```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Loading dataset

```
df = pd.read_csv('/content/customer_churn.csv')
df.sample(5)
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	Mul
1215	9897-KXHCM	Female	0	Yes	Yes	3	Yes	
1629	3533-UVMOM	Male	0	Yes	No	68	Yes	
5163	3472-OAOOR	Male	0	Yes	Yes	19	No	
3498	7730-IUTDZ	Male	0	No	No	43	Yes	
4686	8024-XNAFQ	Female	1	No	No	72	Yes	

5 rows × 21 columns

droping customerID column as it is of no use

```
#step 1:data exploration ie cust_id is useless
df.drop('customerID',axis='columns',inplace=True)
df.dtypes
```

	0
gender	object
SeniorCitizen	int64
Partner	object
Dependents	object
tenure	int64
PhoneService	object
MultipleLines	object
InternetService	object
OnlineSecurity	object
OnlineBackup	object
DeviceProtection	object
TechSupport	object
StreamingTV	object
StreamingMovies	object
Contract	object
PaperlessBilling	object
PaymentMethod	object

PaymentMethod	object
MonthlyCharges	float64
TotalCharges	object
Churn	object

```
#TotalCharges         in str
df.TotalCharges.values
```

```
array(['29.85', '1889.5', '108.15', ..., '346.45', '306.6', '6844.5'],
      dtype=object)
```

```
df.MonthlyCharges.values#numbers
```

```
array([ 29.85,  56.95,  53.85, ...,  29.6 ,  74.4 , 105.65])
```

```
# pd.to_numeric(df.TotalCharges)
```

converting TotalCharges to float as it is in object type

```
#to tackle spaces in TotalCharges
pd.to_numeric(df.TotalCharges,errors='coerce').isnull()   #put na if space in that col
```

	TotalCharges
0	False
1	False
2	False
3	False
4	False
...	...
7038	False
7039	False
7040	False
7041	False
7042	False

7043 rows × 1 columns

```
df[pd.to_numeric(df.TotalCharges,errors='coerce').isnull()]   #tota charges are nulls df
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
488	Female	0	Yes	Yes	0	No	No phone service
753	Male	0	No	Yes	0	Yes	No
936	Female	0	Yes	Yes	0	Yes	No
1082	Male	0	Yes	Yes	0	Yes	Yes
1340	Female	0	Yes	Yes	0	No	No phone service
3331	Male	0	Yes	Yes	0	Yes	No
3826	Male	0	Yes	Yes	0	Yes	Yes
4380	Female	0	Yes	Yes	0	Yes	No
5218	Male	0	Yes	Yes	0	Yes	No
6670	Female	0	Yes	Yes	0	Yes	Yes
6754	Male	0	No	Yes	0	Yes	Yes

```
df[pd.to_numeric(df.TotalCharges,errors='coerce').isnull()].shape
```

```
(11, 20)
```

```
df.shape
```

```
(7043, 20)
```

```
df.iloc[488].TotalCharges #iloc is like indexing in array  (488 row)
```

```
#drop 11 rows
df1 = df[df.TotalCharges!=' ']
df1.shape
```

```
(7032, 20)
```

```
df1.TotalCharges=pd.to_numeric(df1.TotalCharges)
```

```
<ipython-input-83-01816c9a1a9f>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user
```

```
df1.TotalCharges=pd.to_numeric(df1.TotalCharges)
```

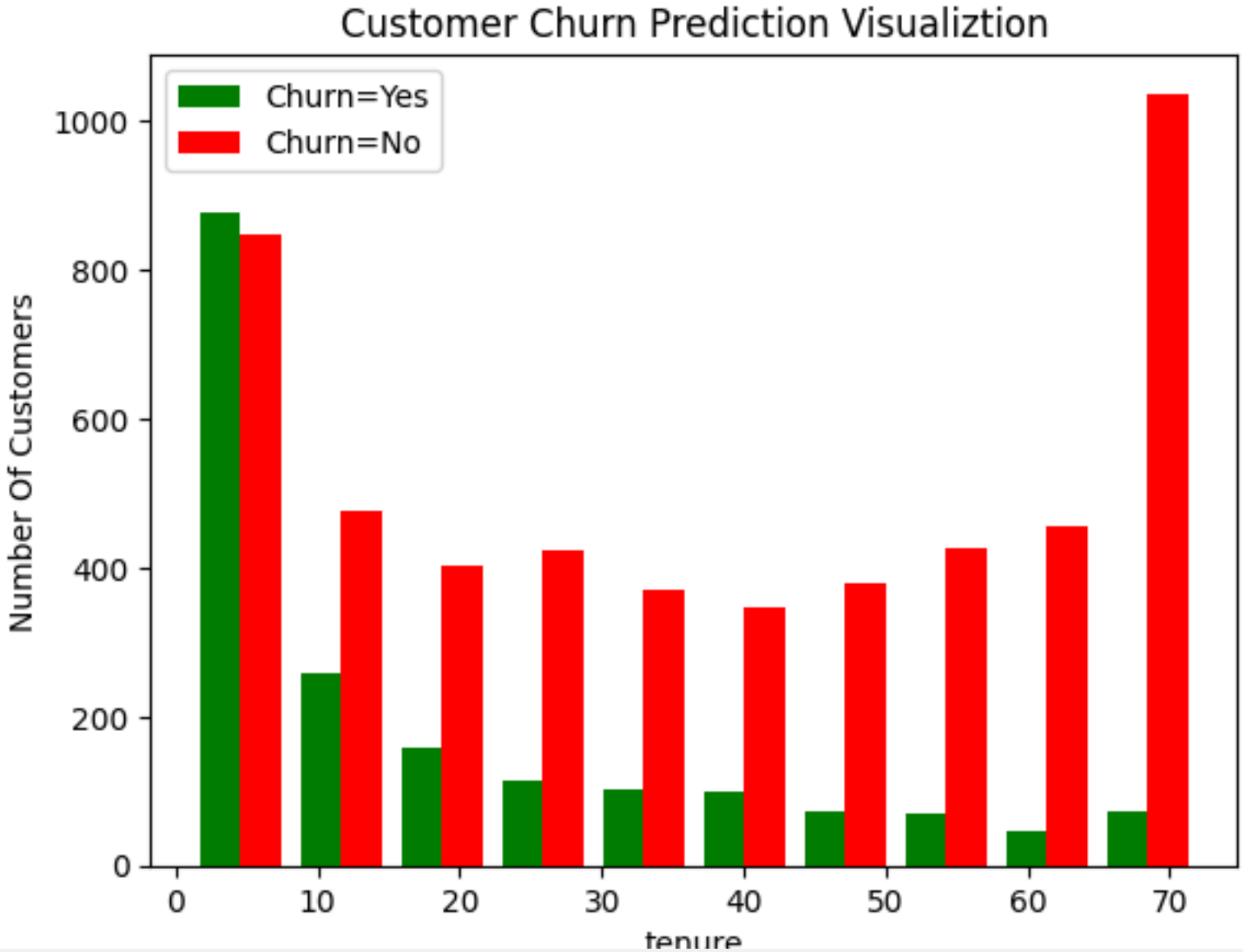
```
df1.TotalCharges.dtypes
```

```
dtype('float64')
```

TENURE - HOW CUSTOMER LOYAL IS

```
tenure_churn_no = df1[df1.Churn == 'No'].tenure #not leaving
tenure_churn_Yes = df1[df1.Churn == 'Yes'].tenure # leaving IN MONTHS


plt.xlabel("tenure")
plt.ylabel("Number Of Customers")
plt.title("Customer Churn Prediction Visualiztion")
plt.hist([tenure_churn_Yes,tenure_churn_no],color=['green','red'],label=['Churn=Yes','Churn
plt.legend()
#AROUND 1000CUST ARE NOT LEAVAING WHERE TENURE =70
```

```
<matplotlib.legend.Legend at 0x7a8e5d346200>
```



Start coding or generate with AI.

```
def print_unique_col_values(df):
  for column in df:
    if df[column].dtypes=='object':
      print(f'{column}: {df[column].unique()}')
```

```
print_unique_col_values(df1)
```

```
gender: ['Female' 'Male']
Partner: ['Yes' 'No']
Dependents: ['No' 'Yes']
PhoneService: ['No' 'Yes']
MultipleLines: ['No phone service' 'No' 'Yes']
InternetService: ['DSL' 'Fiber optic' 'No']
OnlineSecurity: ['No' 'Yes' 'No internet service']
OnlineBackup: ['Yes' 'No' 'No internet service']
DeviceProtection: ['No' 'Yes' 'No internet service']
TechSupport: ['No' 'Yes' 'No internet service']
StreamingTV: ['No' 'Yes' 'No internet service']
StreamingMovies: ['No' 'Yes' 'No internet service']
Contract: ['Month-to-month' 'One year' 'Two year']
PaperlessBilling: ['Yes' 'No']
PaymentMethod: ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
Churn: ['No' 'Yes']
```

```
df1.replace('No internet service','No',inplace=True)
df1.replace('No phone service','No',inplace=True)
print_unique_col_values(df1)
```

```
gender: ['Female' 'Male']
Partner: ['Yes' 'No']
Dependents: ['No' 'Yes']
PhoneService: ['No' 'Yes']
MultipleLines: ['No' 'Yes']
InternetService: ['DSL' 'Fiber optic' 'No']
OnlineSecurity: ['No' 'Yes']
OnlineBackup: ['Yes' 'No']
DeviceProtection: ['No' 'Yes']
TechSupport: ['No' 'Yes']
StreamingTV: ['No' 'Yes']
StreamingMovies: ['No' 'Yes']
Contract: ['Month-to-month' 'One year' 'Two year']
PaperlessBilling: ['Yes' 'No']
PaymentMethod: ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
Churn: ['No' 'Yes']
<ipython-input-88-911fb1bda1c4>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user
  df1.replace('No internet service','No',inplace=True)
<ipython-input-88-911fb1bda1c4>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user
  df1.replace('No phone service','No',inplace=True)
```

```
yes_no_columns = ['Partner','Dependents','PhoneService','MultipleLines','OnlineSecurity','O

for col in yes_no_columns:
  df1[col].replace({'Yes': 1,'No': 0},inplace=True)
```

```
<ipython-input-89-0cbc454eee20>:4: FutureWarning: A value is trying to be set on a copy
The behavior will change in pandas 3.0. This inplace method will never work because the

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({co


  df1[col].replace({'Yes': 1,'No': 0},inplace=True)
<ipython-input-89-0cbc454eee20>:4: FutureWarning: Downcasting behavior in `replace` is
  df1[col].replace({'Yes': 1,'No': 0},inplace=True)
<ipython-input-89-0cbc454eee20>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user
  df1[col].replace({'Yes': 1,'No': 0},inplace=True)
```

```
df1
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
0	Female	0	1	0	1	0	0
1	Male	0	0	0	34	1	0
2	Male	0	0	0	2	1	0
3	Male	0	0	0	45	0	0
4	Female	0	0	0	2	1	0
...	...	...	...	...	...	...	...
7038	Male	0	1	1	24	1	1
7039	Female	0	1	1	72	1	1
7040	Female	0	1	1	11	0	0
7041	Male	1	1	0	4	1	1
7042	Male	0	0	0	66	1	0

7032 rows × 20 columns

Next steps: Generate code with df1 | View recommended plots | New interactive sheet

```
print_unique_col_values(df1)
```

```
gender: ['Female' 'Male']
InternetService: ['DSL' 'Fiber optic' 'No']
Contract: ['Month-to-month' 'One year' 'Two year']
PaymentMethod: ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
```

```
df1['gender'].replace({'Female':1,'Male':0},inplace=True)
```

```
<ipython-input-92-ba153b6b6960>:1: FutureWarning: A value is trying to be set on a copy
The behavior will change in pandas 3.0. This inplace method will never work because the

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({co


  df1['gender'].replace({'Female':1,'Male':0},inplace=True)
<ipython-input-92-ba153b6b6960>:1: FutureWarning: Downcasting behavior in `replace` is
  df1['gender'].replace({'Female':1,'Male':0},inplace=True)
<ipython-input-92-ba153b6b6960>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user
  df1['gender'].replace({'Female':1,'Male':0},inplace=True)
```

```
df1.gender.unique()
```

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

```
print_unique_col_values(df1)
```

```
InternetService: ['DSL' 'Fiber optic' 'No']
Contract: ['Month-to-month' 'One year' 'Two year']
PaymentMethod: ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
```

```
df2 = pd.get_dummies(data=df1,columns=['InternetService','Contract','PaymentMethod']) # ONE H
df2.columns

# df2
```

```
Index(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure',
       'PhoneService', 'MultipleLines', 'OnlineSecurity', 'OnlineBackup',
       'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
       'PaperlessBilling', 'MonthlyCharges', 'TotalCharges', 'Churn',
       'InternetService_DSL', 'InternetService_Fiber optic',
       'InternetService_No', 'Contract_Month-to-month', 'Contract_One year',
       'Contract_Two year', 'PaymentMethod_Bank transfer (automatic)',
       'PaymentMethod_Credit card (automatic)',
       'PaymentMethod_Electronic check', 'PaymentMethod_Mailed check'],
      dtype='object')
```

```
df2.sample(3)
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	(
4775	1	0	1	0	24	1	0	
5440	1	0	1	1	19	1	0	
6076	1	0	1	1	65	1	0	

3 rows × 27 columns

```
df2.dtypes
```

	0
gender	int64
SeniorCitizen	int64
Partner	int64
Dependents	int64
tenure	int64
PhoneService	int64
MultipleLines	int64
OnlineSecurity	int64
OnlineBackup	int64
DeviceProtection	int64
TechSupport	int64
StreamingTV	int64
StreamingMovies	int64
PaperlessBilling	int64
MonthlyCharges	float64
TotalCharges	float64
Churn	int64

Churn	int64
InternetService_DSL	bool
InternetService_Fiber optic	bool
InternetService_No	bool
Contract_Month-to-month	bool
Contract_One year	bool
Contract_Two year	bool
PaymentMethod_Bank transfer (automatic)	bool
PaymentMethod_Credit card (automatic)	bool
PaymentMethod_Electronic check	bool
PaymentMethod_Mailed check	bool

SCALING /255

```
cols_to_scale = ['tenure','MonthlyCharges','TotalCharges']#these cols not interms of 1s and 0
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df2[cols_to_scale] = scaler.fit_transform(df2[cols_to_scale])
```

```
for col in df2:
  print(f'{col}: {df2[col].unique()}')
```

```
gender: [1 0]
SeniorCitizen: [0 1]
Partner: [1 0]
Dependents: [0 1]
tenure: [0.         0.46478873 0.01408451 0.61971831 0.09859155 0.29577465
 0.12676056 0.38028169 0.85915493 0.16901408 0.21126761 0.8028169
 0.67605634 0.33802817 0.95774648 0.71830986 0.98591549 0.28169014
 0.15492958 0.4084507  0.64788732 1.         0.22535211 0.36619718
 0.05633803 0.63380282 0.14084507 0.97183099 0.87323944 0.5915493
 0.1971831  0.83098592 0.23943662 0.91549296 0.11267606 0.02816901
 0.42253521 0.69014085 0.88732394 0.77464789 0.08450704 0.57746479
 0.47887324 0.66197183 0.3943662  0.90140845 0.52112676 0.94366197
 0.43661972 0.76056338 0.50704225 0.49295775 0.56338028 0.07042254
 0.04225352 0.45070423 0.92957746 0.30985915 0.78873239 0.84507042
 0.18309859 0.26760563 0.73239437 0.54929577 0.81690141 0.32394366
 0.6056338  0.25352113 0.74647887 0.70422535 0.35211268 0.53521127]
PhoneService: [0 1]
MultipleLines: [0 1]
```

```
OnlineSecurity: [0 1]
OnlineBackup: [1 0]
DeviceProtection: [0 1]
TechSupport: [0 1]
StreamingTV: [0 1]
StreamingMovies: [0 1]
PaperlessBilling: [1 0]
MonthlyCharges: [0.11542289 0.38507463 0.35422886 ... 0.44626866 0.25820896 0.60149254]
TotalCharges: [0.0012751  0.21586661 0.01031041 ... 0.03780868 0.03321025 0.78764136]
Churn: [0 1]
InternetService_DSL: [ True False]
InternetService_Fiber optic: [False  True]
InternetService_No: [False  True]
Contract_Month-to-month: [ True False]
Contract_One year: [False  True]
Contract_Two year: [False  True]
PaymentMethod_Bank transfer (automatic): [False  True]
PaymentMethod_Credit card (automatic): [False  True]
PaymentMethod_Electronic check: [ True False]
PaymentMethod_Mailed check: [False  True]
```

```
x = df2.drop('Churn',axis='columns')
y = df2['Churn']
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=5)
x_train.shape
```

```
(5625, 26)
```

```
y_train.shape
```

```
(5625,)
```

```
y_test.shape
```

```
(1407,)
```

```
y_test[0:5]
```

	Churn
2660	0
744	0
5579	1
64	1
3287	1

dtype: int64

```
x_test.head()
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
2660	0	0	0	1	0.169014	1	0
744	1	0	0	0	0.056338	1	0
5579	1	0	1	1	0.971831	1	1
64	1	0	0	0	0.112676	1	1
3287	0	0	1	1	0.253521	1	1

5 rows × 26 columns

```
x_test.shape
```

```
(1407, 26)
```

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=100)
clf.fit(x_train,y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=100)
```

```
clf.score(x_test,y_test)
```

```
0.7114427860696517
```

```
from sklearn.ensemble import RandomForestClassifier
clf_forest = RandomForestClassifier()
clf_forest.fit(x_train,y_train)
```

```
RandomForestClassifier
RandomForestClassifier()
```

```
clf_forest.score(x_test,y_test)
```

```
0.7725657427149965
```

```
from sklearn.ensemble import AdaBoostClassifier

ada_boost_clf = AdaBoostClassifier(n_estimators=30)
ada_boost_clf.fit(x_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: Futur
  warnings.warn(
```

```
AdaBoostClassifier
AdaBoostClassifier(n_estimators=30)
```

```
ada_boost_clf.score(x_test,y_test)
```

```
0.7910447761194029
```

```
import tensorflow as tf
from tensorflow import keras

model = keras.Sequential([    #each neuron in i/p layer accept 1 feature
    keras.layers.Dense(20, input_shape=(26,), activation='relu'), #20 hidden
    keras.layers.Dense(1, activation='sigmoid'),
])
#ML is an art of experiments there is no like golden rule here

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=50)
```

```
Epoch 39/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8191 - loss: 0.3913
Epoch 40/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8191 - loss: 0.3937
Epoch 41/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 0s 916us/step - accuracy: 0.8107 - loss: 0.3923
Epoch 42/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 0s 918us/step - accuracy: 0.8129 - loss: 0.4012
Epoch 43/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 0s 846us/step - accuracy: 0.8222 - loss: 0.3905
Epoch 44/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 0s 907us/step - accuracy: 0.8208 - loss: 0.3867
Epoch 45/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 0s 856us/step - accuracy: 0.8123 - loss: 0.3993
Epoch 46/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 0s 849us/step - accuracy: 0.8278 - loss: 0.3747
Epoch 47/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 0s 846us/step - accuracy: 0.8170 - loss: 0.3945
Epoch 48/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 0s 913us/step - accuracy: 0.8164 - loss: 0.3940
Epoch 49/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 0s 932us/step - accuracy: 0.8167 - loss: 0.3945
Epoch 50/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 0s 884us/step - accuracy: 0.8114 - loss: 0.4024
<keras.src.callbacks.history.History at 0x7a8e5d4d5300>
```

```
model.evaluate(x_test, y_test)
```

```
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 718us/step - accuracy: 0.7969 - loss: 0.4352
[0.44227853417396545, 0.7889125943183899]
```

```
ypred = model.predict(x_test)
ypred[:5] #<0.5 means 0
```

```
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
array([[0.14174142],
       [0.4087761 ],
       [0.01403251],
       [0.71766174],
       [0.5009575 ]], dtype=float32)
```

```
y_pred =[]
for element in ypred:
  if element > 0.5:
    y_pred.append(1)
  else:
    y_pred.append(0)
```

```
y_pred[0:7]
```

```
[0, 0, 0, 1, 1, 1, 0]
```

```
y_test[0:7]
```

	Churn
2660	0
744	0
5579	1
64	1
3287	1
816	1
2670	0

dtype: int64

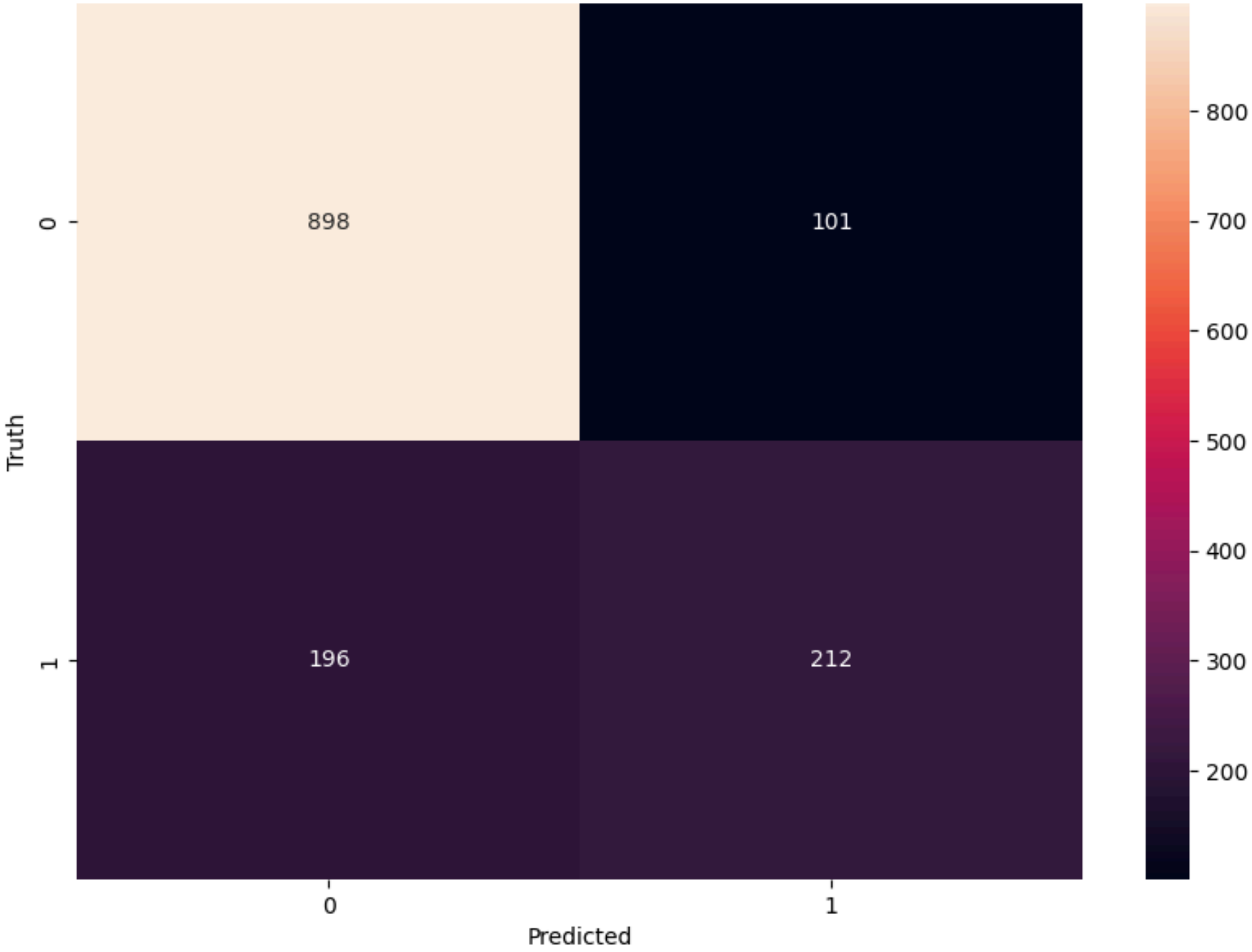
```
from sklearn.metrics import confusion_matrix , classification_report

print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.90      0.86       999
           1       0.68      0.52      0.59       408

    accuracy                           0.79      1407
   macro avg       0.75      0.71      0.72      1407
```

```
weighted avg       0.78      0.79      0.78      1407
```

```
import seaborn as sn
cm = tf.math.confusion_matrix(labels=y_test,predictions=y_pred)
plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
#
```

```
Text(95.72222222222221, 0.5, 'Truth')
```



ACCURACY

```
round((893+209)/(893+199+209+106),2)
```

```
0.78
```

Precision for 0 class i.e Precision for customer who did not churn

```
893/(893+106)
```

```
0.8938938938938938
```

Precision for 1 class i.e Precision for customer who actually churned

```
209/(209+209)
```

```
0.5
```

Recall for 0 class ie total correct pred for 0 class / total 0th samples

```
875/(875+124)
```

```
0.8758758758758759
```

Recall for 1 class ie total correct pred for 1 class / total 1th samples

```
240/(240+168)
```

```
0.5882352941176471
```

SMOTE - To Handle imbalance dataset

```
# Class count
count_class_0, count_class_1 = df1.Churn.value_counts()

# Dividing by class
df_class_0 = df2[df2['Churn'] == 0]
df_class_1 = df2[df2['Churn'] == 1]
```

```
df_class_0.shape
```

```
(5163, 27)
```

```
df_class_1.shape
```

```
(1869, 27)
```

```
X = df2.drop('Churn',axis='columns')
y = df2['Churn']
```

```
#imbalanced learn
from imblearn.over_sampling import SMOTE

smote = SMOTE(sampling_strategy='minority')
X_sm, y_sm = smote.fit_resample(X, y)

y_sm.value_counts()
```

Churn	count
0	5163
1	5163

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_sm, y_sm, test_size=0.2, random_state
```

```
y_train.value_counts()
```

Churn	count
0	4130

Start coding or generate with AI.

```
import tensorflow as tf
from tensorflow import keras
from sklearn.metrics import confusion_matrix , classification_report
```