Customer Churn Prediction Using Artificial Neural Network (ANN)

This project focuses on predicting customer churn for a telecommunications company using machine learning techniques. Customer churn, which refers to customers terminating their subscriptions or switching to a different service provider, is a significant challenge faced by many businesses, particularly in the telecommunications industry. The ability to accurately predict customer churn can help companies devise effective strategies to retain valuable customers and improve overall customer satisfaction.

Project Overview

The project utilizes a dataset containing customer information, such as demographic details, service subscriptions, billing information, and churn status. The goal is to build a predictive model that can identify customers who are likely to churn, enabling the company to take proactive measures to prevent customer attrition.

Key Steps

Data Exploration and Preprocessing: The dataset is loaded and explored to gain insights into the data structure, identify missing values, and handle any inconsistencies or outliers.

Feature Engineering: Relevant features are extracted, and necessary transformations are performed, such as one-hot encoding for categorical variables and scaling for numerical features.

Model Building: An artificial neural network (ANN) model is constructed using TensorFlow/Keras, a popular deep learning library. The model architecture is defined, and the model is trained on the preprocessed data.

Model Evaluation: The trained model is evaluated on a held-out test dataset to assess its performance using various metrics, such as accuracy, precision, recall, and the confusion matrix.

Interpretation and Insights: The results of the model are analyzed, and insights are derived to understand the factors contributing to customer churn and the potential impact on the business.

In [3]:
```
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
%matplotlib inline
```

Load the data

In [7]:
```
df = pd.read_csv("C:/Users\Dipanwita Sikder\Documents\Python Project\custome
df.sample(5)
```

Out[7]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	Multiple
1745	2418-TPEUN	Female	0	Yes	Yes	56	Yes	
1533	8519-QJGJD	Female	0	No	No	14	Yes	
6381	0927-CNGRH	Male	0	No	Yes	1	Yes	
6037	7537-CBQUZ	Male	1	No	No	63	Yes	
3894	5989-AXPUC	Female	0	Yes	No	68	Yes	

5 rows × 21 columns

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

In [8]:
```
df.drop(columns=['customerID'], inplace=True)
```

In [9]:
```
df.dtypes
```

Out[9]:
```
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
```

TotalCharges should be float but it is an object.

In [10]:
```
df.TotalCharges.values
```

Out[10]:
```
array(['29.85', '1889.5', '108.15', ..., '346.45', '306.6', '6844.5'],
      dtype=object)
```

Convert this column to numbers

In [11]:
```
pd.to_numeric(df.TotalCharges)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\anaconda3\lib\site-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)
~\AppData\Local\Temp\ipykernel_6600\2112264836.py in <module>
----> 1 pd.to_numeric(df.TotalCharges)

~\anaconda3\lib\site-packages\pandas\core\tools\numeric.py in to_numeric(arg, errors, downcast)
    182         coerce_numeric = errors not in ("ignore", "raise")
    183         try:
--> 184             values, _ = lib.maybe_convert_numeric(
    185                 values, set(), coerce_numeric=coerce_numeric
    186             )

~\anaconda3\lib\site-packages\pandas\_libs\lib.pyx in pandas._libs.lib.maybe_convert_numeric()

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

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

In [12]:
```
pd.to_numeric(df.TotalCharges,errors='coerce').isnull()
```

Out[12]:
```
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
```

In [13]:
```
df[pd.to_numeric(df.TotalCharges,errors='coerce').isnull()]
```

Out[13]:

ernetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	Strear
DSL	Yes	No	Yes	Yes	Yes	
No	No internet service	No internet service	No internet service	No internet service	No internet service	
DSL	Yes	Yes	Yes	No	Yes	
No	No internet service	No internet service	No internet service	No internet service	No internet service	
DSL	Yes	Yes	Yes	Yes	Yes	
No	No internet service	No internet service	No internet service	No internet service	No internet service	
No	No internet service	No internet service	No internet service	No internet service	No internet service	
No	No internet service	No internet service	No internet service	No internet service	No internet service	
No	No internet service	No internet service	No internet service	No internet service	No internet service	
DSL	No	Yes	Yes	Yes	Yes	
DSL	Yes	Yes	No	Yes	No	

In [14]:
```
df.shape
```

Out[14]:
```
(7043, 20)
```

In [15]:
```
df.iloc[488].TotalCharges
```

Out[15]:
```
' '
```

In [16]:
```
df[df.TotalCharges!=' '].shape
```

Out[16]:
```
(7032, 20)
```

Remove rows with space in TotalCharges

In [17]:
```
df1 = df[df.TotalCharges!=' ']
df1.shape
```

Out[17]:
```
(7032, 20)
```

In [18]:
```
df1.dtypes
```

```
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
```

In [19]:
```
df1.TotalCharges = pd.to_numeric(df1.TotalCharges)
```

```
C:\Users\Dipanwita Sikder\AppData\Local\Temp\ipykernel_6600\973151263.py:
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-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df1.TotalCharges = pd.to_numeric(df1.TotalCharges)
```

In [20]:
```
df1.TotalCharges.values
```

Out[20]:
```
array([  29.85, 1889.5 ,  108.15, ...,  346.45,  306.6 , 6844.5 ])
```

In [21]:

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

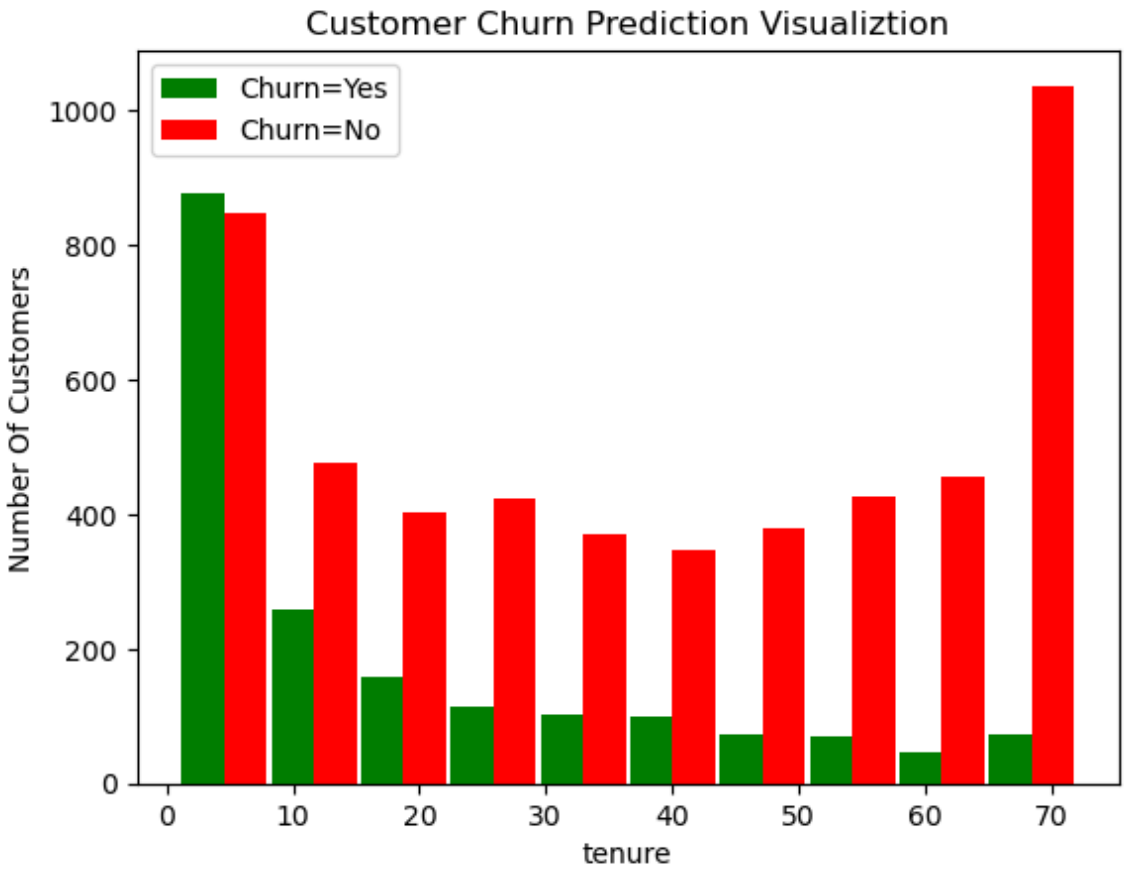
Out[21]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Int
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	

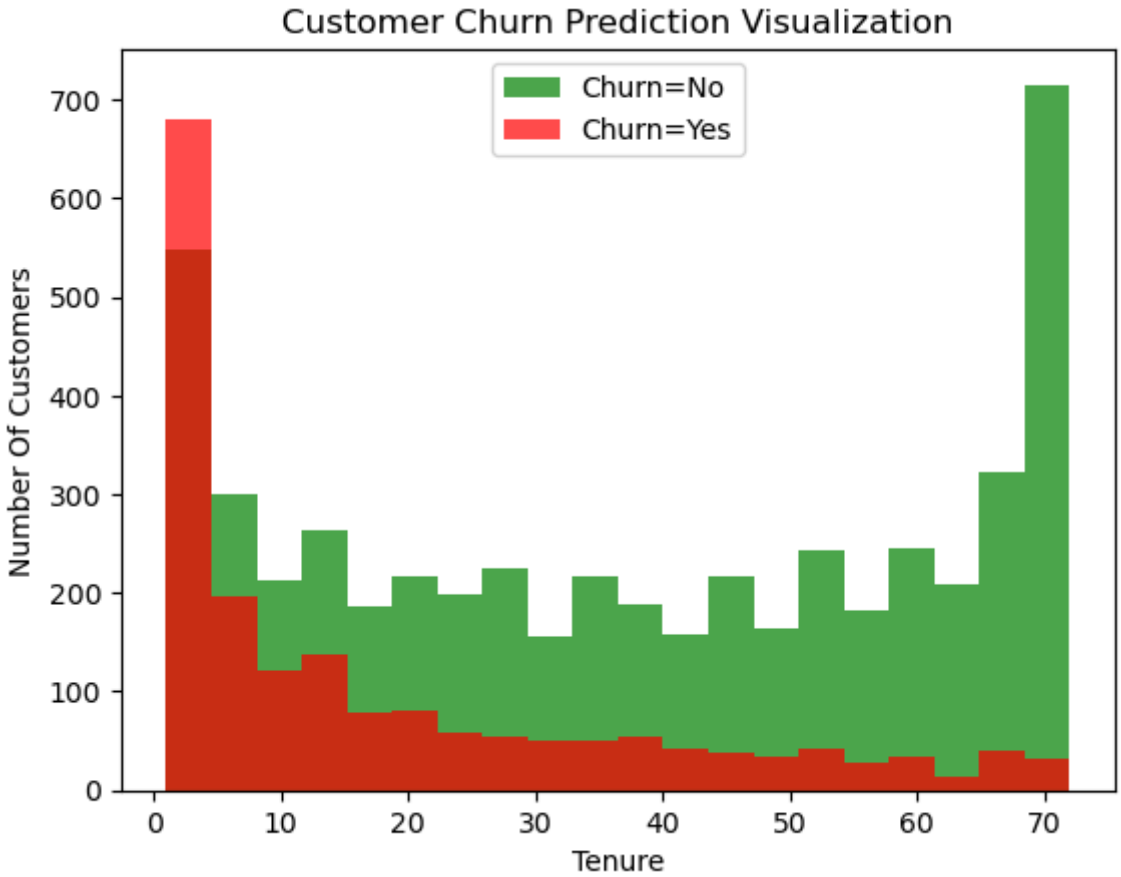
Data Visualization

In [22]:
```
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','
plt.legend()
```

Out[22]:
```
<matplotlib.legend.Legend at 0x1ca24cab730>
```



In [23]:
```
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=N
plt.hist(tenure_churn_yes, bins=20, color='red', alpha=0.7, label='Churn=Ye
plt.legend()
plt.show()
```



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

In [24]:
```
def print_unique_col_values(df):
       for column in df:
            if df[column].dtypes=='object':
                print(f'{column}: {df[column].unique()}')
```

In [25]:
```
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 (automati
c)'
 '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

In [26]:
```
df1.replace('No internet service','No',inplace=True)
df1.replace('No phone service','No',inplace=True)
```

```
C:\Users\Dipanwita Sikder\AppData\Local\Temp\ipykernel_6600\2045096646.py:
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-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df1.replace('No internet service','No',inplace=True)
C:\Users\Dipanwita Sikder\AppData\Local\Temp\ipykernel_6600\2045096646.py:
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-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df1.replace('No phone service','No',inplace=True)
```

In [27]:
```
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 (automati
c)'
 'Credit card (automatic)']
Churn: ['No' 'Yes']
```

Convert Yes and No to 1 or 0

In [28]:
```
yes_no_columns = ['Partner','Dependents','PhoneService','MultipleLines','On
                  'DeviceProtection','TechSupport','StreamingTV','Streaming
for col in yes_no_columns:
    df1[col].replace({'Yes': 1,'No': 0},inplace=True)
```

```
C:\Users\Dipanwita Sikder\AppData\Local\Temp\ipykernel_6600\1648037665.py:
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-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df1[col].replace({'Yes': 1,'No': 0},inplace=True)
```

In [29]:
```
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 (automati
c)'
 '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]
```

In [30]:
```
df1['gender'].replace({'Female':1,'Male':0},inplace=True)
```

```
C:\Users\Dipanwita Sikder\AppData\Local\Temp\ipykernel_6600\698335744.py:
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-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df1['gender'].replace({'Female':1,'Male':0},inplace=True)
```

In [31]:
```
df1.gender.unique()
```

Out[31]:
```
array([1, 0], dtype=int64)
```

One hot encoding for categorical columns

In [32]:
```
df2 = pd.get_dummies(data=df1, columns=['InternetService','Contract','Payme
df2.columns
```

Out[32]:
```
Index(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure',
       'PhoneService', 'MultipleLines', 'OnlineSecurity', 'OnlineBackup',
       'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovie
s',
       'PaperlessBilling', 'MonthlyCharges', 'TotalCharges', 'Churn',
       'InternetService_DSL', 'InternetService_Fiber optic',
       'InternetService_No', 'Contract_Month-to-month', 'Contract_One yea
r',
       'Contract_Two year', 'PaymentMethod_Bank transfer (automatic)',
       'PaymentMethod_Credit card (automatic)',
       'PaymentMethod_Electronic check', 'PaymentMethod_Mailed check'],
      dtype='object')
```

In [33]:
```
df2.sample(5)
```

Out[33]:

es	OnlineSecurity	OnlineBackup	DeviceProtection	...	InternetService_DSL	InternetService_Fib op
1	1	1	1	...	1	
0	0	0	0	...	0	
0	0	1	0	...	0	
0	0	0	1	...	1	
0	0	0	1	...	1	

In [34]:
```
df2.dtypes
```

Out[34]:
```
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
```

In [35]:
```
cols_to_scale = ['tenure','MonthlyCharges','TotalCharges']

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df2[cols_to_scale] = scaler.fit_transform(df2[cols_to_scale])
```

In [36]:
```
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 0.2582089
6 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

In [37]:
```
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,rando
```

In [38]:
```
X_train.shape
```

Out[38]: (5625, 26)

In [39]:
```
X_test.shape
```

Out[39]: (1407, 26)

In [40]:
```
X_train[:10]
```

Out[40]:

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

10 rows × 26 columns

In [41]:
```
len(X_train.columns)
```

Out[41]: 26

Build a model (ANN) in tensorflow/keras

In [44]:

```
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 1ms/step - loss: 0.4873 - ac
curacy: 0.7572
Epoch 2/100
176/176 [==============================] - 0s 1ms/step - loss: 0.4256 - ac
curacy: 0.7964
Epoch 3/100
176/176 [==============================] - 0s 1ms/step - loss: 0.4192 - ac
curacy: 0.8011
Epoch 4/100
176/176 [==============================] - 0s 1ms/step - loss: 0.4150 - ac
curacy: 0.8036
Epoch 5/100
176/176 [==============================] - 0s 1ms/step - loss: 0.4125 - ac
curacy: 0.8027
Epoch 6/100
176/176 [==============================] - 0s 1ms/step - loss: 0.4109 - ac
curacy: 0.8039
Epoch 7/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4090 - ac
curacy: 0.8052
Epoch 8/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4078 - ac
curacy: 0.8069
Epoch 9/100
176/176 [==============================] - 0s 1ms/step - loss: 0.4060 - ac
curacy: 0.8073
Epoch 10/100
176/176 [==============================] - 0s 1ms/step - loss: 0.4040 - ac
curacy: 0.8076
Epoch 11/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4041 - ac
curacy: 0.8128
Epoch 12/100
176/176 [==============================] - 0s 1ms/step - loss: 0.4019 - ac
curacy: 0.8080
Epoch 13/100
176/176 [==============================] - 0s 1ms/step - loss: 0.4009 - ac
curacy: 0.8098
Epoch 14/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3996 - ac
curacy: 0.8117
Epoch 15/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3981 - ac
curacy: 0.8107
Epoch 16/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3972 - ac
curacy: 0.8142
Epoch 17/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3966 - ac
curacy: 0.8128
Epoch 18/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3958 - ac
curacy: 0.8162
Epoch 19/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3954 - ac
curacy: 0.8149
Epoch 20/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3933 - ac
curacy: 0.8148
Epoch 21/100
```

```
176/176 [==============================] - 0s 1ms/step - loss: 0.3921 - ac
curacy: 0.8133
Epoch 22/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3907 - ac
curacy: 0.8140
Epoch 23/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3910 - ac
curacy: 0.8155
Epoch 24/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3902 - ac
curacy: 0.8140
Epoch 25/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3894 - ac
curacy: 0.8164
Epoch 26/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3888 - ac
curacy: 0.8167
Epoch 27/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3874 - ac
curacy: 0.8165
Epoch 28/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3870 - ac
curacy: 0.8208
Epoch 29/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3864 - ac
curacy: 0.8190
Epoch 30/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3857 - ac
curacy: 0.8178
Epoch 31/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3840 - ac
curacy: 0.8169
Epoch 32/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3846 - ac
curacy: 0.8192
Epoch 33/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3821 - ac
curacy: 0.8199
Epoch 34/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3816 - ac
curacy: 0.8194
Epoch 35/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3808 - ac
curacy: 0.8206
Epoch 36/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3795 - ac
curacy: 0.8217
Epoch 37/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3793 - ac
curacy: 0.8201
Epoch 38/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3781 - ac
curacy: 0.8231
Epoch 39/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3772 - ac
curacy: 0.8204
Epoch 40/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3765 - ac
curacy: 0.8188
Epoch 41/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3757 - ac
```

```
curacy: 0.8247
Epoch 42/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3762 - ac
curacy: 0.8240
Epoch 43/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3742 - ac
curacy: 0.8240
Epoch 44/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3739 - ac
curacy: 0.8236
Epoch 45/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3736 - ac
curacy: 0.8219
Epoch 46/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3718 - ac
curacy: 0.8260
Epoch 47/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3714 - ac
curacy: 0.8249
Epoch 48/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3718 - ac
curacy: 0.8244
Epoch 49/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3691 - ac
curacy: 0.8256
Epoch 50/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3692 - ac
curacy: 0.8274
Epoch 51/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3690 - ac
curacy: 0.8260
Epoch 52/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3680 - ac
curacy: 0.8267
Epoch 53/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3678 - ac
curacy: 0.8284
Epoch 54/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3674 - ac
curacy: 0.8283
Epoch 55/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3670 - ac
curacy: 0.8277
Epoch 56/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3661 - ac
curacy: 0.8292
Epoch 57/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3646 - ac
curacy: 0.8277
Epoch 58/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3660 - ac
curacy: 0.8295
Epoch 59/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3648 - ac
curacy: 0.8270
Epoch 60/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3629 - ac
curacy: 0.8288
Epoch 61/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3645 - ac
curacy: 0.8267
```

```
Epoch 62/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3627 - ac
curacy: 0.8286
Epoch 63/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3610 - ac
curacy: 0.8284
Epoch 64/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3606 - ac
curacy: 0.8304
Epoch 65/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3600 - ac
curacy: 0.8315
Epoch 66/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3596 - ac
curacy: 0.8345
Epoch 67/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3599 - ac
curacy: 0.8300
Epoch 68/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3598 - ac
curacy: 0.8316
Epoch 69/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3598 - ac
curacy: 0.8322
Epoch 70/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3590 - ac
curacy: 0.8359
Epoch 71/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3586 - ac
curacy: 0.8336
Epoch 72/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3567 - ac
curacy: 0.8331
Epoch 73/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3568 - ac
curacy: 0.8343
Epoch 74/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3563 - ac
curacy: 0.8363
Epoch 75/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3547 - ac
curacy: 0.8345
Epoch 76/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3560 - ac
curacy: 0.8308
Epoch 77/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3559 - ac
curacy: 0.8336
Epoch 78/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3550 - ac
curacy: 0.8347
Epoch 79/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3547 - ac
curacy: 0.8352
Epoch 80/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3536 - ac
curacy: 0.8373
Epoch 81/100
```

```
176/176 [==============================] - 0s 2ms/step - loss: 0.3527 - ac
curacy: 0.8336
Epoch 82/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3526 - ac
curacy: 0.8364
Epoch 83/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3529 - ac
curacy: 0.8341
Epoch 84/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3529 - ac
curacy: 0.8375
Epoch 85/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3528 - ac
curacy: 0.8354
Epoch 86/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3520 - ac
curacy: 0.8345
Epoch 87/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3516 - ac
curacy: 0.8416
Epoch 88/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3510 - ac
curacy: 0.8377
Epoch 89/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3503 - ac
curacy: 0.8363
Epoch 90/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3503 - ac
curacy: 0.8386
Epoch 91/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3486 - ac
curacy: 0.8389
Epoch 92/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3498 - ac
curacy: 0.8388
Epoch 93/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3487 - ac
curacy: 0.8366
Epoch 94/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3479 - ac
curacy: 0.8373
Epoch 95/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3472 - ac
curacy: 0.8398
Epoch 96/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3463 - ac
curacy: 0.8402
Epoch 97/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3466 - ac
curacy: 0.8384
Epoch 98/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3463 - ac
curacy: 0.8386
Epoch 99/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3459 - ac
curacy: 0.8388
Epoch 100/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3455 - ac
curacy: 0.8388
```

Out[44]:

```
<keras.callbacks.History at 0x1ca33fa7d30>
```

In [45]:
```
model.evaluate(X_test, y_test)
```

```
44/44 [==============================] - 0s 2ms/step - loss: 0.5014 - accu
racy: 0.7520
```

Out[45]:
```
[0.5013915300369263, 0.7519544959068298]
```

In [46]:
```
yp = model.predict(X_test)
yp[:5]
```

```
44/44 [==============================] - 0s 1ms/step
```

Out[46]:
```
array([[0.12830515],
       [0.3967417 ],
       [0.0235492 ],
       [0.6783335 ],
       [0.6498011 ]], dtype=float32)
```

In [47]:
```
y_pred = []
for element in yp:
    if element > 0.5:
        y_pred.append(1)
    else:
        y_pred.append(0)
```

In [48]:
```
y_pred[:10]
```

Out[48]:
```
[0, 0, 0, 1, 1, 1, 0, 0, 0, 0]
```

In [49]:
```
y_test[:10]
```

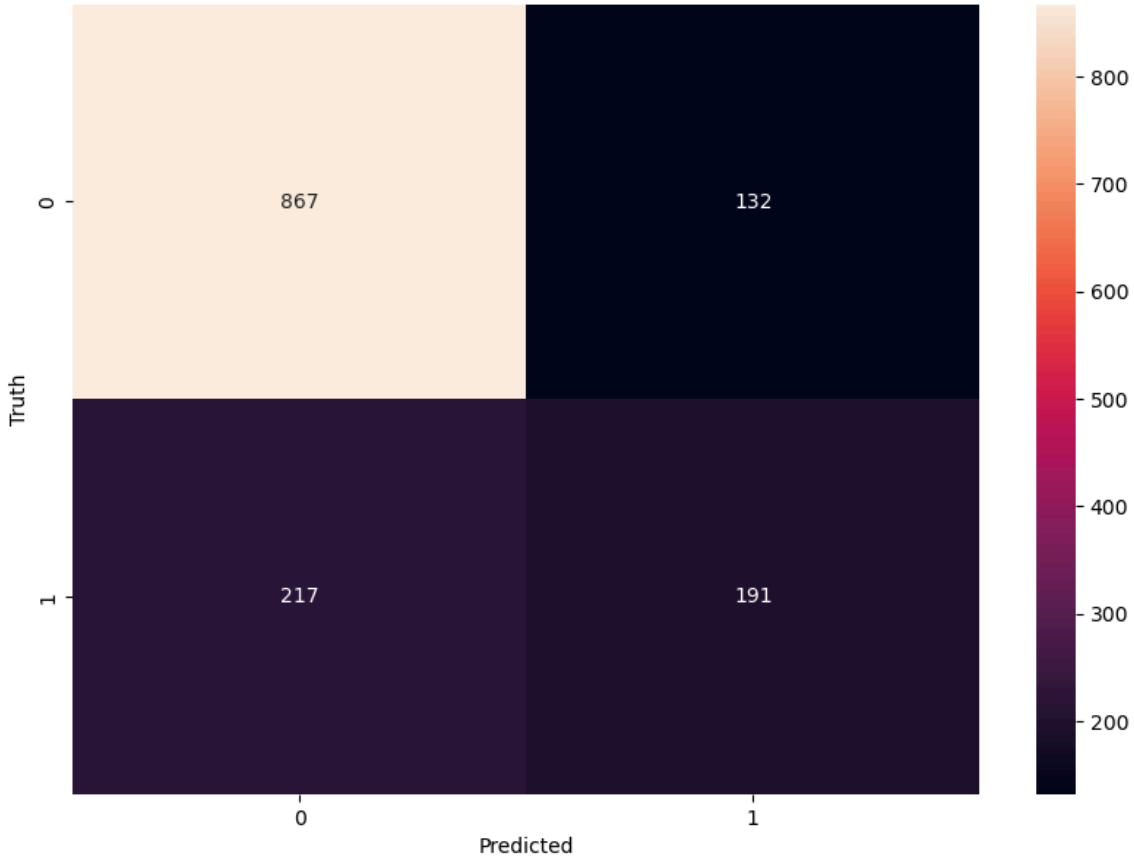
Out[49]:
```
2660    0
744     0
5579    1
64      1
3287    1
816     1
2670    0
5920    0
1023    0
6087    0
Name: Churn, dtype: int64
```

In [50]:
```
from sklearn.metrics import confusion_matrix , classification_report

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

```
              precision    recall  f1-score   support

           0       0.80      0.87      0.83       999
           1       0.59      0.47      0.52       408

    accuracy                           0.75      1407
   macro avg       0.70      0.67      0.68      1407
weighted avg       0.74      0.75      0.74      1407
```

In [51]:
```
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')
```

Out[51]: Text(95.72222222222221, 0.5, 'Truth')



In [52]:
```
y_test.shape
```

Out[52]: (1407,)

Accuracy

In [53]:
```
round((862+229)/(862+229+137+179),2)
```

Out[53]: 0.78

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

In [54]:
```
round(862/(862+179),2)
```

Out[54]: 0.83

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

In [55]:
```
round(229/(229+137),2)
```
Out[55]: 0.63

Recall for 0 class

In [56]:
```
round(862/(862+137),2)
```
Out[56]: 0.86

In [57]:
```
round(229/(229+179),2)
```
Out[57]: 0.56

In [ ]:

In [ ]: