In [1]:

```
# Customer Churn Prediction Using Artificial Neural Network (ANN)
```

In [ ]:

```
## Customer churn prediction is to measure why customers are leaving a business
```

In [7]:

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

In [10]:

```
df = pd.read_csv("/content/Customer_chun.csv", sep=';', error_bad_lines=False)
df.sample(5)
```

Out[10]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSe
1593	0620-XEFWH	Male	0	Yes	Yes	4	Yes	No	No	No int se
1424	7560-QRBXH	Female	0	No	Yes	48	Yes	No	No	No int se
213	4709-LKHYG	Female	0	Yes	Yes	29	Yes	No	No	No int se
5507	5753-QQWPW	Female	0	No	No	28	Yes	No	DSL	
1836	6365-MTGZX	Male	0	No	No	24	Yes	No	Fiber optic	

In [12]:

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

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

In [13]:

```
df.dtypes
```

Out[13]:

```
gender                int64
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 [14]:

```
## 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
```

Out[14]:

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

In [15]:

```
## it is string. Lets convert it to numbers

pd.to_numeric(df.TotalCharges)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
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)
<ipython-input-15-50cee5d59a76> in <module>()
      1 ## it is string. Lets convert it to numbers
      2
----> 3 pd.to_numeric(df.TotalCharges)

/usr/local/lib/python3.7/dist-packages/pandas/core/tools/numeric.py in to_numeric(arg, er
rors, downcast)
    151         try:
    152             values = lib.maybe_convert_numeric(
--> 153                 values, set(), coerce_numeric=coerce_numeric
    154             )
    155         except (ValueError, TypeError):

pandas/_libs/lib.pyx in pandas._libs.lib.maybe_convert_numeric()

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

In [ ]:

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

In [16]:

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

Out[16]:

```
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 [17]:

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

Out[17]:

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

In [ ]:

In [18]:

```
df.shape
```

Out[18]:

```
(7043, 20)
```

In [19]:

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

Out[19]:

```
' '
```

In [20]:

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

Out[20]:

```
(7032, 20)
```

In [22]:

```
## Remove rows with space in TotalCharges

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

```
df1.shape
```

Out[22]:

```
(7032, 20)
```

In [23]:

```
df1.dtypes
```

Out[23]:

```
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
```

In [24]:

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

```
/usr/local/lib/python3.7/dist-packages/pandas/core/generic.py:5170: SettingWithCopyWarnin
g:
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_g
uide/indexing.html#returning-a-view-versus-a-copy
  self[name] = value
```

In [25]:

```
df1.TotalCharges.values
```

Out[25]:

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

In [26]:

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

Out[26]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	Online
0	Female	0	Yes	No	1	No	No phone service	DSL	No	
1	Male	0	No	No	34	Yes	No	DSL	Yes	
3	Male	0	No	No	45	No	No phone service	DSL	Yes	
6	Male	0	No	Yes	22	Yes	Yes	Fiber optic	No	

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	Online
7	Female	0	No	No	10	No	No phone service	DSL	Yes	
...	...	...	...	...	...	...	...	...	...	
7037	Female	0	No	No	72	Yes	No	No	No internet service	No
7038	Male	0	Yes	Yes	24	Yes	Yes	DSL	Yes	
7039	Female	0	Yes	Yes	72	Yes	Yes	Fiber optic	No	
7040	Female	0	Yes	Yes	11	No	No phone service	DSL	Yes	
7042	Male	0	No	No	66	Yes	No	Fiber optic	Yes	

5163 rows × 20 columns

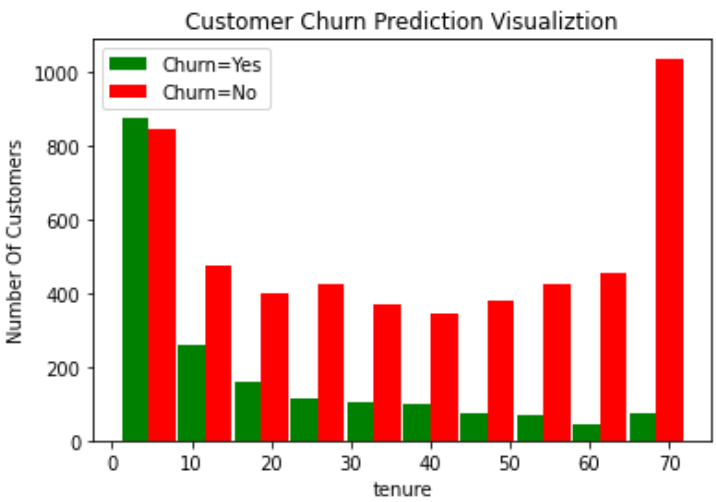
In [27]:

```
## 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")

blood_sugar_men = [113, 85, 90, 150, 149, 88, 93, 115, 135, 80, 77, 82, 129]
blood_sugar_women = [67, 98, 89, 120, 133, 150, 84, 69, 89, 79, 120, 112, 100]

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

```
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83: VisibleDeprecationWarni
ng: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-o
r-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do
this, you must specify 'dtype=object' when creating the ndarray
  return array(a, dtype, copy=False, order=order)
```

Out[27]:

```
<matplotlib.legend.Legend at 0x7f6b95b58710>
```



In [28]:

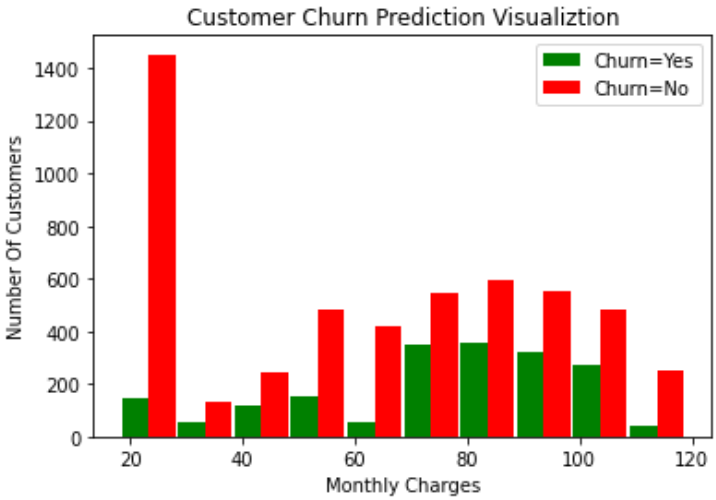
```
mc_churn_no = df1[df1.Churn=='No'].MonthlyCharges
```

```
mc_churn_no = df1[df1.Churn=='No'].MonthlyCharges
mc_churn_yes = df1[df1.Churn=='Yes'].MonthlyCharges

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

blood_sugar_men = [113, 85, 90, 150, 149, 88, 93, 115, 135, 80, 77, 82, 129]
blood_sugar_women = [67, 98, 89, 120, 133, 150, 84, 69, 89, 79, 120, 112, 100]

plt.hist([mc_churn_yes, mc_churn_no], rwidth=0.95, color=['green','red'],label=['Churn=Y
es','Churn=No'])
plt.legend()
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83: VisibleDeprecationWarni
ng: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-o
r-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do
this, you must specify 'dtype=object' when creating the ndarray
  return array(a, dtype, copy=False, order=order)
```

Out[28]:

```
<matplotlib.legend.Legend at 0x7f6b95276b90>
```



In [29]:

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

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

In [30]:

```
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']
```

In [31]:

```
## 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)
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py:4389: 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_g
uide/indexing.html#returning-a-view-versus-a-copy
  method=method,
```

In [32]:

```
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']
```

In [33]:

```
## Convert Yes and No to 1 or 0

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

```
/usr/local/lib/python3.7/dist-packages/pandas/core/series.py:4582: 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_g
uide/indexing.html#returning-a-view-versus-a-copy
  method=method,
```

In [34]:

```
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]
```

In [35]:

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

```
/usr/local/lib/python3.7/dist-packages/pandas/core/series.py:4582: 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_g
uide/indexing.html#returning-a-view-versus-a-copy
  method=method,
```

In [36]:

```
## One hot encoding for categorical columns

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

Out[36]:

```
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')
```

In [37]:

```
df2.sample(5)
```

Out[37]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	OnlineSecurity	OnlineBackup	DeviceP
1843	1	0	1	1	71	1	0	0	0	
5116	0	0	0	0	1	1	0	0	0	
1509	0	0	0	0	17	1	0	0	0	
1518	1	0	0	0	5	1	0	0	0	
3231	1	0	0	0	15	1	0	1	1	

In [38]:

```
df2.dtypes
```

Out[38]:

```
gender                                     int64
SeniorCitizen                              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
```

In [39]:

```
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 [40]:

```
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_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]
```

In [41]:

```
## 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)
```

In [42]:

```
X_train.shape
```

Out[42]:

```
(5625, 26)
```

In [43]:

```
X_test.shape
```

Out[43]:

```
(1407, 26)
```

In [44]:

```
X_train[:10]
```

Out[44]:

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

In [45]:

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

Out[45]:

```
26
```

In [46]:

```
## Build a model (ANN) in tensorflow/keras
```

```python
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 [==============================] - 3s 4ms/step - loss: 0.5069 - accuracy: 0.7388
Epoch 2/100
176/176 [==============================] - 1s 4ms/step - loss: 0.4252 - accuracy: 0.7945
Epoch 3/100
176/176 [==============================] - 1s 3ms/step - loss: 0.4175 - accuracy: 0.8009
Epoch 4/100
176/176 [==============================] - 1s 4ms/step - loss: 0.4147 - accuracy: 0.8060
Epoch 5/100
176/176 [==============================] - 1s 3ms/step - loss: 0.4128 - accuracy: 0.8066
Epoch 6/100
176/176 [==============================] - 1s 3ms/step - loss: 0.4099 - accuracy: 0.8084
Epoch 7/100
176/176 [==============================] - 1s 4ms/step - loss: 0.4087 - accuracy: 0.8092
Epoch 8/100
176/176 [==============================] - 1s 4ms/step - loss: 0.4072 - accuracy: 0.8087
Epoch 9/100
176/176 [==============================] - 1s 3ms/step - loss: 0.4065 - accuracy: 0.8091
Epoch 10/100
176/176 [==============================] - 1s 4ms/step - loss: 0.4055 - accuracy: 0.8100
Epoch 11/100
176/176 [==============================] - 1s 3ms/step - loss: 0.4033 - accuracy: 0.8112
Epoch 12/100
176/176 [==============================] - 1s 4ms/step - loss: 0.4030 - accuracy: 0.8116
Epoch 13/100
176/176 [==============================] - 1s 4ms/step - loss: 0.4030 - accuracy: 0.8114
Epoch 14/100
176/176 [==============================] - 1s 3ms/step - loss: 0.4009 - accuracy: 0.8116
Epoch 15/100
176/176 [==============================] - 1s 4ms/step - loss: 0.4015 - accuracy: 0.8126
Epoch 16/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3985 - accuracy: 0.8128
Epoch 17/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3989 - accuracy: 0.8146
Epoch 18/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3972 - accuracy: 0.8133
Epoch 19/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3963 - accuracy: 0.8160
Epoch 20/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3965 - accuracy: 0.8164
Epoch 21/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3961 - accuracy: 0.8139
Epoch 22/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3942 - accuracy: 0.8183
Epoch 23/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3943 - accuracy: 0.8172
Epoch 24/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3940 - accuracy: 0.8160
Epoch 25/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3918 - accuracy: 0.8183
Epoch 26/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3917 - accuracy: 0.8194
Epoch 27/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3910 - accuracy: 0.8172
```

```
Epoch 28/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3908 - accuracy: 0.8196
Epoch 29/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3890 - accuracy: 0.8180
Epoch 30/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3896 - accuracy: 0.8178
Epoch 31/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3897 - accuracy: 0.8162
Epoch 32/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3886 - accuracy: 0.8192
Epoch 33/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3882 - accuracy: 0.8174
Epoch 34/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3873 - accuracy: 0.8180
Epoch 35/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3867 - accuracy: 0.8210
Epoch 36/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3856 - accuracy: 0.8201
Epoch 37/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3844 - accuracy: 0.8199
Epoch 38/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3852 - accuracy: 0.8199
Epoch 39/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3837 - accuracy: 0.8228
Epoch 40/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3839 - accuracy: 0.8204
Epoch 41/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3828 - accuracy: 0.8224
Epoch 42/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3824 - accuracy: 0.8231
Epoch 43/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3811 - accuracy: 0.8219
Epoch 44/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3811 - accuracy: 0.8238
Epoch 45/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3795 - accuracy: 0.8236
Epoch 46/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3792 - accuracy: 0.8249
Epoch 47/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3789 - accuracy: 0.8244
Epoch 48/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3773 - accuracy: 0.8244
Epoch 49/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3786 - accuracy: 0.8228
Epoch 50/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3765 - accuracy: 0.8252
Epoch 51/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3761 - accuracy: 0.8240
Epoch 52/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3758 - accuracy: 0.8274
Epoch 53/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3753 - accuracy: 0.8272
Epoch 54/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3735 - accuracy: 0.8281
Epoch 55/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3741 - accuracy: 0.8261
Epoch 56/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3722 - accuracy: 0.8279
Epoch 57/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3732 - accuracy: 0.8267
Epoch 58/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3716 - accuracy: 0.8290
Epoch 59/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3701 - accuracy: 0.8286
Epoch 60/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3715 - accuracy: 0.8256
Epoch 61/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3701 - accuracy: 0.8293
Epoch 62/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3702 - accuracy: 0.8308
Epoch 63/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3676 - accuracy: 0.8334
```

```
Epoch 64/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3692 - accuracy: 0.8309
Epoch 65/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3679 - accuracy: 0.8324
Epoch 66/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3671 - accuracy: 0.8299
Epoch 67/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3665 - accuracy: 0.8320
Epoch 68/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3659 - accuracy: 0.8295
Epoch 69/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3652 - accuracy: 0.8309
Epoch 70/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3644 - accuracy: 0.8354
Epoch 71/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3640 - accuracy: 0.8325
Epoch 72/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3636 - accuracy: 0.8313
Epoch 73/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3623 - accuracy: 0.8329
Epoch 74/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3619 - accuracy: 0.8352
Epoch 75/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3608 - accuracy: 0.8364
Epoch 76/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3607 - accuracy: 0.8345
Epoch 77/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3605 - accuracy: 0.8361
Epoch 78/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3594 - accuracy: 0.8341
Epoch 79/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3598 - accuracy: 0.8359
Epoch 80/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3579 - accuracy: 0.8359
Epoch 81/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3580 - accuracy: 0.8375
Epoch 82/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3575 - accuracy: 0.8350
Epoch 83/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3556 - accuracy: 0.8356
Epoch 84/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3562 - accuracy: 0.8382
Epoch 85/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3552 - accuracy: 0.8412
Epoch 86/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3565 - accuracy: 0.8389
Epoch 87/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3546 - accuracy: 0.8361
Epoch 88/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3551 - accuracy: 0.8379
Epoch 89/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3529 - accuracy: 0.8373
Epoch 90/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3533 - accuracy: 0.8373
Epoch 91/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3520 - accuracy: 0.8409
Epoch 92/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3514 - accuracy: 0.8375
Epoch 93/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3506 - accuracy: 0.8411
Epoch 94/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3496 - accuracy: 0.8405
Epoch 95/100
176/176 [==============================] - 1s 3ms/step - loss: 0.3506 - accuracy: 0.8395
Epoch 96/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3490 - accuracy: 0.8416
Epoch 97/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3482 - accuracy: 0.8405
Epoch 98/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3488 - accuracy: 0.8420
Epoch 99/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3476 - accuracy: 0.8416
```

```
Epoch 100/100
176/176 [==============================] - 1s 4ms/step - loss: 0.3464 - accuracy: 0.8432
```

Out[46]:

```
<keras.callbacks.History at 0x7f6b102e64d0>
```

In [47]:

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

```
44/44 [==============================] - 0s 3ms/step - loss: 0.4800 - accuracy: 0.7783
```

Out[47]:

```
[0.48004645109176636, 0.778251588344574]
```

In [48]:

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

Out[48]:

```
array([[0.04065612],
       [0.6170846 ],
       [0.01563833],
       [0.62529767],
       [0.33679527]], dtype=float32)
```

In [49]:

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

In [50]:

```
y_pred[:10]
```

Out[50]:

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

In [51]:

```
y_test[:10]
```

Out[51]:

```
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 [52]:

```
from sklearn.metrics import confusion_matrix , classification_report

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

```
              precision    recall  f1-score   support

           0       0.82      0.89      0.85       999
```

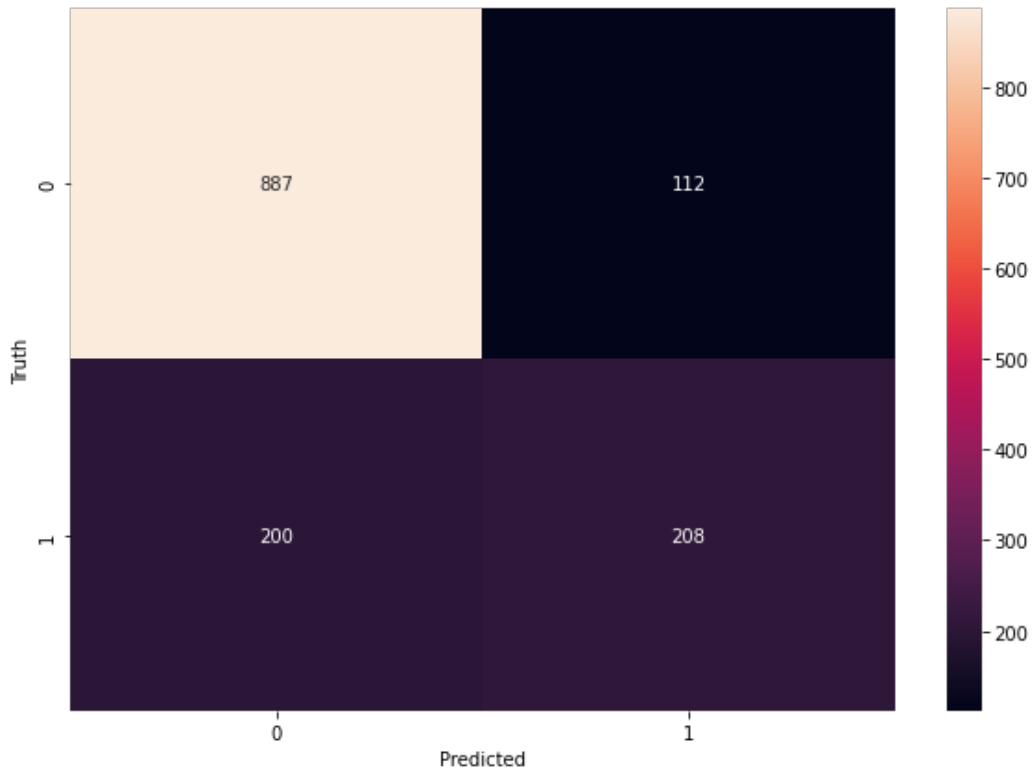
```
           1       0.65      0.51      0.57       408

    accuracy                           0.78      1407
   macro avg       0.73      0.70      0.71      1407
weighted avg       0.77      0.78      0.77      1407
```

In [53]:

```
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[53]:

```
Text(69.0, 0.5, 'Truth')
```



In [54]:

```
y_test.shape
```

Out[54]:

```
(1407,)
```

In [55]:

```
## Accuracy

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

Out[55]:

```
0.78
```

In [56]:

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

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

Out[56]:

0.83

In [57]:

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

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

Out[57]:

0.63

In [58]:

```
## Recall for 0 class

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

Out[58]:

0.86

In [59]:

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

Out[59]:

0.56

In [ ]:

```

```

In [ ]:

```

```