Customer Churn Prediction Using Artificial Neural Network (ANN)

Customer churn prediction is to measure why customers are leaving a business. In this tutorial we will be looking at customer churn in telecom business. Build a **deep learning model** to predict the **churn** and use **precision,recall, f1-score** to measure performance of our model

```
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
%matplotlib inline
```

Load the data

```
df = pd.read_csv("customer_churn.csv")
df.sample(5)
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity
5815	2642-DTVCO	Male	1	No	No	9	Yes	Yes	Fiber optic	No
5611	6847-KJLTS	Female	1	Yes	No	58	Yes	Yes	Fiber optic	No
6850	0531-XBKMM	Male	0	No	Yes	66	Yes	Yes	DSL	Yes
6970	8083-YTZES	Male	0	No	No	4	Yes	Yes	Fiber optic	No
4481	8644-XYTSV	Male	0	Yes	No	42	No	No phone service	DSL	Yes

5 rows × 21 columns

First of all, drop customerID column as it is of no use

```
df.drop('customerID',axis='columns',inplace=True)
```

```
df.dtypes
```

```
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
MonthlyCharges      float64
TotalCharges         object
Churn                object
dtype: object
```

Quick glance at above makes me realize that TotalCharges should be float but it is an object. Let's check what's going on with this column

```
df.TotalCharges.values
```

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

Ahh... it is string. Lets convert it to numbers

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

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/_libs/lib.pyx in
pandas._libs.lib.maybe_convert_numeric()

ValueError: Unable to parse string " "

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
                                    2 frames
/usr/local/lib/python3.10/dist-packages/pandas/_libs/lib.pyx in
pandas._libs.lib.maybe_convert_numeric()

ValueError: Unable to parse string " " at position 488
```

Next steps: Explain error

some values seems to be not numbers but blank string. Let's find out such rows

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

```
0       False
1       False
2       False
3       False
4       False
        ...
7038    False
7039    False
7040    False
7041    False
7042    False
Name: TotalCharges, Length: 7043, dtype: bool
```

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

DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilli
Yes	Yes	Yes	No	Two year	Y
No internet service	No internet service	No internet service	No internet service	Two year	
Yes	No	Yes	Yes	Two year	
No internet service	No internet service	No internet service	No internet service	Two year	
Yes	Yes	Yes	No	Two year	
No internet service	No internet service	No internet service	No internet service	Two year	
No internet service	No internet service	No internet service	No internet service	Two year	
No internet service	No internet service	No internet service	No internet service	Two year	
No internet service	No internet service	No internet service	No internet service	One year	Y
Yes	Yes	Yes	No	Two year	
No	Yes	No	No	Two year	Y

```
df.shape
```

```
(7043, 20)
```

```
df.iloc[488].TotalCharges
```

```
' '
```

```
df[df.TotalCharges!=' '].shape
```

```
(7032, 20)
```

Remove rows with space in TotalCharges

```
df1 = df[df.TotalCharges!=' ']
df1.shape
```

```
(7032, 20)
```

```
df1.dtypes
```

```
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
MonthlyCharges      float64
TotalCharges         object
Churn                object
dtype: object
```

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

```
<ipython-input-67-b67e0c3d31a6>: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_guide/indexing.html#returning-a-view-versus
  df1.TotalCharges = pd.to_numeric(df1.TotalCharges)
```

```
df1.TotalCharges.values
```

```
array([  29.85, 1889.5 ,  108.15, ...,  346.45,  306.6 , 6844.5 ])
```

```
df1[df1.Churn=='No']
```

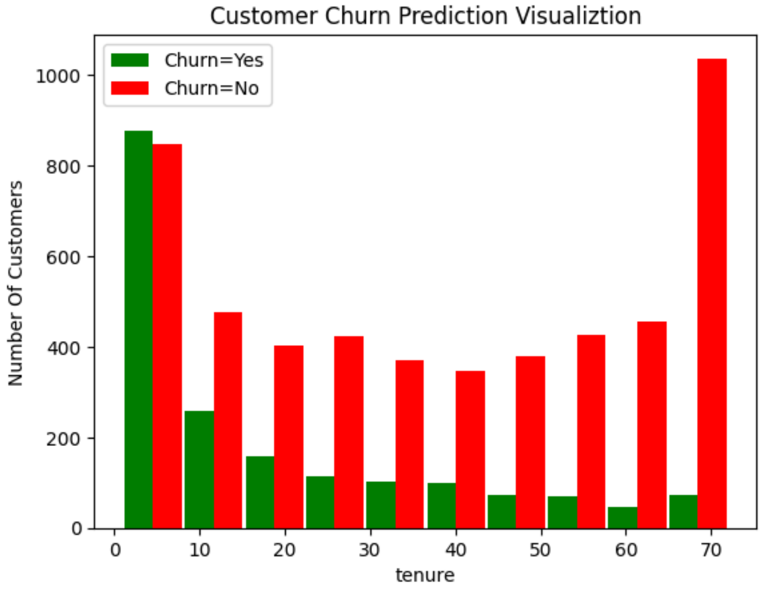
	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
0	Female	0	Yes	No	1	No	No phone service
1	Male	0	No	No	34	Yes	No
3	Male	0	No	No	45	No	No phone service
6	Male	0	No	Yes	22	Yes	Yes
7	Female	0	No	No	10	No	No phone service
...	...	...	...	...	...	...	..
7037	Female	0	No	No	72	Yes	No
7038	Male	0	Yes	Yes	24	Yes	Yes
7039	Female	0	Yes	Yes	72	Yes	Yes
7040	Female	0	Yes	Yes	11	No	No phone service
7042	Male	0	No	No	66	Yes	No

5163 rows × 20 columns

Data Visualization

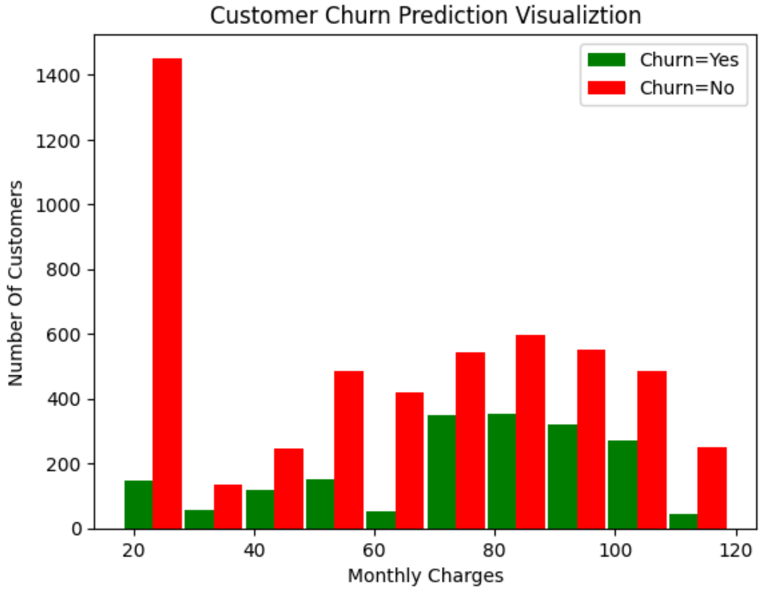
```
tenure_churn_no = df1[df1.Churn=='No'].tenure
tenure_churn_yes = df1[df1.Churn=='Yes'].tenure

plt.xlabel("tenure")
plt.ylabel("Number Of Customers")
plt.title("Customer Churn Prediction Visualiztion")


plt.hist([tenure_churn_yes, tenure_churn_no], rwidth=0.95, color=['green','red'],label=['Churn=Yes','Churn=No'])
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7aedc4833eb0>
```



```
import matplotlib.pyplot as plt

tenure_churn_no = df1[df1.Churn=='No'].tenure
tenure_churn_yes = df1[df1.Churn=='Yes'].tenure

plt.xlabel("Tenure")
plt.ylabel("Number Of Customers")
plt.title("Customer Churn Prediction Visualization")

plt.hist(tenure_churn_no, bins=20, color='green', alpha=0.7, label='Churn=No')
plt.hist(tenure_churn_yes, bins=20, color='red', alpha=0.7, label='Churn=Yes')
plt.legend()
plt.show()
```

```
<matplotlib.legend.Legend at 0x7aedc636add0>
```



Many of the columns are yes, no etc. Let's print unique values in object columns to see data values

```
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']
```

Some of the columns have no internet service or no phone service, that can be replaced with a simple No

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

```
<ipython-input-86-104b877f3854>: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_guide/indexing.html#returning-a-view-versus
  df1.replace('No internet service','No',inplace=True)
<ipython-input-86-104b877f3854>: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_guide/indexing.html#returning-a-view-versus
  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']
```

Convert Yes and No to 1 or 0

```
yes_no_columns = ['Partner','Dependents','PhoneService','MultipleLines','OnlineSecurity','OnlineBackup',
                  'DeviceProtection','TechSupport','StreamingTV','StreamingMovies','PaperlessBilling','Churn']
for col in yes_no_columns:
    df1[col].replace({'Yes': 1,'No': 0},inplace=True)
```

```
<ipython-input-88-34dfac0bf179>: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_guide/indexing.html#returning-a-view-versus
  df1[col].replace({'Yes': 1,'No': 0},inplace=True)
```

```
for col in df1:
    print(f'{col}: {df1[col].unique()}')
```

```
gender: ['Female' 'Male']
SeniorCitizen: [0 1]
Partner: [1 0]
Dependents: [0 1]
tenure: [ 1 34  2 45  8 22 10 28 62 13 16 58 49 25 69 52 71 21 12 30 47 72 17 27
  5 46 11 70 63 43 15 60 18 66  9  3 31 50 64 56  7 42 35 48 29 65 38 68
 32 55 37 36 41  6  4 33 67 23 57 61 14 20 53 40 59 24 44 19 54 51 26 39]
PhoneService: [0 1]
MultipleLines: [0 1]
InternetService: ['DSL' 'Fiber optic' 'No']
OnlineSecurity: [0 1]
OnlineBackup: [1 0]
DeviceProtection: [0 1]
TechSupport: [0 1]
StreamingTV: [0 1]
StreamingMovies: [0 1]
Contract: ['Month-to-month' 'One year' 'Two year']
PaperlessBilling: [1 0]
PaymentMethod: ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
MonthlyCharges: [29.85 56.95 53.85 ... 63.1  44.2  78.7 ]
TotalCharges: [  29.85 1889.5   108.15 ...  346.45  306.6  6844.5 ]
Churn: [0 1]
```

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

```
<ipython-input-90-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_guide/indexing.html#returning-a-view-versus
  df1['gender'].replace({'Female':1,'Male':0},inplace=True)
```

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

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

One hot encoding for categorical columns

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

```
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(5)
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
798	1	0	1	1	1	1	(
3968	1	0	1	0	72	1	(
5915	0	0	0	0	69	1	(
6599	1	0	1	0	13	1	1
2077	1	0	0	0	1	1	(

5 rows × 27 columns

```
df2.dtypes
```

```
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
InternetService_DSL                        uint8
InternetService_Fiber optic                uint8
InternetService_No                         uint8
Contract_Month-to-month                    uint8
Contract_One year                          uint8
Contract_Two year                          uint8
PaymentMethod_Bank transfer (automatic)    uint8
PaymentMethod_Credit card (automatic)      uint8
PaymentMethod_Electronic check             uint8
PaymentMethod_Mailed check                 uint8
dtype: object
```

```
cols_to_scale = ['tenure','MonthlyCharges','TotalCharges']

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: [1 0]
InternetService_Fiber optic: [0 1]
InternetService_No: [0 1]
Contract_Month-to-month: [1 0]
Contract_One year: [0 1]
Contract_Two year: [0 1]
PaymentMethod_Bank transfer (automatic): [0 1]
PaymentMethod_Credit card (automatic): [0 1]
PaymentMethod_Electronic check: [1 0]
PaymentMethod_Mailed check: [0 1]
```

Train test split

```
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)
```

```
X_test.shape
```

```
(1407, 26)
```

```
X_train[:10]
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLin
5664	1	1	0	0	0.126761	1	
101	1	0	1	1	0.000000	1	
2621	0	0	1	0	0.985915	1	
392	1	1	0	0	0.014085	1	
1327	0	0	1	0	0.816901	1	
3607	1	0	0	0	0.169014	1	
2773	0	0	1	0	0.323944	0	
1936	1	0	1	0	0.704225	1	
5387	0	0	0	0	0.042254	0	
4331	0	0	0	0	0.985915	1	

10 rows × 26 columns

```
len(X_train.columns)
```

```
26
```

Build a model (ANN) in tensorflow/keras

```
import tensorflow as tf
from tensorflow import keras
```

```
model = keras.Sequential([
    keras.layers.Dense(26, input_shape=(26,), activation='relu'),
    keras.layers.Dense(15, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

# opt = keras.optimizers.Adam(learning_rate=0.01)

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

model.fit(X_train, y_train, epochs=100)
```

```
Epoch 1/100
176/176 [==============================] - 1s 2ms/step - loss: 0.5281 - accuracy: 0.7223
Epoch 2/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4300 - accuracy: 0.7915
Epoch 3/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4191 - accuracy: 0.8011
Epoch 4/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4143 - accuracy: 0.8046
Epoch 5/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4118 - accuracy: 0.8044
Epoch 6/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4094 - accuracy: 0.8112
Epoch 7/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4075 - accuracy: 0.8092
Epoch 8/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4058 - accuracy: 0.8116
Epoch 9/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4043 - accuracy: 0.8132
Epoch 10/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4035 - accuracy: 0.8151
Epoch 11/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4023 - accuracy: 0.8172
Epoch 12/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4017 - accuracy: 0.8160
Epoch 13/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4003 - accuracy: 0.8199
Epoch 14/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3991 - accuracy: 0.8169
Epoch 15/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3996 - accuracy: 0.8187
```

```
Epoch 16/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3980 - accuracy: 0.8178
Epoch 17/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3973 - accuracy: 0.8169
Epoch 18/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3959 - accuracy: 0.8160
Epoch 19/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3946 - accuracy: 0.8185
Epoch 20/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3953 - accuracy: 0.8213
Epoch 21/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3928 - accuracy: 0.8194
Epoch 22/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3928 - accuracy: 0.8187
Epoch 23/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3920 - accuracy: 0.8199
Epoch 24/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3901 - accuracy: 0.8197
Epoch 25/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3902 - accuracy: 0.8190
Epoch 26/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3889 - accuracy: 0.8197
Epoch 27/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3881 - accuracy: 0.8192
Epoch 28/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3881 - accuracy: 0.8206
Epoch 29/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3869 - accuracy: 0.8213
```

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

```
44/44 [==============================] - 0s 2ms/step - loss: 0.4867 - accuracy: 0.7747
[0.4866882264614105, 0.7746979594230652]
```

```
yp = model.predict(X_test)
yp[:5]
```

```
44/44 [==============================] - 0s 2ms/step
array([[0.47797865],
       [0.5675569 ],
       [0.00207685],
       [0.74981403],
       [0.29448453]], dtype=float32)
```

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

```
y_pred[:10]
```

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

```
y_test[:10]
```

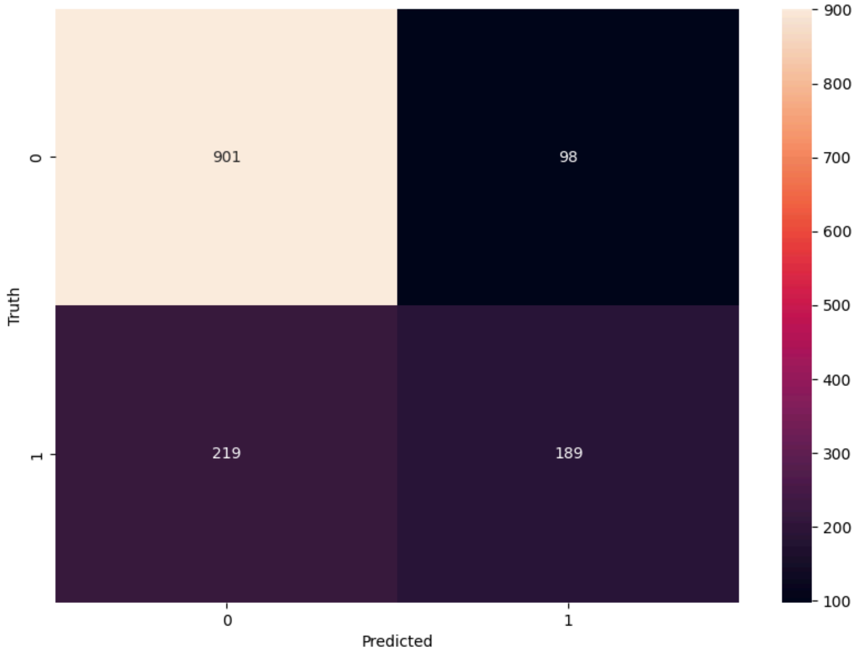
```
2660    0
744     0
5579    1
64      1
3287    1
816     1
2670    0
5920    0
1023    0
6087    0
Name: Churn, dtype: int64
```

```
from sklearn.metrics import confusion_matrix , classification_report

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

```
              precision    recall  f1-score   support

           0       0.80      0.90      0.85       999
           1       0.66      0.46      0.54       408

    accuracy                           0.77      1407
   macro avg       0.73      0.68      0.70      1407
weighted avg       0.76      0.77      0.76      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')
```



```
y_test.shape
```

```
(1407,)
```

Accuracy

```
round((862+229)/(862+229+137+179),2)
```

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

```
round(862/(862+179),2)
```

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

```
round(229/(229+137),2)
```

Recall for 0 class

```
round(862/(862+137),2)
```

```
round(229/(229+179),2)
```

Start coding or generate with AI.