In [1]:
```
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
%matplotlib inline
```

In [2]:
```
df = pd.read_csv("Customer_Churn.csv")
df.head()
```

Out[2]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Inte
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	
4	9237-HQITU	Female	0	No	No	2	Yes	No	

5 rows × 21 columns

In [ ]:
```
# since we are predicting customer id is not required
```

In [4]:
```
df.drop('customerID',axis='columns',inplace=True)
df.dtypes #table heading along with their type
```

Out[4]:
```
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 [5]:
```
#convert string to numbers
```

In [6]:
```
df.TotalCharges.values
```

Out[6]:
```
array(['29.85', '1889.5', '108.15', ..., '346.45', '306.6', '6844.5'],
      dtype=object)
```

In [7]:
```
df.MonthlyCharges.values
```

Out[7]:
```
array([ 29.85,  56.95,  53.85, ...,  29.6 ,  74.4 , 105.65])
```

In [8]:
```
pd.to_numeric(df.TotalCharges)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\_libs\lib.py
x:2369, 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)
Cell In [8], line 1
----> 1 pd.to_numeric(df.TotalCharges)

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\tools\nu
meric.py:185, in to_numeric(arg, errors, downcast)
    183 coerce_numeric = errors not in ("ignore", "raise")
    184 try:
--> 185     values, _ = lib.maybe_convert_numeric(
    186         values, set(), coerce_numeric=coerce_numeric
    187     )
    188 except (ValueError, TypeError):
    189     if errors == "raise":

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\_libs\lib.py
x:2411, in pandas._libs.lib.maybe_convert_numeric()

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

In [9]:
```
#since there are some spaces in between , we will convert just the string and ignore t
```

In [10]:
```
pd.to_numeric(df.TotalCharges,errors='coerce')
```

Out[10]:
```
0         29.85
1       1889.50
2        108.15
3       1840.75
4        151.65
         ...   
7038    1990.50
7039    7362.90
7040     346.45
7041     306.60
7042    6844.50
Name: TotalCharges, Length: 7043, dtype: float64
```

In [11]:
```
#lets check for null values in terms of boolean
```

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]:
```
#when we put inside the dataframe df ,it will only show the columns which are true
```

In [14]:
```
df[pd.to_numeric(df.TotalCharges,errors='coerce').isnull()]
```

Out[14]:

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

In [15]:
```
#total counts
```

In [16]:
```
df[pd.to_numeric(df.TotalCharges,errors='coerce').isnull()].shape
```

Out[16]:
```
(11, 20)
```

In [17]:
```
df.shape
```

Out[17]:
```
(7043, 20)
```

In [18]:
```
#iloc interger location
# droping null values rows
```

In [19]:
```
df1 = df[df.TotalCharges!=' ']
df1.shape
```

Out[19]:
```
(7032, 20)
```

In [20]:
```
pd.to_numeric(df1.TotalCharges)
```

Out[20]:
```
0         29.85
1       1889.50
2        108.15
3       1840.75
4        151.65
         ...
7038    1990.50
7039    7362.90
7040     346.45
7041     306.60
7042    6844.50
Name: TotalCharges, Length: 7032, dtype: float64
```

In [21]:
```
df1.TotalCharges = pd.to_numeric(df1.TotalCharges)
```

```
C:\Users\Dell\AppData\Local\Temp\ipykernel_19940\973151263.py:1: SettingWithCopyWarni
ng:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  df1.TotalCharges = pd.to_numeric(df1.TotalCharges)
```

In [22]:
```
df1.TotalCharges.dtypes
```

Out[22]:
```
dtype('float64')
```

In [23]:
```
df1[df1.Churn=='No']
```

Out[23]:

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

5163 rows × 20 columns

In [24]:

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

Out[24]:

```
0        1
1       34
3       45
6       22
7       10
        ..
7037    72
7038    24
7039    72
7040    11
7042    66
Name: tenure, Length: 5163, dtype: int64
```

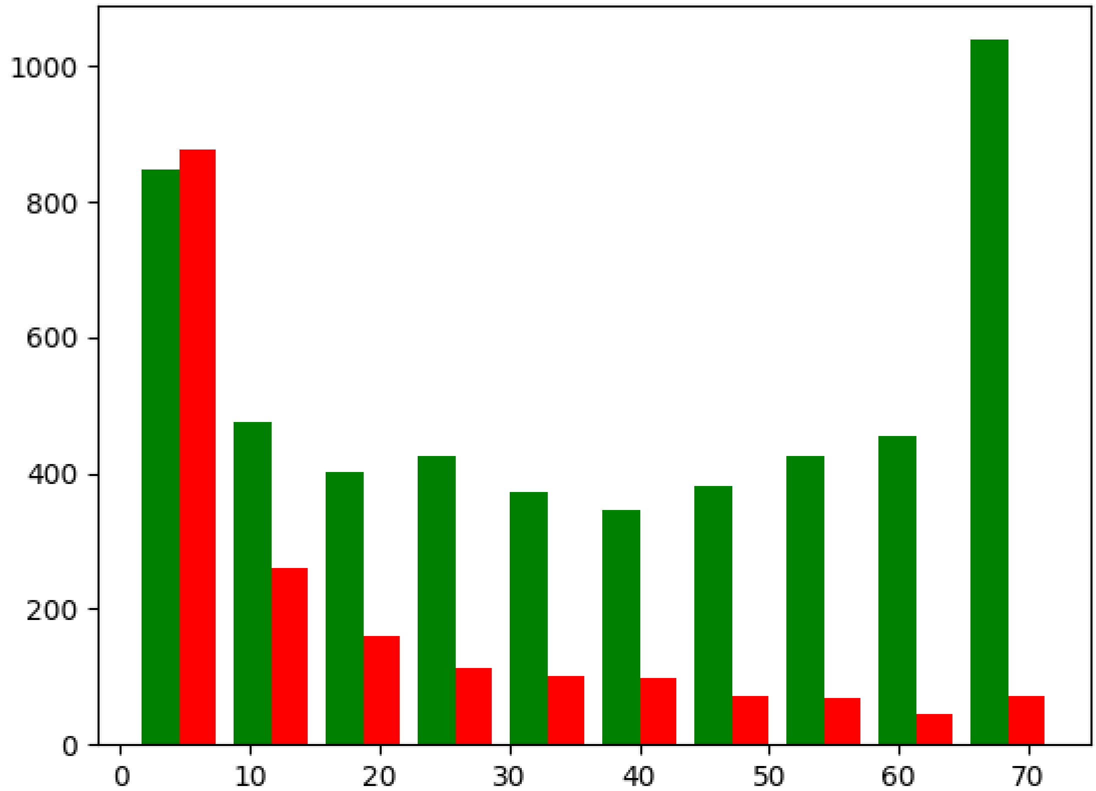
In [25]:

```
#visualization with tenure and churn
```

In [26]:

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

plt.hist([tenure_churn_no,tenure_churn_yes], color=['green','red'])
```

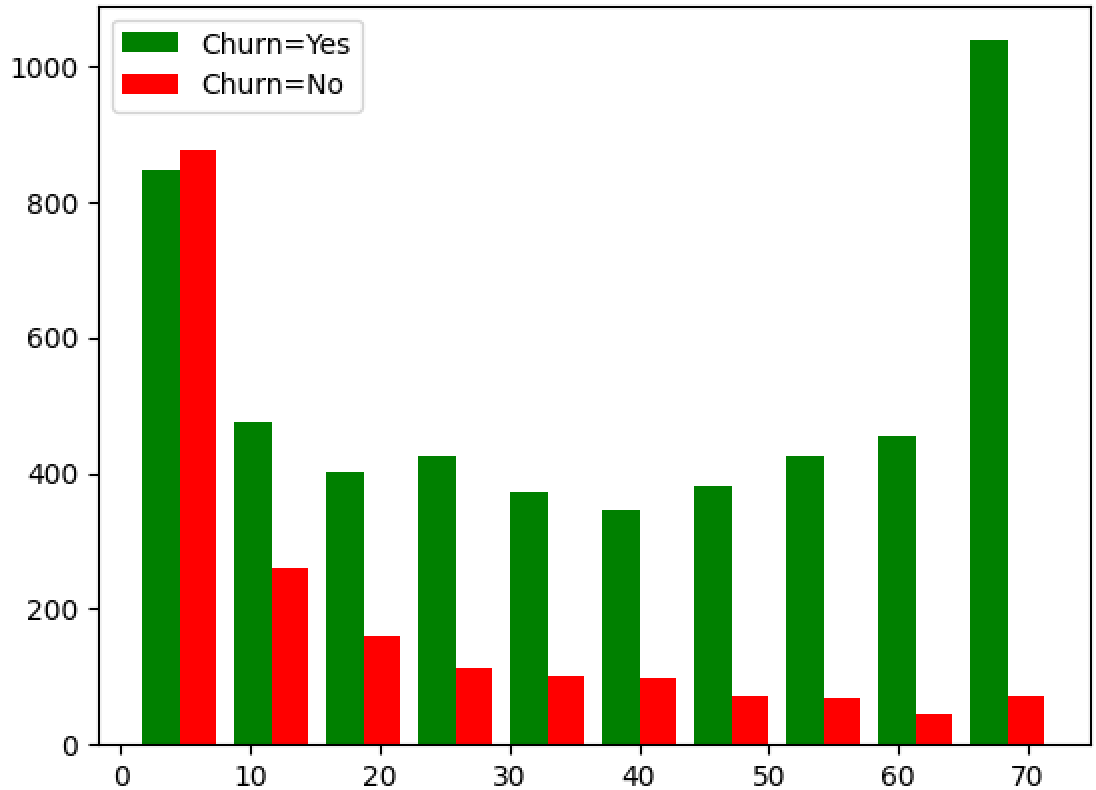
Out[26]:
```
(array([[ 847.,  476.,  402.,  424.,  371.,  346.,  380.,  425.,  455.,
         1037.],
        [ 877.,  259.,  159.,  114.,  102.,   98.,   72.,   70.,   46.,
           72.]]),
 array([ 1. ,  8.1, 15.2, 22.3, 29.4, 36.5, 43.6, 50.7, 57.8, 64.9, 72. ]),
 <a list of 2 BarContainer objects>)
```



In [27]:
```
# along with legend
```

In [28]:
```
tenure_churn_no = df1[df1.Churn=='No'].tenure
tenure_churn_yes = df1[df1.Churn=='Yes'].tenure

plt.hist([tenure_churn_no,tenure_churn_yes], color=['green','red'], label=['Churn=Yes'
plt.legend()
```

Out[28]:
```
<matplotlib.legend.Legend at 0x2ad0cc1f6d0>
```



In [29]:

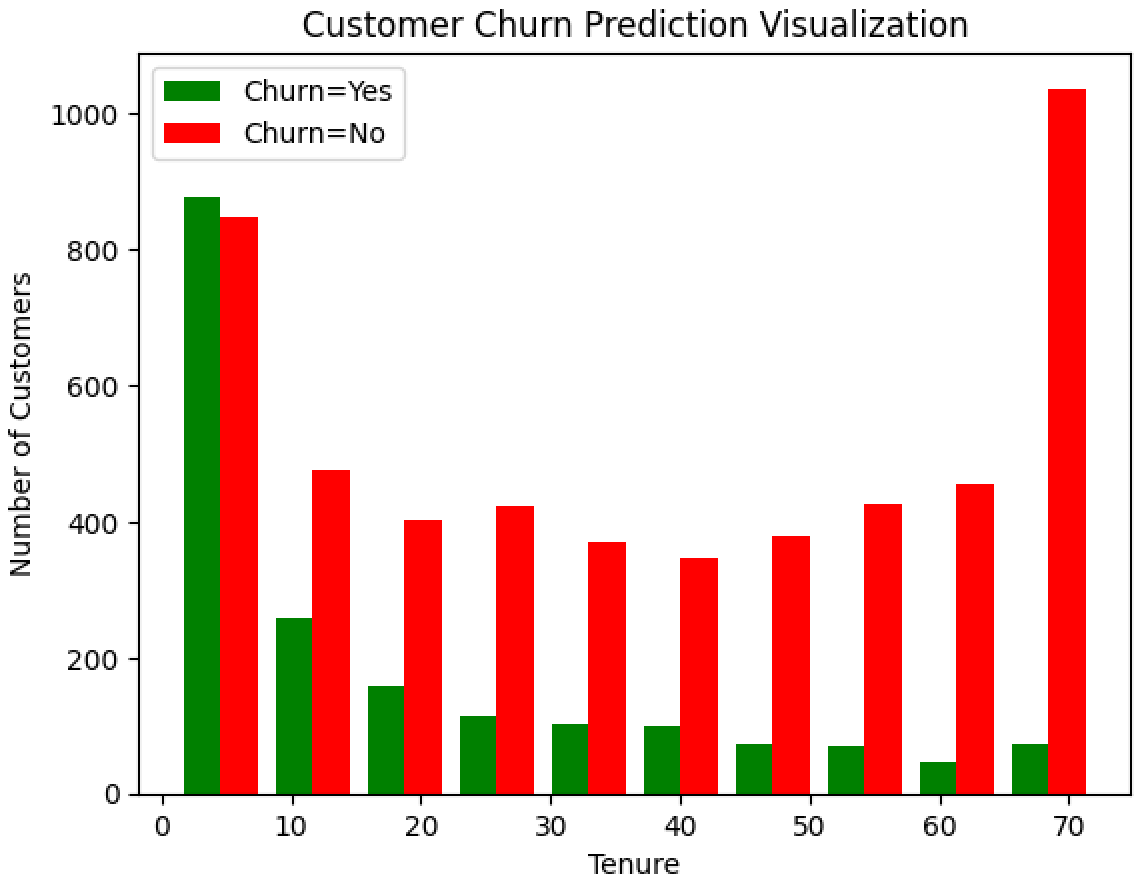
```
#along with x , y axis and title
```

In [30]:

```
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_yes,tenure_churn_no], color=['green','red'], label=['Churn=Yes'
plt.legend()
```

Out[30]:

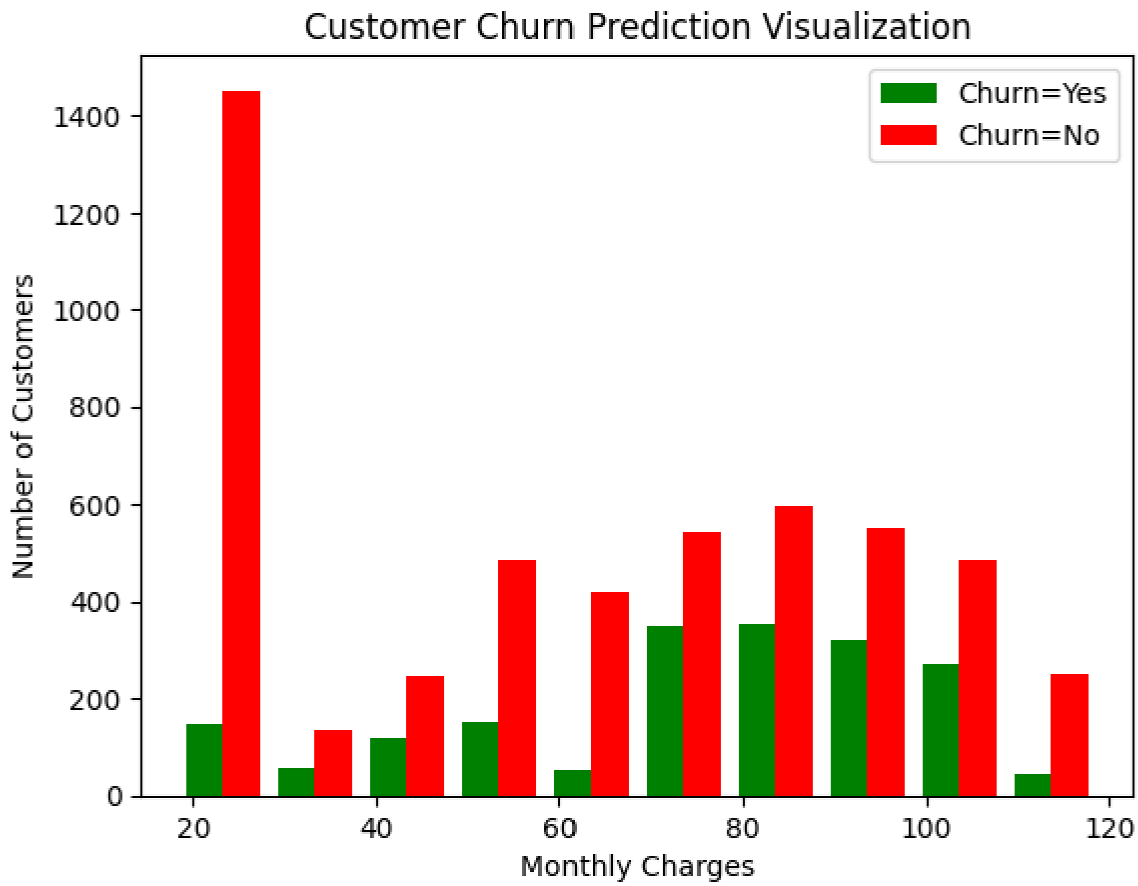
```
<matplotlib.legend.Legend at 0x2ad0cc1d510>
```



In [31]:
```
# for monthly Charges
```

In [32]:
```
tenure_churn_no = df1[df1.Churn=='No'].MonthlyCharges
tenure_churn_yes = df1[df1.Churn=='Yes'].MonthlyCharges

plt.xlabel('Monthly Charges')
plt.ylabel('Number of Customers')
plt.title('Customer Churn Prediction Visualization')

plt.hist([tenure_churn_yes,tenure_churn_no], color=['green','red'], label=['Churn=Yes'
plt.legend()
```

Out[32]:
```
<matplotlib.legend.Legend at 0x2ad0cc1f340>
```



In [33]:
```
#for loop for finding unique values
```

In [34]:
```
for column in df:
    print(f'{column} : {df[column].unique()}')
```

```
gender : ['Female' 'Male']
SeniorCitizen : [0 1]
Partner : ['Yes' 'No']
Dependents : ['No' 'Yes']
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  0
 39]
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)']
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 : ['No' 'Yes']
```

In [35]:
```
#for only object type of columns
```

In [36]:
```
for column in df:
    if df[column].dtypes=='object':
        print(f'{column} : {df[column].unique()}')
```

```
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)']
TotalCharges : ['29.85' '1889.5' '108.15' ... '346.45' '306.6' '6844.5']
Churn : ['No' 'Yes']
```

In [37]:
```
def print_unique_col_values(df):
    for column in df:
        if df[column].dtypes=='object':
            print(f'{column} : {df[column].unique()}')
```

In [38]:
```
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 [39]:
```
df1.replace('No internet service','No',inplace=True)
df1.replace('No phone service','No',inplace=True)
```

```
C:\Users\Dell\AppData\Local\Temp\ipykernel_19940\2045096646.py:1: SettingWithCopyWarn
ing:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  df1.replace('No internet service','No',inplace=True)
C:\Users\Dell\AppData\Local\Temp\ipykernel_19940\2045096646.py:2: SettingWithCopyWarn
ing:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  df1.replace('No phone service','No',inplace=True)
```

In [40]:

```
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 [41]:

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

```
C:\Users\Dell\AppData\Local\Temp\ipykernel_19940\1648037665.py:4: SettingWithCopyWarn
ing:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  df1[col].replace({'Yes': 1,'No': 0},inplace=True)
```

In [42]:

```
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 [43]:
```
df1['gender'].replace({'Female':1,'Male':0},inplace=True)
```

```
C:\Users\Dell\AppData\Local\Temp\ipykernel_19940\698335744.py:1: SettingWithCopyWarni
ng:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  df1['gender'].replace({'Female':1,'Male':0},inplace=True)
```

In [44]:
```
df1.gender.unique()
```

Out[44]:
```
array([1, 0], dtype=int64)
```

In [45]:
```
#for dates month to month - we have to use hot one encoding approach
#basically for one columns it will create multiple columns(3)
```

In [46]:
```
df2 = pd.get_dummies(data=df1, columns=['InternetService','Contract','PaymentMethod'])
df2.columns
```

Out[46]:
```
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 [47]:
```
df2.sample(5)
```

Out[47]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	OnlineSecurit
5331	0	0	1	1	44	0	0	
6441	1	0	1	1	17	1	0	
6958	1	0	1	1	13	1	0	
983	0	0	0	0	1	1	1	
494	0	0	0	0	1	1	0	

5 rows × 27 columns

In [51]:
```
df2.dtypes
```

Out[51]:
```
gender                                      int64
SeniorCitizen                               int64
Partner                                     int64
Dependents                                  int64
tenure                                    float64
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 [ ]:
```
# for scaling columns
#'tenure','MonthlyCharges','TotalCharges' since these are not in 0,1
#we make use of minmax scaler
```

In [49]:
```
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 [50]:
```
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 0.6014925
4]
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]
```

In [52]:
```python
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 [53]:
```python
X_train.shape
```

Out[53]:
```
(5625, 26)
```

In [54]:
```python
X_test.shape
```

Out[54]:
```
(1407, 26)
```

In [55]:
```python
X_train[:10]
```

Out[55]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	OnlineSecu
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 [58]:

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

Out[58]:

26

In [ ]:

```
#building a ANN in tensor flow
```

In [57]:

```
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 [==============================] - 5s 3ms/step - loss: 0.4986 - accuracy: 0.7
616
Epoch 2/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4270 - accuracy: 0.7
964
Epoch 3/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4195 - accuracy: 0.8
009
Epoch 4/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4155 - accuracy: 0.8
039
Epoch 5/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4129 - accuracy: 0.8
050
Epoch 6/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4111 - accuracy: 0.8
066
Epoch 7/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4098 - accuracy: 0.8
059
Epoch 8/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4091 - accuracy: 0.8
073
Epoch 9/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4078 - accuracy: 0.8
094
Epoch 10/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4060 - accuracy: 0.8
108
Epoch 11/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4060 - accuracy: 0.8
087
Epoch 12/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4049 - accuracy: 0.8
116
Epoch 13/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4036 - accuracy: 0.8
089
Epoch 14/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4030 - accuracy: 0.8
110
Epoch 15/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4015 - accuracy: 0.8
130
Epoch 16/100
176/176 [==============================] - 0s 2ms/step - loss: 0.4001 - accuracy: 0.8
142
Epoch 17/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3996 - accuracy: 0.8
137
Epoch 18/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3982 - accuracy: 0.8
142
Epoch 19/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3976 - accuracy: 0.8
135
Epoch 20/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3970 - accuracy: 0.8
130
```

```
Epoch 21/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3962 - accuracy: 0.8
117
Epoch 22/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3955 - accuracy: 0.8
153
Epoch 23/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3945 - accuracy: 0.8
180
Epoch 24/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3951 - accuracy: 0.8
153
Epoch 25/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3929 - accuracy: 0.8
156
Epoch 26/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3931 - accuracy: 0.8
148
Epoch 27/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3912 - accuracy: 0.8
167
Epoch 28/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3905 - accuracy: 0.8
188
Epoch 29/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3901 - accuracy: 0.8
171
Epoch 30/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3888 - accuracy: 0.8
174
Epoch 31/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3875 - accuracy: 0.8
196
Epoch 32/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3877 - accuracy: 0.8
199
Epoch 33/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3861 - accuracy: 0.8
208
Epoch 34/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3855 - accuracy: 0.8
178
Epoch 35/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3845 - accuracy: 0.8
208
Epoch 36/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3841 - accuracy: 0.8
188
Epoch 37/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3845 - accuracy: 0.8
187
Epoch 38/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3823 - accuracy: 0.8
220
Epoch 39/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3821 - accuracy: 0.8
212
Epoch 40/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3815 - accuracy: 0.8
228
```

```
Epoch 41/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3798 - accuracy: 0.8
212
Epoch 42/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3805 - accuracy: 0.8
229
Epoch 43/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3788 - accuracy: 0.8
222
Epoch 44/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3797 - accuracy: 0.8
212
Epoch 45/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3783 - accuracy: 0.8
238
Epoch 46/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3764 - accuracy: 0.8
238
Epoch 47/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3769 - accuracy: 0.8
247
Epoch 48/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3749 - accuracy: 0.8
286
Epoch 49/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3746 - accuracy: 0.8
213
Epoch 50/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3757 - accuracy: 0.8
245
Epoch 51/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3738 - accuracy: 0.8
258
Epoch 52/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3725 - accuracy: 0.8
256
Epoch 53/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3721 - accuracy: 0.8
268
Epoch 54/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3718 - accuracy: 0.8
279
Epoch 55/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3710 - accuracy: 0.8
254
Epoch 56/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3703 - accuracy: 0.8
277
Epoch 57/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3708 - accuracy: 0.8
295
Epoch 58/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3689 - accuracy: 0.8
252
Epoch 59/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3691 - accuracy: 0.8
286
Epoch 60/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3683 - accuracy: 0.8
277
```

```
Epoch 61/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3672 - accuracy: 0.8
267
Epoch 62/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3683 - accuracy: 0.8
313
Epoch 63/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3661 - accuracy: 0.8
284
Epoch 64/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3648 - accuracy: 0.8
309
Epoch 65/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3670 - accuracy: 0.8
279
Epoch 66/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3652 - accuracy: 0.8
302
Epoch 67/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3638 - accuracy: 0.8
272
Epoch 68/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3634 - accuracy: 0.8
329
Epoch 69/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3631 - accuracy: 0.8
284
Epoch 70/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3625 - accuracy: 0.8
331
Epoch 71/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3634 - accuracy: 0.8
299
Epoch 72/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3611 - accuracy: 0.8
332
Epoch 73/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3622 - accuracy: 0.8
325
Epoch 74/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3606 - accuracy: 0.8
327
Epoch 75/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3600 - accuracy: 0.8
340
Epoch 76/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3589 - accuracy: 0.8
361
Epoch 77/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3587 - accuracy: 0.8
377
Epoch 78/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3580 - accuracy: 0.8
356
Epoch 79/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3591 - accuracy: 0.8
341
Epoch 80/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3573 - accuracy: 0.8
391
```

```
Epoch 81/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3573 - accuracy: 0.8
356
Epoch 82/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3569 - accuracy: 0.8
343
Epoch 83/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3560 - accuracy: 0.8
348
Epoch 84/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3567 - accuracy: 0.8
363
Epoch 85/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3542 - accuracy: 0.8
361
Epoch 86/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3543 - accuracy: 0.8
388
Epoch 87/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3536 - accuracy: 0.8
404
Epoch 88/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3533 - accuracy: 0.8
377
Epoch 89/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3530 - accuracy: 0.8
416
Epoch 90/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3529 - accuracy: 0.8
363
Epoch 91/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3535 - accuracy: 0.8
370
Epoch 92/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3526 - accuracy: 0.8
400
Epoch 93/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3530 - accuracy: 0.8
396
Epoch 94/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3499 - accuracy: 0.8
393
Epoch 95/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3500 - accuracy: 0.8
400
Epoch 96/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3510 - accuracy: 0.8
402
Epoch 97/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3502 - accuracy: 0.8
396
Epoch 98/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3489 - accuracy: 0.8
411
Epoch 99/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3481 - accuracy: 0.8
414
Epoch 100/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3489 - accuracy: 0.8
389
```

Out[57]:
```
<keras.callbacks.History at 0x2ad28ee6770>
```

In [59]:
```
model.evaluate(X_test, y_test)
```

```
44/44 [==============================] - 0s 2ms/step - loss: 0.4973 - accuracy: 0.761
9
```

Out[59]:
```
[0.4973292052745819, 0.761904776096344]
```

In [60]:
```
yp = model.predict(X_test)
yp[:5]
```

```
44/44 [==============================] - 0s 2ms/step
```

Out[60]:
```
array([[0.18845144],
       [0.80721986],
       [0.00203073],
       [0.7945085 ],
       [0.5812861 ]], dtype=float32)
```

In [61]:
```
y_pred = []
for element in yp:
    if element > 0.5:
        y_pred.append(1)
    else:
        y_pred.append(0)
```

In [62]:
```
y_pred[:10]
```

Out[62]:
```
[0, 1, 0, 1, 1, 1, 0, 0, 0, 0]
```

In [63]:
```
y_test[:10]
```

Out[63]:
```
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 [64]:
```
from sklearn.metrics import confusion_matrix , classification_report

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

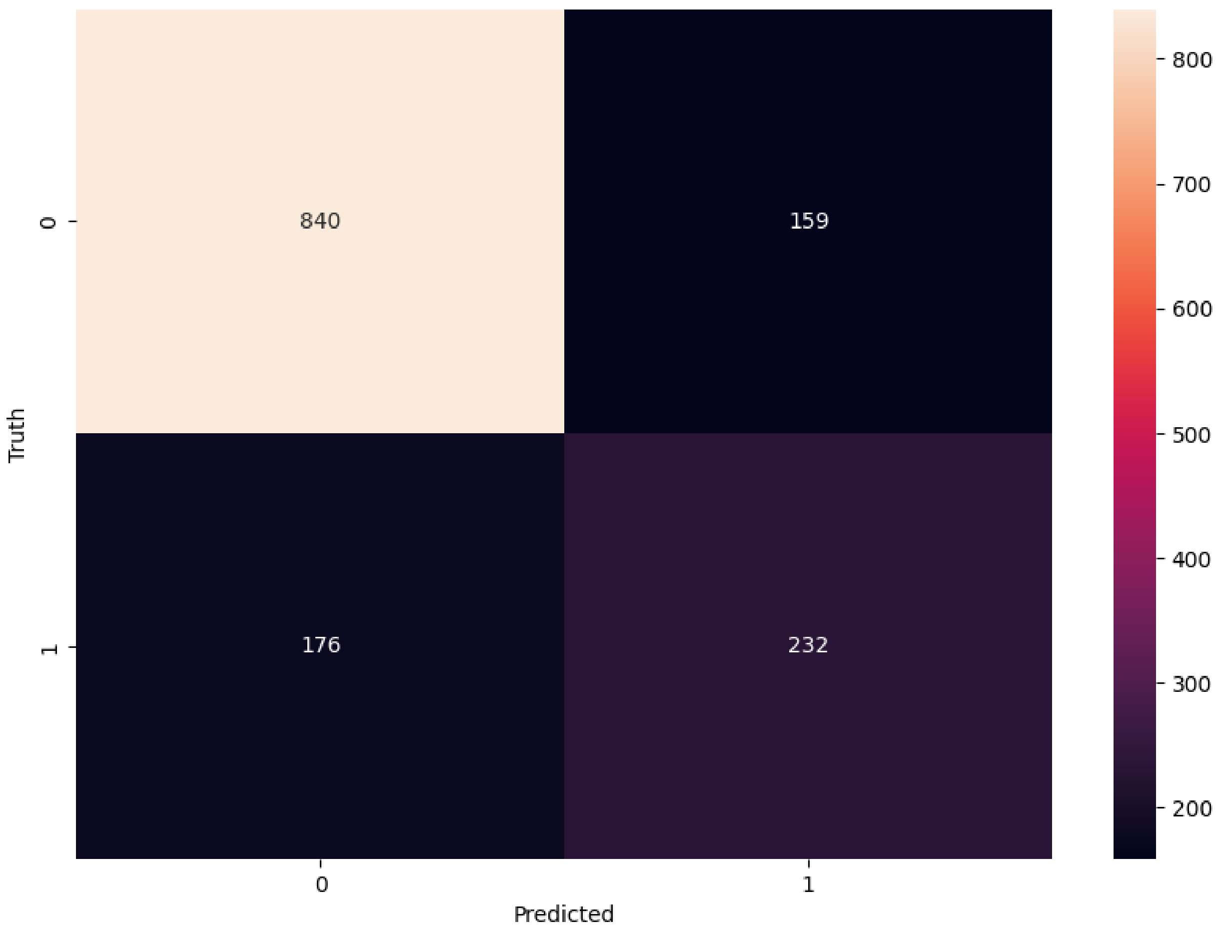
```
              precision    recall  f1-score   support

           0       0.83      0.84      0.83       999
           1       0.59      0.57      0.58       408

    accuracy                           0.76      1407
   macro avg       0.71      0.70      0.71      1407
weighted avg       0.76      0.76      0.76      1407
```

In [65]:
```
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[65]:

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



In [68]:
```
y_test.shape
```

Out[68]:
```
(1407,)
```

In [ ]:
```
#accuracy
```

In [69]:
```
round((862+229)/(862+229+137+179),2)
```

Out[69]:
```
0.78
```

In [70]:
```
#Precision for 0 class. i.e. Precision for customers who did not churn
```

In [71]:
```
round(862/(862+179),2)
```

Out[71]:
```
0.83
```

In [72]:
```
#Precision for 1 class. i.e. Precision for customers who actually churned
```

In [73]:
```
round(229/(229+137),2)
```

Out[73]: 0.63

In [74]:
```
#Recall for 0 class
```

In [75]:
```
round(862/(862+137),2)
```

Out[75]: 0.86

In [76]:
```
round(229/(229+179),2)
```

Out[76]: 0.56

In [ ]: