithm is used
Libraries are imported
CPU is used
Operating system is used
python language is used
A fast internet connection needed
RAM is needed
SVM algorithm is used
Storage to store large dataset
COLAB is used
Logistic regression model is used
Dataset is needed
ANN algorithm is used
Machine learning is used
Identify key customer segment
Analyze customer churn

After collaborate

we can export the mural as pdf to share.It is helpful to getting information.

Quick add-ons

A **Share the mural**
Share a view link to the mural with stakeholders to keep them in the loop about the outcomes of the session.

B **Export the mural**
Export a copy of the mural as a PNG or PDF to attach to emails, include in slides, or save in your drive.

Keep moving forward

Strategy blueprint
Define the components of a new idea or strategy.
Open the template →

Customer experience journey map
Understand customer needs, motivations, and obstacles for an experience.
Open the template →

Strengths, weaknesses, opportunities & threats
Identify strengths, weaknesses, opportunities, and threats (SWOT) to develop a plan.
Open the template →

3. SCREEN LAYOUT

Data collection & preparation

Importing the libraries

```
#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
```

Read the Dataset

```
#import dataset
[5] data=pd.read_csv('/content/Telco_Cust_Churn.csv')
data
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	TechSupport
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	No	No
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...	Yes	No
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	No	No
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...	Yes	Yes
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...	No	No
...	...	...	...	...	...	...	...	...	...	...	...	...	...
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	Yes	DSL	Yes	...	Yes	Yes
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	Yes	Fiber optic	No	...	Yes	No

Data preparation

Handling missing values

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   object
 2   SeniorCitizen     7043 non-null   int64
 3   Partner           7043 non-null   object
 4   Dependents        7043 non-null   object
 5   tenure            7043 non-null   int64
 6   PhoneService      7043 non-null   object
 7   MultipleLines     7043 non-null   object
 8   InternetService   7043 non-null   object
 9   OnlineSecurity    7043 non-null   object
 10  OnlineBackup      7043 non-null   object
 11  DeviceProtection  7043 non-null   object
 12  TechSupport       7043 non-null   object
 13  StreamingTV       7043 non-null   object
 14  StreamingMovies   7043 non-null   object
 15  Contract          7043 non-null   object
 16  PaperlessBilling  7043 non-null   object
 17  PaymentMethod     7043 non-null   object
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object
 20  Churn             7043 non-null   object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

```
data.TotalCharges = pd.to_numeric(data.TotalCharges,errors='coerce')
data.isnull().any()
```

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

```
data["TotalCharges"].fillna(data["TotalCharges"].median(),inplace=True)
data.isnull().sum()
```

```
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
```

Handling categorical values

Label Encoding

```
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"])
```

```
data.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...
0	7590-VHVEG	0	0	1	0	1	0	1	0	0	...
1	5575-GNVDE	1	0	0	0	34	1	0	0	2	...
2	3668-QPYBK	1	0	0	0	2	1	0	0	2	...
3	7795-CFOCW	1	0	0	0	45	0	1	0	2	...
4	9237-HQITU	0	0	0	0	2	1	0	1	0	...

5 rows × 21 columns

Splitting the dataset into dependent and independent variable

```
[11] x=data.iloc[:,1:20].values
     y=data.iloc[:,20:21].values
```

```
x
```

```
array([[0.0000e+00, 0.0000e+00, 1.0000e+00, ..., 2.0000e+00, 2.9850e+01,
        2.9850e+01],
       [1.0000e+00, 0.0000e+00, 0.0000e+00, ..., 3.0000e+00, 5.6950e+01,
        1.8895e+03],
       [1.0000e+00, 0.0000e+00, 0.0000e+00, ..., 3.0000e+00, 5.3850e+01,
        1.0815e+02],
       ...,
       [0.0000e+00, 0.0000e+00, 1.0000e+00, ..., 2.0000e+00, 2.9600e+01,
        3.4645e+02],
       [1.0000e+00, 1.0000e+00, 1.0000e+00, ..., 3.0000e+00, 7.4400e+01,
        3.0660e+02],
       [1.0000e+00, 0.0000e+00, 0.0000e+00, ..., 0.0000e+00, 1.0565e+02,
        6.8445e+03]])
```

```
[13] y

array([[0],
       [0],
       [1],
       ...,
       [0],
       [1],
       [0]])
```

OneHot Encoding

```
[17] 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)
```

Handling Imbalance Data

```
[23] y_resample

     array([0, 0, 1, ..., 1, 1, 1])

[24] x.shape,x_resample.shape

     ((7043, 50), (10348, 50))

[25] y.shape,y_resample.shape

     ((7043, 1), (10348,))
```

```
[19] from imblearn.over_sampling import SMOTE

[20] smt=SMOTE()
     x_resample,y_resample = smt.fit_resample(x,y)

[22] x_resample

     array([[0.00000000e+00, 1.00000000e+00, 1.00000000e+00, ...,
             1.00000000e+00, 2.98500000e+01, 2.98500000e+01],
            [1.00000000e+00, 0.00000000e+00, 1.00000000e+00, ...,
             0.00000000e+00, 5.69500000e+01, 1.88950000e+03],
            [1.00000000e+00, 0.00000000e+00, 1.00000000e+00, ...,
             1.00000000e+00, 5.38500000e+01, 1.08150000e+02],
            ...,
            [0.00000000e+00, 1.00000000e+00, 1.00000000e+00, ...,
             9.45066722e-01, 3.51527467e+01, 1.00255386e+02],
            [0.00000000e+00, 1.00000000e+00, 1.00000000e+00, ...,
             1.00000000e+00, 1.00931551e+02, 5.07044271e+03],
            [0.00000000e+00, 1.00000000e+00, 1.00000000e+00, ...,
             1.00000000e+00, 4.44191959e+01, 8.04036347e+01]])
```

Exploratory data analysis
Descriptive statistical

✓ [26] data.describe()
0s

```
data.describe()
```

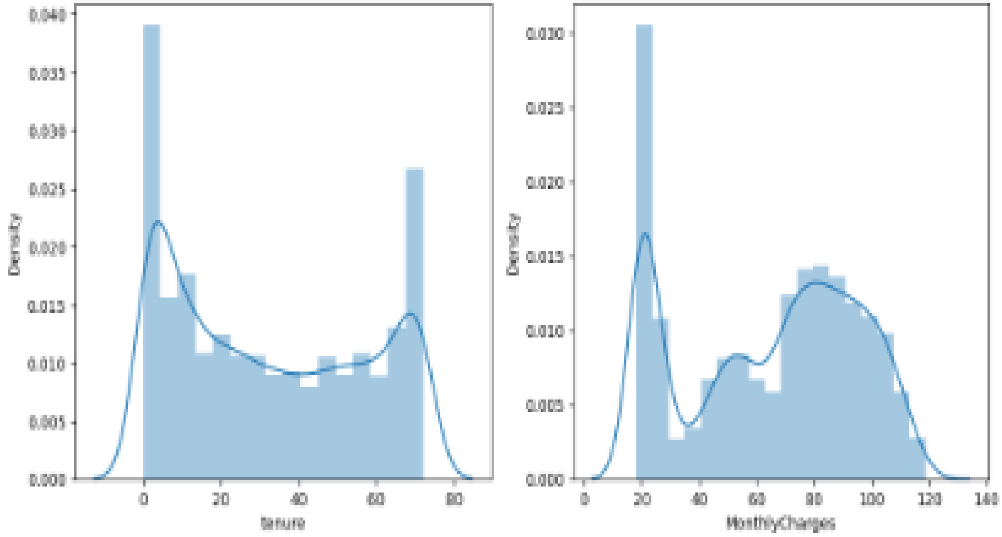
	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	In
count	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	
mean	0.504756	0.162147	0.483033	0.299588	32.371149	0.903166	0.940508	
std	0.500013	0.368612	0.499748	0.458110	24.559481	0.295752	0.948554	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	9.000000	1.000000	0.000000	
50%	1.000000	0.000000	0.000000	0.000000	29.000000	1.000000	1.000000	
75%	1.000000	0.000000	1.000000	1.000000	55.000000	1.000000	2.000000	
max	1.000000	1.000000	1.000000	1.000000	72.000000	1.000000	2.000000	

Visual analysis

Univariate analysis

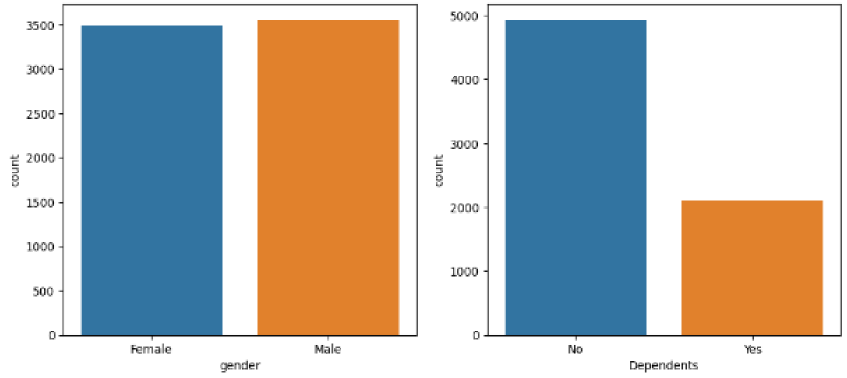
```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
sns.distplot(data["tenure"])
plt.subplot(1,2,2)
sns.distplot(data["MonthlyCharges"])
```

```
<ipython-input-28-3bd718de5fe4>:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data["tenure"])
<ipython-input-28-3bd718de5fe4>:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data["MonthlyCharges"])
<Axes: xlabel='MonthlyCharges', ylabel='Density'>
```



Countplot

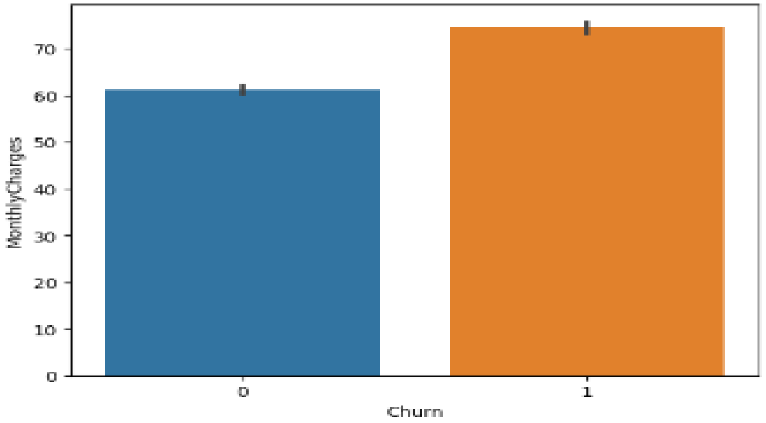
```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
#loading the dataset 'tips'
df=pd.read_csv("/content/Telco_Cust_Churn.csv")
#plotting the graph
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
sns.countplot(x='gender',data=df)
plt.subplot(1,2,2)
sns.countplot(x='Dependents',data=df)
plt.show()
```



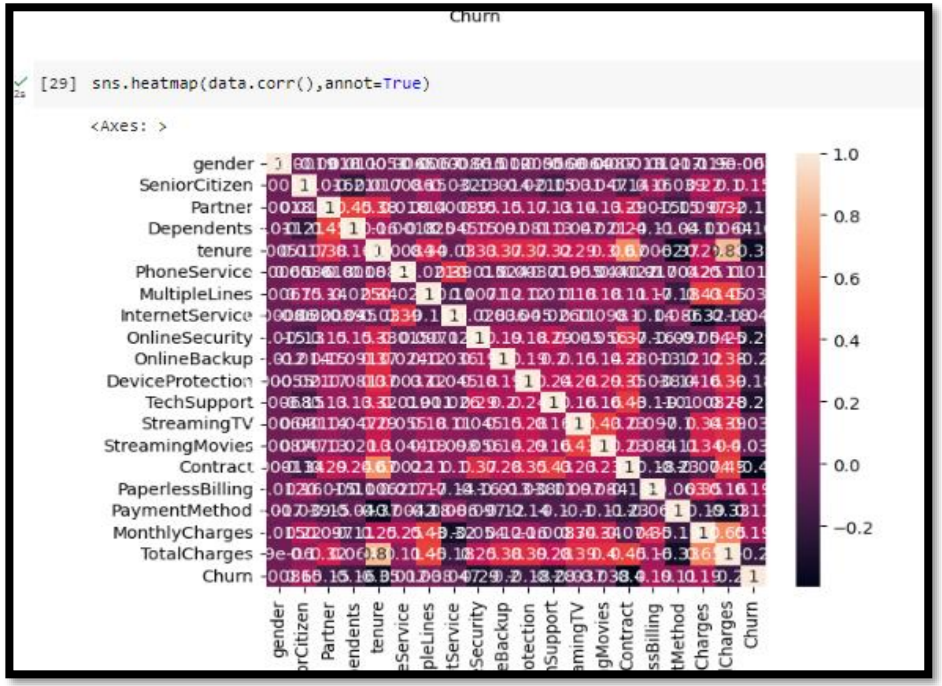
Bivariate anaysis

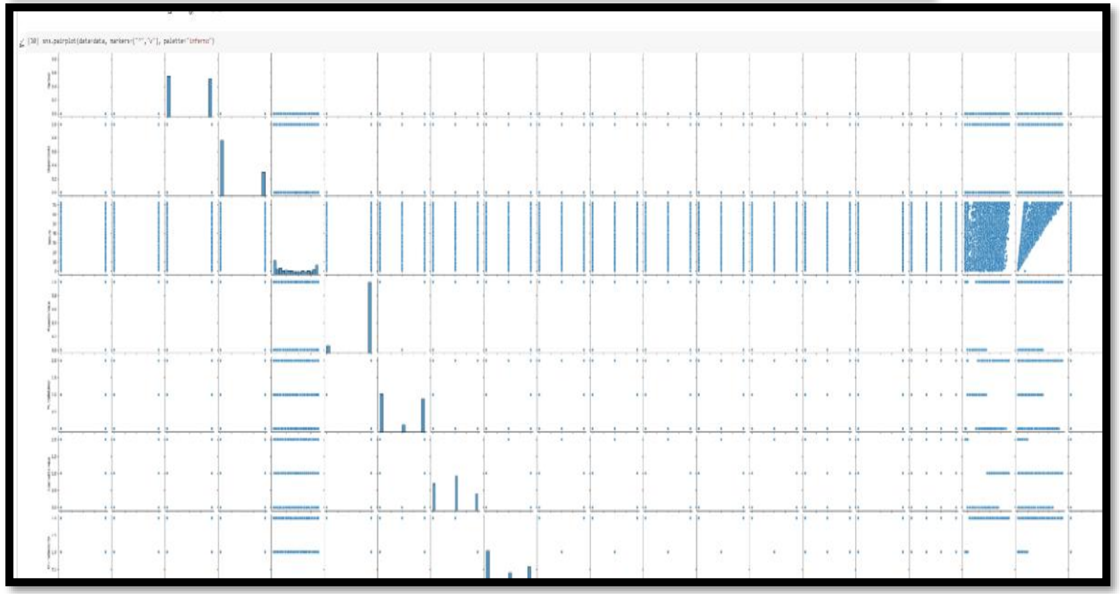
```python
sns.barplot(x="Churn",y="MonthlyCharges",data=data)
```

```
<Axes: xlabel='Churn', ylabel='MonthlyCharges'>
```



Multivariate analysis





Splitting data into train and test

```
[31] from sklearn.model_selection import 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)
```

Scaling the data

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

```
x_train.shape
```

```
(8278, 50)
```

Model building

Logistic Regression model

```
[36] #importig 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))
```

```
[35] #printing the train accuracy and test accuracy respectively
     logreg(x_train,x_test,y_train,y_test)
```

```
0.773737617782073
0.7739130434782608
***Logistic Regression***
Confusion_Matrix
[[753 280]
 [188 849]]
Classification Report
              precision    recall  f1-score   support

           0       0.80      0.73      0.76      1033
           1       0.75      0.82      0.78      1037

    accuracy                           0.77      2070
   macro avg       0.78      0.77      0.77      2070
weighted avg       0.78      0.77      0.77      2070
```

Decision Tree Model

```
[37] #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))
```

```
[38] #printing the train accuracy and test accuracy respectively
     decisionTree(x_train,x_test,y_train,y_test)
```

```
0.9981879681082387
0.7922705314009661
***Decision Tree***
confusion_Matrix
[[830 203]
 [227 810]]
Classification Report
              precision    recall  f1-score   support

           0       0.79      0.80      0.79      1033
           1       0.80      0.78      0.79      1037

    accuracy                           0.79      2070
   macro avg       0.79      0.79      0.79      2070
weighted avg       0.79      0.79      0.79      2070
```

Random Forest Model

```
[40] #importing and building the random forest 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))
```

```
[41] #printing the train accurancy and test accurancy and test accurancy respectively
     RandomForest(x_train,x_test,y_train,y_test)
```

```
0.9885237980188452
0.8120772946859903
***Random Forest
Confusion_Matrix
[[754 279]
 [110 927]]
Classification Report
              precision    recall  f1-score   support

           0       0.87      0.73      0.79      1033
           1       0.77      0.89      0.83      1037

    accuracy                           0.81      2070
   macro avg       0.82      0.81      0.81      2070
weighted avg       0.82      0.81      0.81      2070
```

KNN Model

```
[43] #importing and building the KNN model
     def KNN(x_tarin,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)
```

```
0.8549166465329789
0.7946859903381642
***KNN***
Confusion_Matrix
[[744 289]
 [136 901]]
Classification Report
              precision    recall  f1-score   support

           0       0.85      0.72      0.78      1033
           1       0.76      0.87      0.81      1037

    accuracy                           0.79      2070
   macro avg       0.80      0.79      0.79      2070
weighted avg       0.80      0.79      0.79      2070
```

SVM Model

```
#importing and building the random forest 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 tarin accuracy and test accuracy respectively
    SVM(x_train,x_test,y_train,y_test)
```

```
0.74438270113554
0.744927536231884
***Support Vector Machine***
Confusion_Matrix
[[664 369]
 [159 878]]
Classification Report
              precision    recall  f1-score   support

           0       0.81      0.64      0.72      1033
           1       0.70      0.85      0.77      1037

    accuracy                           0.74      2070
   macro avg       0.76      0.74      0.74      2070
weighted avg       0.76      0.74      0.74      2070
```

ANN Model

[36]
```
#importing the train Keras libraries and packages
import keras
from keras.models import Sequential
from keras.layers import Dense
```

[37]
```
#Initialising the ANN
classifier = Sequential()
```

[38]
```
#Adding the input layer and the first hidden layer
classifier.add(Dense(units=30,activation='relu',input_dim=40))
```

[39]
```
#Adding the second hidden layer
classifier.add(Dense(units=30,activation='relu'))
```

[40]
```
#Adding the output layer
classifier.add(Dense(units=1,activation='sigmoid'))
```

[41]
```
#Compiling theANN
classifier.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

```
#Fitting the ANN in the Training set
model_history = classifier.fit(x_train,y_train, batch_size=10,validation_split=0.33, epochs=200)
```

```
Epoch 1/200
555/555 [==============================] - 3s 3ms/step - loss: 0.5057 - accuracy: 0.7474 - val_loss: 0.4689 - val_accuracy: 0.7753
Epoch 2/200
555/555 [==============================] - 2s 3ms/step - loss: 0.4581 - accuracy: 0.7780 - val_loss: 0.4635 - val_accuracy: 0.7775
Epoch 3/200
555/555 [==============================] - 2s 3ms/step - loss: 0.4447 - accuracy: 0.7847 - val_loss: 0.4501 - val_accuracy: 0.7851
Epoch 4/200
555/555 [==============================] - 2s 4ms/step - loss: 0.4334 - accuracy: 0.7955 - val_loss: 0.4468 - val_accuracy: 0.7888
Epoch 5/200
555/555 [==============================] - 1s 2ms/step - loss: 0.4245 - accuracy: 0.8022 - val_loss: 0.4416 - val_accuracy: 0.7925
Epoch 6/200
555/555 [==============================] - 1s 2ms/step - loss: 0.4125 - accuracy: 0.8125 - val_loss: 0.4385 - val_accuracy: 0.7950
Epoch 7/200
555/555 [==============================] - 1s 2ms/step - loss: 0.4061 - accuracy: 0.8166 - val_loss: 0.4362 - val_accuracy: 0.7972
Epoch 8/200
555/555 [==============================] - 1s 2ms/step - loss: 0.3974 - accuracy: 0.8175 - val_loss: 0.4297 - val_accuracy: 0.8045
Epoch 9/200
555/555 [==============================] - 2s 3ms/step - loss: 0.3884 - accuracy: 0.8265 - val_loss: 0.4260 - val_accuracy: 0.8045
```

```
555/555 [==============================] - 2s 3ms/step - loss: 0.1428 - accuracy: 0.9418 - val_loss: 0.8324 - val_accuracy: 0.8034
Epoch 192/200
555/555 [==============================] - 2s 3ms/step - loss: 0.1443 - accuracy: 0.9380 - val_loss: 0.8078 - val_accuracy: 0.8005
Epoch 193/200
555/555 [==============================] - 2s 3ms/step - loss: 0.1427 - accuracy: 0.9396 - val_loss: 0.8320 - val_accuracy: 0.8108
Epoch 194/200
555/555 [==============================] - 2s 3ms/step - loss: 0.1447 - accuracy: 0.9371 - val_loss: 0.7941 - val_accuracy: 0.8042
Epoch 195/200
555/555 [==============================] - 1s 2ms/step - loss: 0.1434 - accuracy: 0.9380 - val_loss: 0.8497 - val_accuracy: 0.7983
Epoch 196/200
555/555 [==============================] - 1s 2ms/step - loss: 0.1415 - accuracy: 0.9403 - val_loss: 0.8395 - val_accuracy: 0.8078
Epoch 197/200
555/555 [==============================] - 1s 2ms/step - loss: 0.1479 - accuracy: 0.9374 - val_loss: 0.8054 - val_accuracy: 0.8100
Epoch 198/200
555/555 [==============================] - 2s 3ms/step - loss: 0.1396 - accuracy: 0.9391 - val_loss: 0.8340 - val_accuracy: 0.8078
Epoch 199/200
555/555 [==============================] - 2s 3ms/step - loss: 0.1474 - accuracy: 0.9382 - val_loss: 0.8248 - val_accuracy: 0.8023
Epoch 200/200
555/555 [==============================] - 2s 3ms/step - loss: 0.1430 - accuracy: 0.9421 - val_loss: 0.8175 - val_accuracy: 0.8108
```

```
[43] ann_pred =(ann_pred>0.5)
     ann_pred
```

```
65/65 [==============================] - 0s 1ms/step
array([[False],
       [False],
       [ True],
       ...,
       [False],
       [ True],
       [ True]])
```

```
print(accuracy_score(ann_pred,y_test))
print("***ANN Model***")
print("Confusion_Matrix")
print(confusion_matrix(y_test,ann_pred))
print("Classiification Report")
print(classification_report(y_test,ann_pred))
```

```
0.8004830917874396
***ANN Model***
Confusion_Matrix
[[811 222]
 [191 846]]
Classiification Report
              precision    recall  f1-score   support

           0       0.81      0.79      0.80      1033
           1       0.79      0.82      0.80      1037

    accuracy                           0.80      2070
   macro avg       0.80      0.80      0.80      2070
weighted avg       0.80      0.80      0.80      2070
```

Testing the model

```
[45]
#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)
```

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

```
[46]
#testing on random input values
dtc = DecisionTreeClassifier(criterion="entropy",random_state=0)
dtc.fit(x_train,y_train)
print("Predicting on random input")
dtc_pred_own = dtc.predict(sc.transform([[0,0,1,1,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,1,0,0,1,1,0,0,456,1,0,3425,4567]]))
print("output is: ",dtc_pred_own)
```

```
Predicting on random input
output is:  [1]
```

```
[47]
#testing on random input values
rf = RandomForestClassifier(criterion="entropy",n_estimators=10,random_state=0)
rf.fit(x_train,y_train)
print("Prediction 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)
```

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

```
[48] #testing on random input values
     svc = SVC(kernel = "linear")
     svc.fit(x_train,y_train)
     print("Prediction 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)

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

```
[49] #testing on random input values
     knn = KNeighborsClassifier()
     knn.fit(x_train,y_train)
     print("Prediction 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)

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

```
[50] #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)

     predicting on random input
     1/1 [==============================] - 0s 38ms/step
     [[0.]]
     output is: [[False]]
```

Performance testing & hyperparameter tuning

Compare the model

```
[51] 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)
           1       0.75      0.89      0.82      1037

    accuracy                           0.80      2070
   macro avg       0.81      0.80      0.80      2070
weighted avg       0.81      0.80      0.80      2070

--------------------------------------------------------------------------
0.7742208262865427
0.7695652173913043
***Logistic Regression***
Confusion_Matrix
[[755 278]
 [199 838]]
Classification Report
              precision    recall  f1-score   support

           0       0.79      0.73      0.76      1033
           1       0.75      0.81      0.78      1037

    accuracy                           0.77      2070
   macro avg       0.77      0.77      0.77      2070
weighted avg       0.77      0.77      0.77      2070

--------------------------------------------------------------------------
```

```
...
0.7742208262865427
0.7695652173913043
***Logistic Regression***
Confusion_Matrix
[[755 278]
 [199 838]]
Classification Report
              precision    recall  f1-score   support

           0       0.79      0.73      0.76      1033
           1       0.75      0.81      0.78      1037

    accuracy                           0.77      2070
   macro avg       0.77      0.77      0.77      2070
weighted avg       0.77      0.77      0.77      2070

-----------------------------------------------------------------
0.9981879681082387
0.7801932367149759
***Decision Tree***
confusion_Matrix
[[657 376]
 [ 79 958]]
Classification Report
              precision    recall  f1-score   support

           0       0.89      0.64      0.74      1033
           1       0.72      0.92      0.81      1037

    accuracy                           0.78      2070
   macro avg       0.81      0.78      0.78      2070
weighted avg       0.81      0.78      0.78      2070

-----------------------------------------------------------------
```

```
...
0.9876781831360232
0.7710144927536232
***Random Forest
Confusion_Matrix
[[614 419]
 [ 55 982]]
Classification Report
              precision    recall  f1-score   support

           0       0.92      0.59      0.72      1033
           1       0.70      0.95      0.81      1037

    accuracy                           0.77      2070
   macro avg       0.81      0.77      0.76      2070
weighted avg       0.81      0.77      0.76      2070

--------------------------------------------------------------------
0.747281952162358
0.7396135265700483
***Support Vector Machine***
Confusion_Matrix
[[664 369]
 [170 867]]
Classification Report
              precision    recall  f1-score   support

           0       0.80      0.64      0.71      1033
           1       0.70      0.84      0.76      1037

    accuracy                           0.74      2070
   macro avg       0.75      0.74      0.74      2070
weighted avg       0.75      0.74      0.74      2070
```

```
0.8558830635419183
0.7985507246376812
***KNN***
Confusion_Matrix
[[729 304]
 [113 924]]
Classification Report
              precision    recall  f1-score   support

           0       0.87      0.71      0.78      1033
           1       0.75      0.89      0.82      1037

    accuracy                           0.80      2070
   macro avg       0.81      0.80      0.80      2070
weighted avg       0.81      0.80      0.80      2070
```

```
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))
```

```
0.8004830917874396
***ANN Model***
Confusion_Matrix
[[811 222]
 [191 846]]
Classification Report
              precision    recall  f1-score   support

           0       0.81      0.79      0.80      1033
           1       0.79      0.82      0.80      1037

    accuracy                           0.80      2070
   macro avg       0.80      0.80      0.80      2070
weighted avg       0.80      0.80      0.80      2070
```

Comparing model accuracy before & after applying hyperparameter tuning

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# define the random forest classifier model
model = RandomForestClassifier()

# define the hyperparameters to tune
params = {
    'n_estimators': [50, 100, 200],
    'max_depth': [5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# perform grid search cross-validation to find the best hyperparameters
grid_search = GridSearchCV(model, params, cv=5)
grid_search.fit(x_train, y_train)

# print the best hyperparameters found by grid search
print("Best hyperparameters:", grid_search.best_params_)

# get the best model from grid search
model = grid_search.best_estimator_

# evaluate the model on the training set
y_rf = model.predict(x_train)
print("Training set accuracy:", accuracy_score(y_rf, y_train))
```

```python
# evaluate the model on the training set
y_rf = model.predict(x_train)
print("Training set accuracy:", accuracy_score(y_rf, y_train))

# evaluate the model on the test set
yPred_rfcv = model.predict(x_test)
print("Test set accuracy:", accuracy_score(yPred_rfcv, y_test))

# print the confusion matrix and classification report for the test set
print("**Random Forest after Hyperparameter tuning**")
print("Confusion Matrix")
print(confusion_matrix(y_test, yPred_rfcv))
print("Classification Report")
print(classification_report(y_test, yPred_rfcv))

# use the model to predict on a new 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)
```

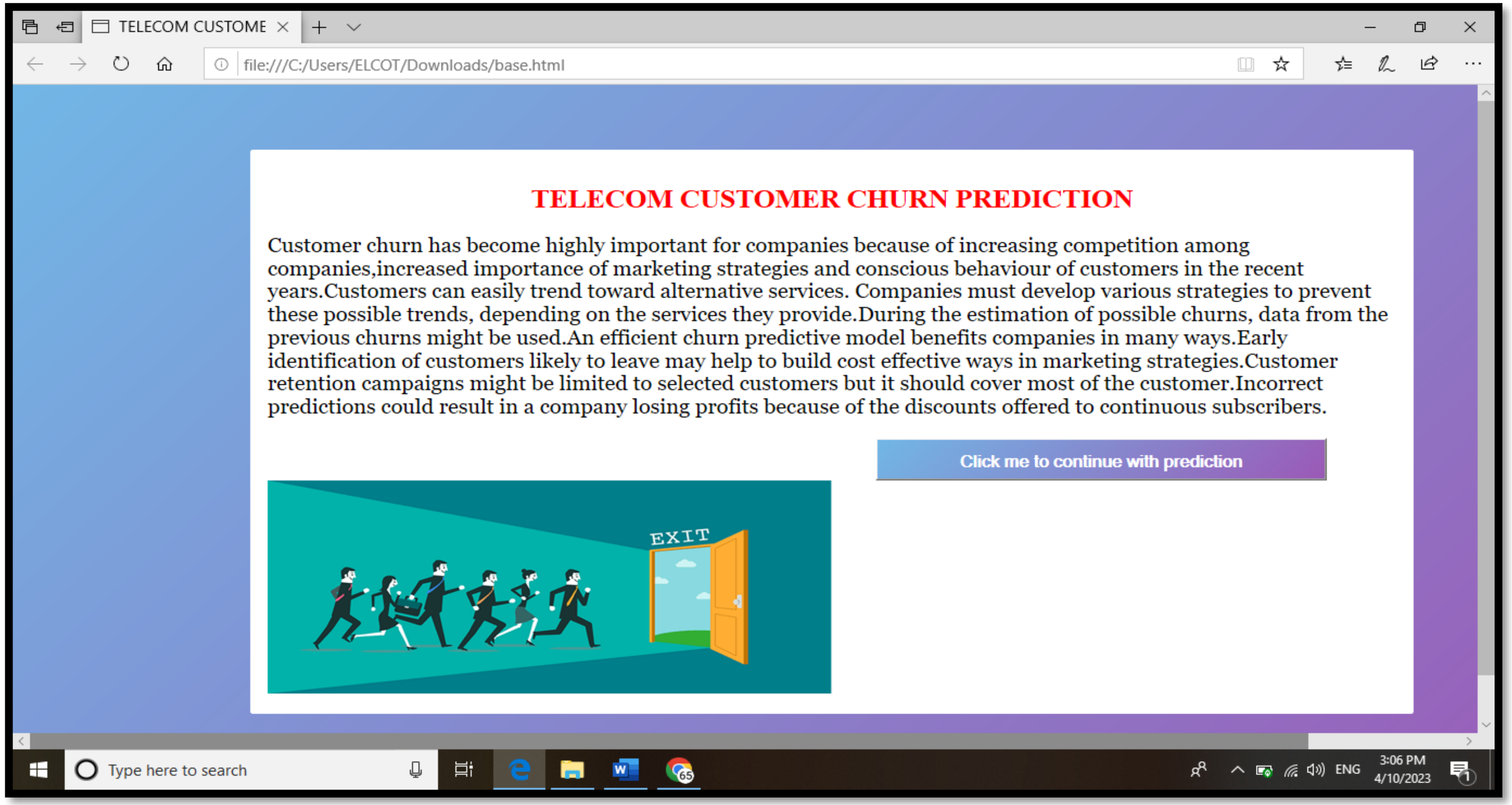
```
Best hyperparameters: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 100}
Training set accuracy: 0.930055568978014
Test set accuracy: 0.7613526570048309
**Random Forest after Hyperparameter tuning**
Confusion Matrix
[[589 444]
 [ 50 987]]
Classification Report
              precision    recall  f1-score   support

           0       0.92      0.57      0.70      1033
           1       0.69      0.95      0.80      1037

    accuracy                           0.76      2070
   macro avg       0.81      0.76      0.75      2070
weighted avg       0.81      0.76      0.75      2070

Output is: [0]
```

Model Deployment

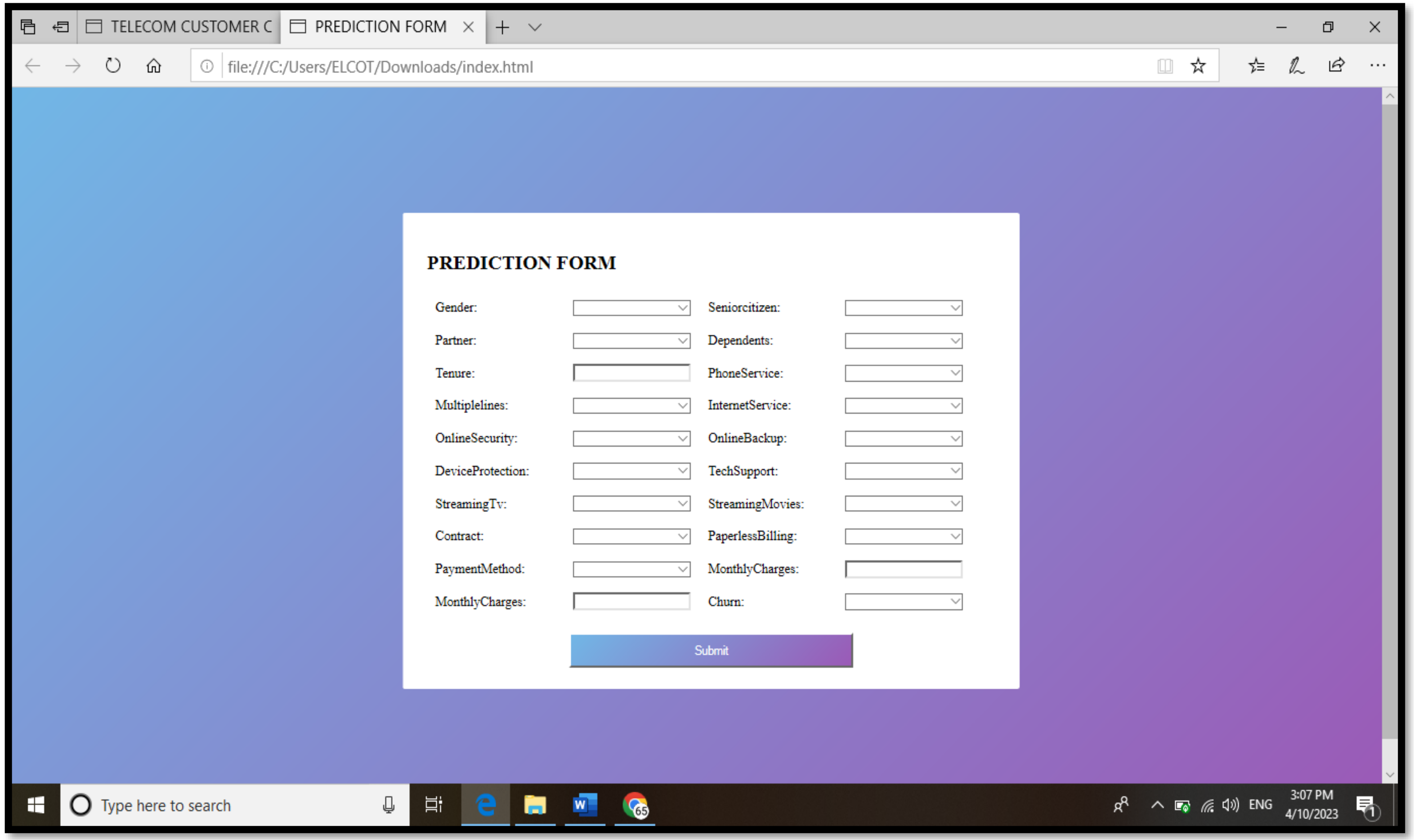
Save the best model

```
classifier.save("telcom_churn.h5")
```

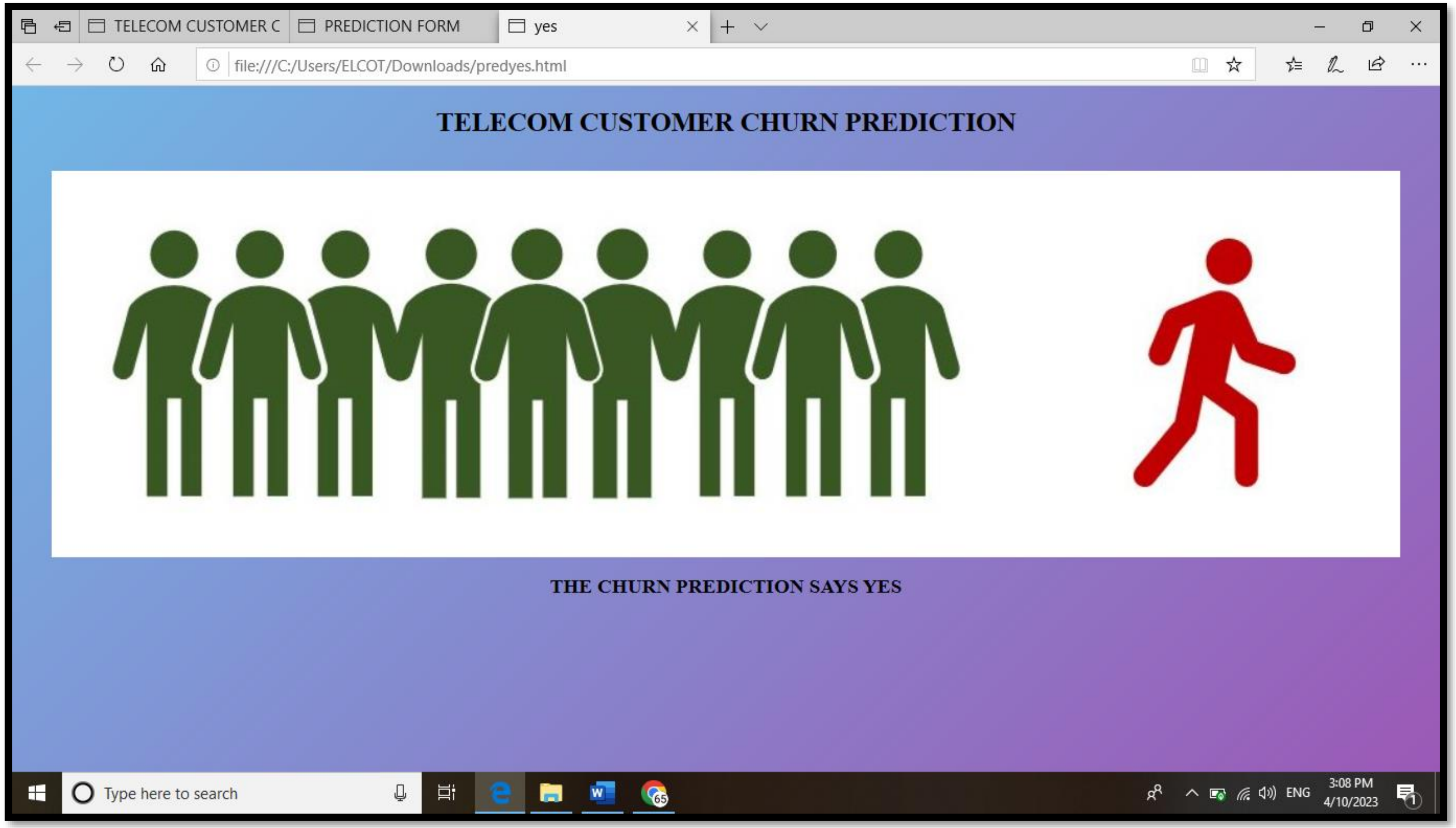
3. RESULT
Base.html



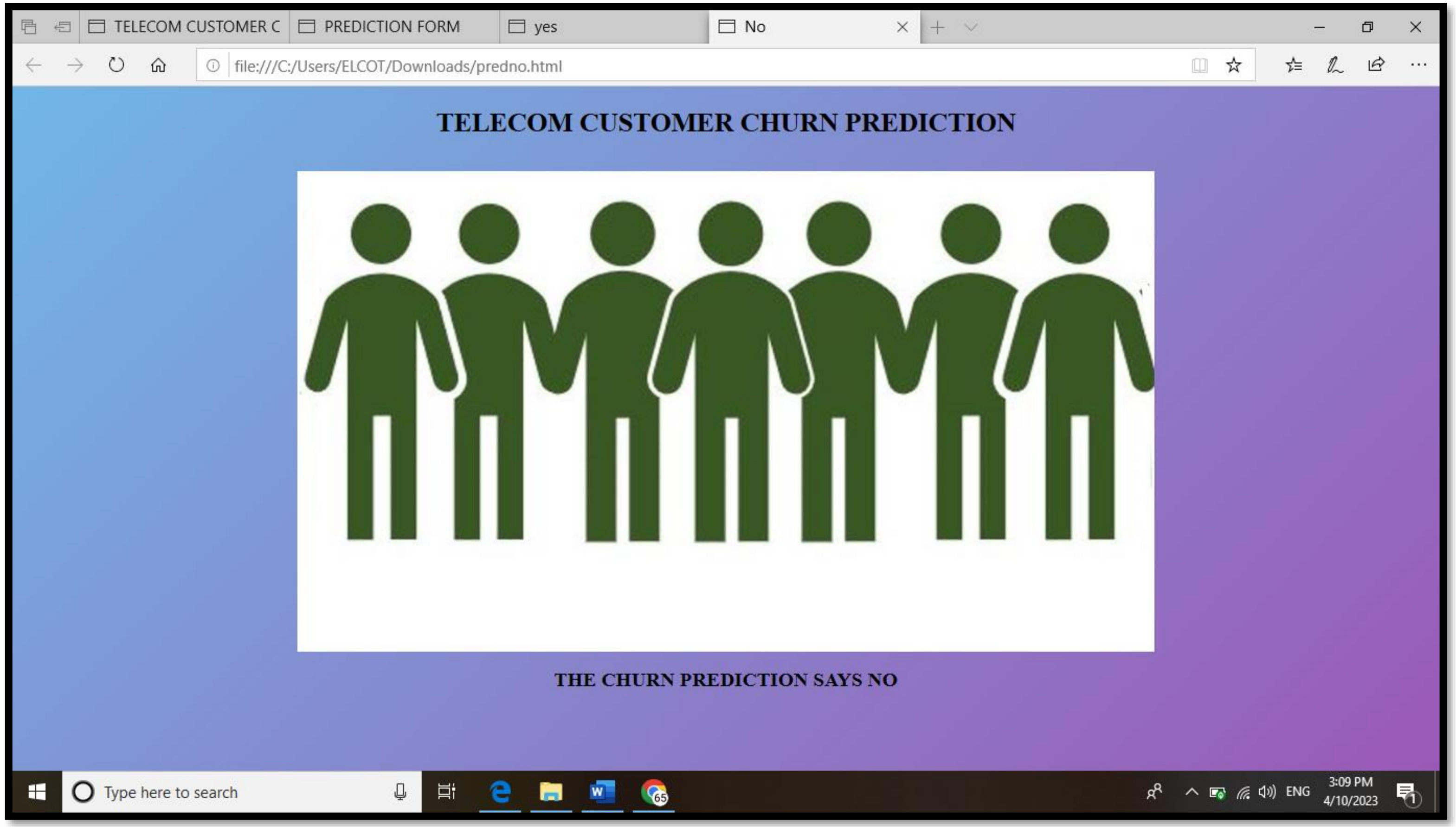
Index.html


TELECOM CUSTOMER C
PREDICTION FORM
file:///C:/Users/ELCOT/Downloads/index.html
PREDICTION FORM
Gender:
Seniorcitizen:
Partner:
Dependents:
Tenure:
PhoneService:
Multiplelines:
InternetService:
OnlineSecurity:
OnlineBackup:
DeviceProtection:
TechSupport:
StreamingTv:
StreamingMovies:
Contract:
PaperlessBilling:
PaymentMethod:
MonthlyCharges:
MonthlyCharges:
Churn:
Submit
Type here to search
ENG
3:07 PM
4/10/2023

Predyes.html



Predno.html



TELECOM CUSTOMER C
PREDICTION FORM
yes
No
file:///C:/Users/ELCOT/Downloads/predno.html
TELECOM CUSTOMER CHURN PREDICTION
THE CHURN PREDICTION SAYS NO
Type here to search
ENG
3:09 PM
4/10/2023

4. ADVANTAGES

Better prediction accuracy: Machine learning algorithms can analyze large amounts of data and identify patterns that may not be evident to humans. By using this data, they can predict which customers are more likely to churn, thus allowing telecom companies to take proactive measures to retain those customers.

Cost savings: By predicting which customers are likely to churn, telecom companies can take steps to retain those customers, which can be less expensive than acquiring new customers.

Customization: Machine learning algorithms can help telecom companies tailor their retention strategies to individual customers based on their unique behaviors and preferences.

Improved customer experience: By using machine learning algorithms to identify and address customer issues, telecom companies can improve their overall customer experience and satisfaction.

5. APPLICATION

Predicting customer churn: Telecom companies can use machine learning algorithms to analyze customer data and identify patterns that indicate a customer is at risk of churning. This can include factors such as usage patterns, call duration, and customer complaints.

Developing targeted retention strategies: Once at-risk customers have been identified, telecom companies can use machine learning to develop targeted retention strategies. These strategies can be customized to each customer based on their behavior, preferences, and past interactions with the company.

Improving customer service: Machine learning algorithms can be used to analyze customer service interactions and identify areas for improvement. By addressing customer issues proactively, telecom companies can improve customer satisfaction and reduce the likelihood of churn.

6. CONCLUSION

In conclusion, the use of machine learning for intelligent customer retention in the telecom industry has shown promising results in predicting and reducing customer churn. By analyzing large amounts of customer data, machine learning models can identify patterns and signals that indicate a customer is likely to churn, allowing telecom companies to proactively intervene with targeted retention strategies.

Moreover, the application of machine learning in customer retention has enabled telecom companies to personalize their retention efforts, tailoring offers and incentives to individual customers based on their unique needs and preferences. This approach has been shown to be more effective than one-size-fits-all retention strategies.

Overall, the use of machine learning in customer retention is a powerful tool for the telecom industry to enhance customer satisfaction, reduce churn, and increase revenue. As machine learning algorithms continue to improve and more data becomes available, the potential for intelligent customer retention will only grow.

7. FUTHURE SCOPE

There are several potential future enhancements for intelligent customer retention using machine learning in the telecom industry:

Incorporating more data sources: Telecom companies can enhance their machine learning models by incorporating additional data sources such as social media activity, device usage, and network performance. This can provide a more comprehensive view of the customer and improve the accuracy of churn prediction.

Real-time interventions: Machine learning models can enable telecom companies to intervene with personalized offers and incentives in real-time. This means that retention strategies can be applied as soon as a customer exhibits signals of churn, increasing the likelihood of retaining them.

Continual learning: Machine learning algorithms can continually learn and improve over time as more data becomes available. By constantly updating and refining their models, telecom companies can stay ahead of changing customer behavior and further improve their retention efforts.

Explainability: While machine learning models have shown impressive accuracy in predicting customer churn, they can be difficult to interpret and understand. Incorporating explainability features into these models can help telecom companies understand the drivers behind customer churn and develop more effective retention strategies.

Integration with other systems: Machine learning models can be integrated with other telecom systems such as customer relationship management (CRM) and billing systems to improve the effectiveness of retention strategies