Data_analysis_Telecom Churn Prediction

October 28, 2023

1 Load necessary python modules

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split


from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

2 Load Dataset

[ ]:
```
df = pd.read_csv("./WA_Fn-UseC_-Telco-Customer-Churn.csv")
df.head()
```

[100]:
```
df.describe()
```

[100]:

	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000

```
25%        0.000000    9.000000      35.500000
50%        0.000000   29.000000      70.350000
75%        0.000000   55.000000      89.850000
max        1.000000   72.000000     118.750000
```

[101]:
```
# customerID: ID gender  SeniorCitizen   Partner   Dependents    tenure
# PhoneService      MultipleLines     InternetService     OnlineSecurity
# OnlineBackup     DeviceProtection   TechSupport       StreamingTV
# StreamingMovies    Contract     PaperlessBilling     PaymentMethod
# MonthlyCharges    TotalCharges    Churn
df.columns
```

[101]:
```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

3 basic Inspection on dataset

[102]:
```
# Remove/drop unwanted columns
df.drop(columns=["customerID"],inplace=True)
#columns/feature names
print(df.columns)
#check the shape rows and columns
print(df.shape)
# Checking the data types of all the columns
print(df.dtypes)
```

```
Index(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure',
       'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity',
       'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',
       'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod',
       'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
(7043, 20)
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
```

[103]:

```python
#Null / Nan Values
print (df.isnull().sum())
print (df.isnull().value_counts())
print ('------')
#Balanced / Imbalanced Dataset
print (df["Churn"].value_counts(normalize=True))
#describe
print (df.describe())
print (df.info())

# Observations

# Dataset has rows/sample=7043 ,Coulmns/features=20
# Dataset - features: gender', 'SeniorCitizen', 'Partner', 'Dependents',␣
↪'tenure', 'PhoneService', 'MultipleLines', 'InternetService',␣
↪'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',␣
↪'StreamingTV','StreamingMovies', 'Contract', 'PaperlessBilling',␣
↪'PaymentMethod','MonthlyCharges', 'TotalCharges', 'Churn'
# Churn is dependent variable.
# It is imbalanced Dataset
# Dataset did not have - Null / Nan Values
```

```
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
```

```
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
gender  SeniorCitizen  Partner  Dependents  tenure  PhoneService  MultipleLines
InternetService  OnlineSecurity  OnlineBackup  DeviceProtection  TechSupport
StreamingTV  StreamingMovies  Contract  PaperlessBilling  PaymentMethod
MonthlyCharges  TotalCharges  Churn
False   False          False    False       False   False         False
False            False           False         False            False
False        False            False     False             False          False
False         False    7043
Name: count, dtype: int64
------
Churn
No     0.73463
Yes    0.26537
Name: proportion, dtype: float64
       SeniorCitizen       tenure  MonthlyCharges
count    7043.000000  7043.000000     7043.000000
mean        0.162147    32.371149       64.761692
std         0.368612    24.559481       30.090047
min         0.000000     0.000000       18.250000
25%         0.000000     9.000000       35.500000
50%         0.000000    29.000000       70.350000
75%         0.000000    55.000000       89.850000
max         1.000000    72.000000      118.750000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   gender            7043 non-null   object
 1   SeniorCitizen     7043 non-null   int64
 2   Partner           7043 non-null   object
 3   Dependents        7043 non-null   object
 4   tenure            7043 non-null   int64
 5   PhoneService      7043 non-null   object
 6   MultipleLines     7043 non-null   object
 7   InternetService   7043 non-null   object
 8   OnlineSecurity    7043 non-null   object
 9   OnlineBackup      7043 non-null   object
 10  DeviceProtection  7043 non-null   object
```

```
 11  TechSupport       7043 non-null   object
 12  StreamingTV       7043 non-null   object
 13  StreamingMovies   7043 non-null   object
 14  Contract          7043 non-null   object
 15  PaperlessBilling  7043 non-null   object
 16  PaymentMethod     7043 non-null   object
 17  MonthlyCharges    7043 non-null   float64
 18  TotalCharges      7043 non-null   object
 19  Churn             7043 non-null   object
dtypes: float64(1), int64(2), object(17)
memory usage: 1.1+ MB
None
```

[104]:
```python
# Convert non numerical data to numerical
df.SeniorCitizen = df.SeniorCitizen.apply(lambda x: "No" if x==0 else "Yes")
print (df["SeniorCitizen"])
# df["SeniorCitizen"] = df.SeniorCitizen.apply(lambda x: "No" if x==0 else "Yes")
df["TotalCharges"]=pd.to_numeric(df.TotalCharges,errors='coerce')
df["TotalCharges"]
```

```
0        No
1        No
2        No
3        No
4        No
       ...
7038     No
7039     No
7040     No
7041    Yes
7042     No
Name: SeniorCitizen, Length: 7043, dtype: object
```

[104]:
```
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
```

[105]:
```
# Return unique values based on a hash table.
for col in df:
    if (df[col].dtype=="object"):
        print(col,":",df[col].unique())
```

```
gender : ['Female' 'Male']
SeniorCitizen : ['No' 'Yes']
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']
```

[106]:
```
for col in df:
    if (df[col].dtype=="object") and (df[col].nunique()==3):
        df[col] = df[col].apply(lambda x: "No" if x=="No internet service" else␣
↪x)
```

[107]:
```
# Return unique values based on a hash table.

for col in df:
    if (df[col].dtype=="object"):
        print(col,":",df[col].unique())
```

```
gender : ['Female' 'Male']
SeniorCitizen : ['No' 'Yes']
Partner : ['Yes' 'No']
Dependents : ['No' 'Yes']
PhoneService : ['No' 'Yes']
MultipleLines : ['No phone service' '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']
```

[108]:
```
df["MultipleLines"] = df["MultipleLines"].apply(lambda x: "No" if x=="No phone␣
↪service" else x)
```

[109]:
```
for col in df:
    if (df[col].dtype=="object"):
        print(col,":",df[col].unique())
```

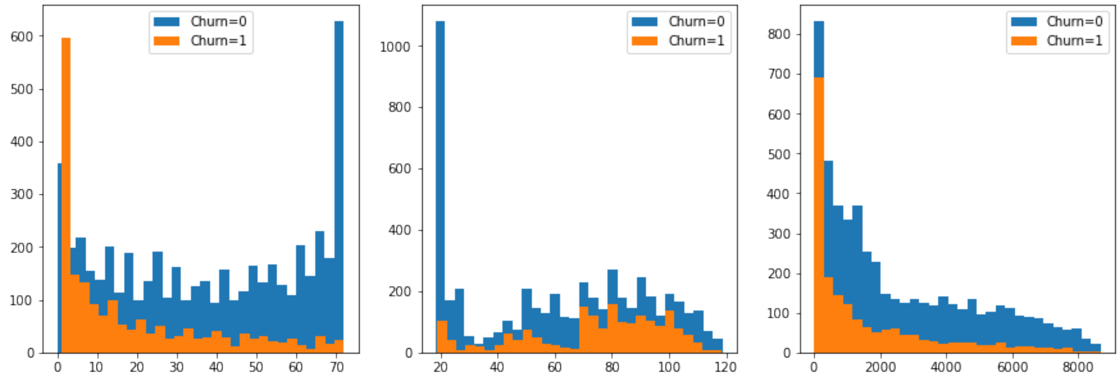
```
gender : ['Female' 'Male']
SeniorCitizen : ['No' 'Yes']
Partner : ['Yes' 'No']
Dependents : ['No' 'Yes']
PhoneService : ['No' 'Yes']
MultipleLines : ['No' 'Yes']
InternetService : ['DSL' 'Fiber optic' 'No']
OnlineSecurity : ['No' 'Yes']
OnlineBackup : ['Yes' 'No']
DeviceProtection : ['No' 'Yes']
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']
```

4 Data Analysis, Visualization and Interpretation

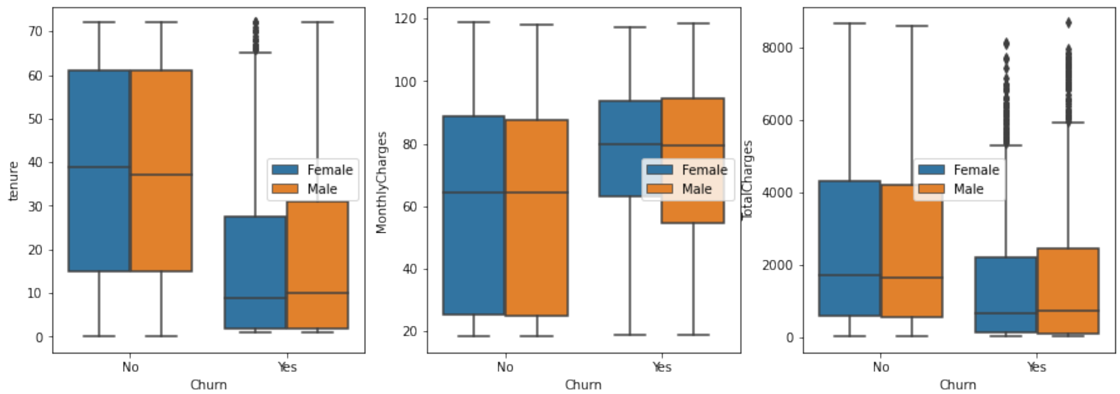
[110]:
```
con_vara = [i for i in df.columns if df[i].dtype != "object"]
print(con_vara)
```

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

[111]:
```
# Plot univariate or bivariate histograms to show distributions of datasets.
fig,ax = plt.subplots(1,3,figsize=(15,5))
for i,x in enumerate(con_vara):
    ax[i].hist(df[x][df.Churn=="No"],label="Churn=0",bins=30)
    ax[i].hist(df[x][df.Churn=="Yes"],label="Churn=1",bins=30)
    ax[i].legend()
```



[112]:
```
# Box-Plots: continuous variable
fig,ax = plt.subplots(1,3,figsize=(15,5))
for i,x in enumerate(con_vara):
    sns.boxplot(x=df.Churn, y=df[x] ,ax=ax[i],hue=df.gender)
    ax[i].legend()
```

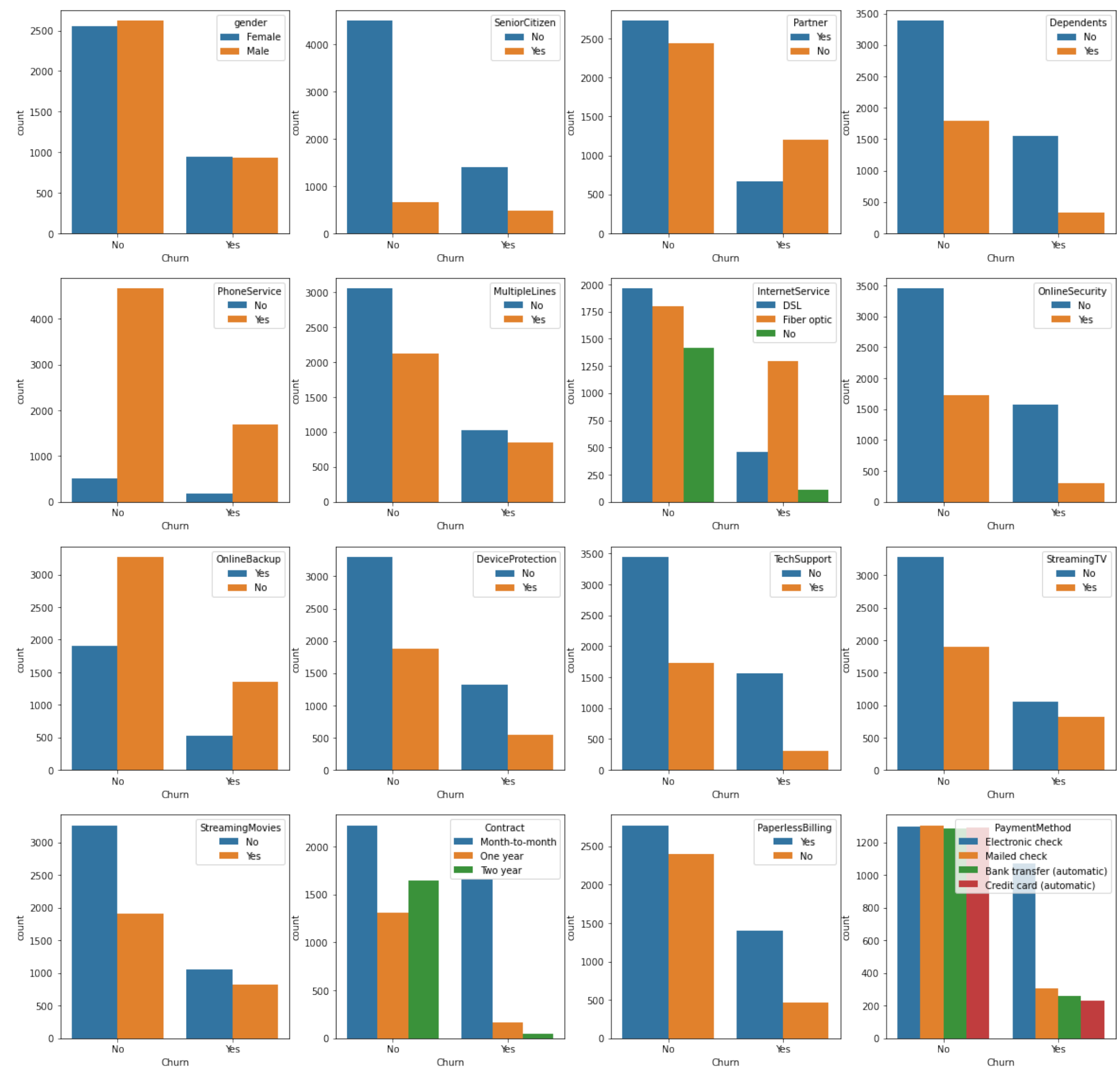


[113]:
```
# Observations

# Total Charges has outliers
# Tenure has outliers
# Monthyly charges did not have outliers
```

[114]:
```
# Count-Plots: categorical variable

cat_vara = [i for i in df.columns if df[i].dtype == "object"]
print(cat_vara)
cat_vara.pop()
print(cat_vara)
```

```
['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'PhoneService',
'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup',
'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
'PaperlessBilling', 'PaymentMethod', 'Churn']
['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'PhoneService',
'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup',
'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
'PaperlessBilling', 'PaymentMethod']
```

[115]:
```python
fig,ax = plt.subplots(4,4,figsize=(20,20))
for axi,x in zip(ax.flat,cat_vara):
    sns.countplot(x=df.Churn,hue=df[x],ax=axi)
```



[116]:
```
# Observations

# In gender , female tends more to churn - leave /exit the service
# In Age - Young/Youth tends to churn - leave /exit the service
# In partener - Individual tends to churn - leave /exit the service
# Phone Service - Havving - tends to churn - leave /exit the service
# Month to Month -Contract - tends to churn - leave /exit the service
```

5 Categorical Data Encoding

[117]:
```
le = LabelEncoder()
for col in df:
    if (df[col].dtype=="object") and (df[col].nunique()==2):
        df[col] = le.fit_transform(df[col])
```

[118]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   gender            7043 non-null   int64
 1   SeniorCitizen     7043 non-null   int64
 2   Partner           7043 non-null   int64
 3   Dependents        7043 non-null   int64
 4   tenure            7043 non-null   int64
 5   PhoneService      7043 non-null   int64
 6   MultipleLines     7043 non-null   int64
 7   InternetService   7043 non-null   object
 8   OnlineSecurity    7043 non-null   int64
 9   OnlineBackup      7043 non-null   int64
 10  DeviceProtection  7043 non-null   int64
 11  TechSupport       7043 non-null   int64
 12  StreamingTV       7043 non-null   int64
 13  StreamingMovies   7043 non-null   int64
 14  Contract          7043 non-null   object
 15  PaperlessBilling  7043 non-null   int64
 16  PaymentMethod     7043 non-null   object
 17  MonthlyCharges    7043 non-null   float64
 18  TotalCharges      7032 non-null   float64
 19  Churn             7043 non-null   int64
dtypes: float64(2), int64(15), object(3)
memory usage: 1.1+ MB
```

[119]:
```
# Convert categorical variable into dummy/indicator variables.
df = pd.get_dummies(df,columns=[i for i in df.columns if df[i].
 ↪dtypes=='object'],drop_first=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 24 columns):
 #   Column                                 Non-Null Count  Dtype
---  ------                                 --------------  -----
 0   gender                                 7043 non-null   int64
 1   SeniorCitizen                          7043 non-null   int64
 2   Partner                                7043 non-null   int64
 3   Dependents                             7043 non-null   int64
 4   tenure                                 7043 non-null   int64
 5   PhoneService                           7043 non-null   int64
 6   MultipleLines                          7043 non-null   int64
 7   OnlineSecurity                         7043 non-null   int64
 8   OnlineBackup                           7043 non-null   int64
 9   DeviceProtection                       7043 non-null   int64
 10  TechSupport                            7043 non-null   int64
 11  StreamingTV                            7043 non-null   int64
 12  StreamingMovies                        7043 non-null   int64
 13  PaperlessBilling                       7043 non-null   int64
 14  MonthlyCharges                         7043 non-null   float64
 15  TotalCharges                           7032 non-null   float64
 16  Churn                                  7043 non-null   int64
 17  InternetService_Fiber optic            7043 non-null   bool
 18  InternetService_No                     7043 non-null   bool
 19  Contract_One year                      7043 non-null   bool
 20  Contract_Two year                      7043 non-null   bool
 21  PaymentMethod_Credit card (automatic)  7043 non-null   bool
 22  PaymentMethod_Electronic check         7043 non-null   bool
 23  PaymentMethod_Mailed check             7043 non-null   bool
dtypes: bool(7), float64(2), int64(15)
memory usage: 983.7 KB
```

6 Scalling and Spliting

[120]:
```
df.dropna(inplace=True)
```

[121]:
```
from sklearn.utils import resample
df_majority = df[(df["Churn"]==0)]
df_minority = df[(df["Churn"]==1)]
```

```
df_minority_upsampled =␣
↪resample(df_minority,replace=True,n_samples=5000,random_state=42)
df = pd.concat([df_minority_upsampled,df_majority])
df["Churn"].value_counts()
```

[121]:
```
Churn
0    5163
1    5000
Name: count, dtype: int64
```

[122]:
```
# Standardize features by removing the mean and scaling to unit variance
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit_transform(df)
```

[122]:
```
array([[-1.01757045, -0.4880392 ,  1.11622885, …, -0.47765793,
         1.19638611, -0.51340358],
       [-1.01757045, -0.4880392 , -0.89587364, …, -0.47765793,
         1.19638611, -0.51340358],
       [ 0.98273294, -0.4880392 , -0.89587364, …, -0.47765793,
         1.19638611, -0.51340358],
       …,
       [-1.01757045, -0.4880392 ,  1.11622885, …,  2.0935484 ,
        -0.83585056, -0.51340358],
       [-1.01757045, -0.4880392 ,  1.11622885, …, -0.47765793,
         1.19638611, -0.51340358],
       [ 0.98273294, -0.4880392 , -0.89587364, …, -0.47765793,
        -0.83585056, -0.51340358]])
```

[123]:
```
# Split arrays or matrices into random train and test subsets.
X=df.drop(['Churn'],axis='columns')
Y=df['Churn']
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.30)
print("train data length:",len(X_train),'---',X_train.shape)
print("test data length:",len(X_test),'---',X_test.shape)
len(X_train.columns)
```

```
train data length: 7114 --- (7114, 23)
test data length: 3049 --- (3049, 23)
```

[123]:
```
23
```

7 Model - NN

[124]:

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

X_train=np.asarray(X_train).astype(np.float)
Y_train=np.asarray(Y_train).astype(np.float)

X_test=np.asarray(X_test).astype(np.float)
Y_test=np.asarray(Y_test).astype(np.float)

model = keras.Sequential([
    keras.layers.Dense(64, input_shape=(23,), activation='relu'),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])


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

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

```
<ipython-input-124-1d647ec4671e>:4: DeprecationWarning: `np.float` is a
deprecated alias for the builtin `float`. To silence this warning, use `float`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  X_train=np.asarray(X_train).astype(np.float)
<ipython-input-124-1d647ec4671e>:5: DeprecationWarning: `np.float` is a
deprecated alias for the builtin `float`. To silence this warning, use `float`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  Y_train=np.asarray(Y_train).astype(np.float)
<ipython-input-124-1d647ec4671e>:7: DeprecationWarning: `np.float` is a
deprecated alias for the builtin `float`. To silence this warning, use `float`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  X_test=np.asarray(X_test).astype(np.float)
<ipython-input-124-1d647ec4671e>:8: DeprecationWarning: `np.float` is a
deprecated alias for the builtin `float`. To silence this warning, use `float`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.float64` here.
```

```
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  Y_test=np.asarray(Y_test).astype(np.float)

Epoch 1/100
223/223 [==============================] - 1s 3ms/step - loss: 2.8008 -
accuracy: 0.6594
Epoch 2/100
223/223 [==============================] - 1s 3ms/step - loss: 1.1821 -
accuracy: 0.6934
Epoch 3/100
223/223 [==============================] - 1s 3ms/step - loss: 1.2583 -
accuracy: 0.7017
Epoch 4/100
223/223 [==============================] - 1s 3ms/step - loss: 1.4689 -
accuracy: 0.6916
Epoch 5/100
223/223 [==============================] - 1s 3ms/step - loss: 1.0398 -
accuracy: 0.7114
Epoch 6/100
223/223 [==============================] - 1s 3ms/step - loss: 1.2636 -
accuracy: 0.6986
Epoch 7/100
223/223 [==============================] - 1s 3ms/step - loss: 1.3960 -
accuracy: 0.6889
Epoch 8/100
223/223 [==============================] - 1s 4ms/step - loss: 1.0429 -
accuracy: 0.7179
Epoch 9/100
223/223 [==============================] - 1s 3ms/step - loss: 1.0566 -
accuracy: 0.7072
Epoch 10/100
223/223 [==============================] - 1s 3ms/step - loss: 1.0990 -
accuracy: 0.7076
Epoch 11/100
223/223 [==============================] - 1s 3ms/step - loss: 1.3831 -
accuracy: 0.6990
Epoch 12/100
223/223 [==============================] - 1s 3ms/step - loss: 0.9564 -
accuracy: 0.7200
Epoch 13/100
223/223 [==============================] - 1s 3ms/step - loss: 0.7572 -
accuracy: 0.7297
Epoch 14/100
223/223 [==============================] - 1s 4ms/step - loss: 0.9012 -
accuracy: 0.7175
Epoch 15/100
223/223 [==============================] - 1s 3ms/step - loss: 0.7188 -
```

```
accuracy: 0.7333
Epoch 16/100
223/223 [==============================] - 1s 3ms/step - loss: 0.8022 -
accuracy: 0.7256
Epoch 17/100
223/223 [==============================] - 1s 3ms/step - loss: 0.7762 -
accuracy: 0.7267
Epoch 18/100
223/223 [==============================] - 1s 3ms/step - loss: 0.8247 -
accuracy: 0.7300
Epoch 19/100
223/223 [==============================] - 1s 3ms/step - loss: 0.8345 -
accuracy: 0.7220
Epoch 20/100
223/223 [==============================] - 1s 3ms/step - loss: 0.7902 -
accuracy: 0.7269
Epoch 21/100
223/223 [==============================] - 1s 5ms/step - loss: 0.8796 -
accuracy: 0.7272
Epoch 22/100
223/223 [==============================] - 1s 4ms/step - loss: 0.9689 -
accuracy: 0.7144
Epoch 23/100
223/223 [==============================] - 1s 4ms/step - loss: 0.7317 -
accuracy: 0.7304
Epoch 24/100
223/223 [==============================] - 1s 4ms/step - loss: 0.8721 -
accuracy: 0.7176
Epoch 25/100
223/223 [==============================] - 1s 3ms/step - loss: 1.0080 -
accuracy: 0.7177
Epoch 26/100
223/223 [==============================] - 1s 3ms/step - loss: 0.6893 -
accuracy: 0.7369
Epoch 27/100
223/223 [==============================] - 1s 3ms/step - loss: 0.9689 -
accuracy: 0.7052
Epoch 28/100
223/223 [==============================] - 1s 4ms/step - loss: 0.8744 -
accuracy: 0.7168
Epoch 29/100
223/223 [==============================] - 1s 3ms/step - loss: 0.6769 -
accuracy: 0.7369
Epoch 30/100
223/223 [==============================] - 1s 3ms/step - loss: 0.9194 -
accuracy: 0.7151
Epoch 31/100
223/223 [==============================] - 1s 3ms/step - loss: 0.6579 -
```

```
accuracy: 0.7402
Epoch 32/100
223/223 [==============================] - 1s 3ms/step - loss: 0.6462 -
accuracy: 0.7336
Epoch 33/100
223/223 [==============================] - 1s 3ms/step - loss: 0.8743 -
accuracy: 0.7248
Epoch 34/100
223/223 [==============================] - 1s 3ms/step - loss: 0.6137 -
accuracy: 0.7398
Epoch 35/100
223/223 [==============================] - 1s 3ms/step - loss: 0.6797 -
accuracy: 0.7353
Epoch 36/100
223/223 [==============================] - 1s 3ms/step - loss: 0.6797 -
accuracy: 0.7260
Epoch 37/100
223/223 [==============================] - 1s 4ms/step - loss: 0.7121 -
accuracy: 0.7288
Epoch 38/100
223/223 [==============================] - 1s 3ms/step - loss: 0.6440 -
accuracy: 0.7367
Epoch 39/100
223/223 [==============================] - 1s 3ms/step - loss: 0.6845 -
accuracy: 0.7398
Epoch 40/100
223/223 [==============================] - 1s 3ms/step - loss: 0.7300 -
accuracy: 0.7315
Epoch 41/100
223/223 [==============================] - 1s 4ms/step - loss: 0.7820 -
accuracy: 0.7252
Epoch 42/100
223/223 [==============================] - 1s 4ms/step - loss: 0.7677 -
accuracy: 0.7229
Epoch 43/100
223/223 [==============================] - 1s 4ms/step - loss: 0.6110 -
accuracy: 0.7432
Epoch 44/100
223/223 [==============================] - 1s 4ms/step - loss: 0.6949 -
accuracy: 0.7269
Epoch 45/100
223/223 [==============================] - 1s 5ms/step - loss: 0.5969 -
accuracy: 0.7446
Epoch 46/100
223/223 [==============================] - 1s 4ms/step - loss: 0.6416 -
accuracy: 0.7409
Epoch 47/100
223/223 [==============================] - 1s 4ms/step - loss: 0.6768 -
```

```
accuracy: 0.7364
Epoch 48/100
223/223 [==============================] - 1s 5ms/step - loss: 0.6502 -
accuracy: 0.7357
Epoch 49/100
223/223 [==============================] - 1s 5ms/step - loss: 0.6329 -
accuracy: 0.7378: 0s - loss: 0
Epoch 50/100
223/223 [==============================] - 1s 4ms/step - loss: 0.6217 -
accuracy: 0.7390
Epoch 51/100
223/223 [==============================] - 1s 4ms/step - loss: 0.6596 -
accuracy: 0.7360
Epoch 52/100
223/223 [==============================] - 1s 4ms/step - loss: 0.5449 -
accuracy: 0.7487
Epoch 53/100
223/223 [==============================] - 1s 5ms/step - loss: 0.6683 -
accuracy: 0.7356
Epoch 54/100
223/223 [==============================] - 1s 4ms/step - loss: 0.7865 -
accuracy: 0.7198
Epoch 55/100
223/223 [==============================] - 1s 6ms/step - loss: 0.6289 -
accuracy: 0.7340
Epoch 56/100
223/223 [==============================] - 1s 5ms/step - loss: 0.5609 -
accuracy: 0.7459
Epoch 57/100
223/223 [==============================] - 1s 5ms/step - loss: 0.6208 -
accuracy: 0.7421
Epoch 58/100
223/223 [==============================] - 1s 6ms/step - loss: 0.5602 -
accuracy: 0.7474
Epoch 59/100
223/223 [==============================] - 1s 6ms/step - loss: 0.5782 -
accuracy: 0.7512
Epoch 60/100
223/223 [==============================] - 1s 4ms/step - loss: 0.5363 -
accuracy: 0.7550
Epoch 61/100
223/223 [==============================] - 1s 5ms/step - loss: 0.6426 -
accuracy: 0.7300
Epoch 62/100
223/223 [==============================] - 1s 5ms/step - loss: 0.5785 -
accuracy: 0.7412
Epoch 63/100
223/223 [==============================] - 1s 5ms/step - loss: 0.5625 -
```

```
accuracy: 0.7470
Epoch 64/100
223/223 [==============================] - 1s 5ms/step - loss: 0.5864 -
accuracy: 0.7429
Epoch 65/100
223/223 [==============================] - 1s 6ms/step - loss: 0.5684 -
accuracy: 0.7489: 0s - loss: 0.5844 - accuracy: 0.74 - ETA: 0s - loss: 0.5844 -
accura
Epoch 66/100
223/223 [==============================] - 1s 5ms/step - loss: 0.5847 -
accuracy: 0.7435
Epoch 67/100
223/223 [==============================] - 1s 5ms/step - loss: 0.5520 -
accuracy: 0.7464
Epoch 68/100
223/223 [==============================] - 1s 5ms/step - loss: 0.5783 -
accuracy: 0.7513
Epoch 69/100
223/223 [==============================] - 1s 5ms/step - loss: 0.5518 -
accuracy: 0.7432
Epoch 70/100
223/223 [==============================] - 1s 6ms/step - loss: 0.5442 -
accuracy: 0.7502
Epoch 71/100
223/223 [==============================] - 1s 5ms/step - loss: 0.5836 -
accuracy: 0.7387
Epoch 72/100
223/223 [==============================] - 1s 5ms/step - loss: 0.5981 -
accuracy: 0.7405
Epoch 73/100
223/223 [==============================] - 1s 5ms/step - loss: 0.5587 -
accuracy: 0.7440
Epoch 74/100
223/223 [==============================] - 1s 5ms/step - loss: 0.5414 -
accuracy: 0.7487
Epoch 75/100
223/223 [==============================] - 1s 5ms/step - loss: 0.5476 -
accuracy: 0.7504
Epoch 76/100
223/223 [==============================] - 1s 5ms/step - loss: 0.5432 -
accuracy: 0.7513
Epoch 77/100
223/223 [==============================] - 1s 5ms/step - loss: 0.5344 -
accuracy: 0.7504
Epoch 78/100
223/223 [==============================] - 1s 6ms/step - loss: 0.5539 -
accuracy: 0.7475
Epoch 79/100
```

```
223/223 [==============================] - 1s 5ms/step - loss: 0.5266 -
accuracy: 0.7551
Epoch 80/100
223/223 [==============================] - 1s 3ms/step - loss: 0.5822 -
accuracy: 0.7461
Epoch 81/100
223/223 [==============================] - 1s 3ms/step - loss: 0.5891 -
accuracy: 0.7423
Epoch 82/100
223/223 [==============================] - 1s 4ms/step - loss: 0.5056 -
accuracy: 0.7627
Epoch 83/100
223/223 [==============================] - 1s 3ms/step - loss: 0.5107 -
accuracy: 0.7534
Epoch 84/100
223/223 [==============================] - 1s 4ms/step - loss: 0.5249 -
accuracy: 0.7515
Epoch 85/100
223/223 [==============================] - 1s 3ms/step - loss: 0.5182 -
accuracy: 0.7513
Epoch 86/100
223/223 [==============================] - 1s 3ms/step - loss: 0.5206 -
accuracy: 0.7572
Epoch 87/100
223/223 [==============================] - 1s 3ms/step - loss: 0.5229 -
accuracy: 0.7543
Epoch 88/100
223/223 [==============================] - 1s 3ms/step - loss: 0.5101 -
accuracy: 0.7565
Epoch 89/100
223/223 [==============================] - 1s 4ms/step - loss: 0.5645 -
accuracy: 0.7429
Epoch 90/100
223/223 [==============================] - 1s 3ms/step - loss: 0.5145 -
accuracy: 0.7582
Epoch 91/100
223/223 [==============================] - 1s 3ms/step - loss: 0.5059 -
accuracy: 0.7619
Epoch 92/100
223/223 [==============================] - 1s 3ms/step - loss: 0.5222 -
accuracy: 0.7498
Epoch 93/100
223/223 [==============================] - 1s 3ms/step - loss: 0.5142 -
accuracy: 0.7529
Epoch 94/100
223/223 [==============================] - 1s 3ms/step - loss: 0.5179 -
accuracy: 0.7532
Epoch 95/100
```

```
223/223 [==============================] - 1s 3ms/step - loss: 0.4991 -
accuracy: 0.7581
Epoch 96/100
223/223 [==============================] - 1s 3ms/step - loss: 0.5003 -
accuracy: 0.7616
Epoch 97/100
223/223 [==============================] - 1s 3ms/step - loss: 0.4952 -
accuracy: 0.7653
Epoch 98/100
223/223 [==============================] - 1s 2ms/step - loss: 0.5141 -
accuracy: 0.7543
Epoch 99/100
223/223 [==============================] - 1s 3ms/step - loss: 0.4942 -
accuracy: 0.7593
Epoch 100/100
223/223 [==============================] - 1s 3ms/step - loss: 0.5168 -
accuracy: 0.7578
```

[124]:
```
<keras.callbacks.History at 0x7fe2de089550>
```

[125]:
```python
print(model.summary())
X_test=np.asarray(X_test).astype(np.float)

Y_test=np.asarray(Y_test).astype(np.float)

model.evaluate(X_test, Y_test)
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_15 (Dense)            (None, 64)                1536

 dense_16 (Dense)            (None, 32)                2080

 dense_17 (Dense)            (None, 1)                 33

=================================================================
Total params: 3,649
Trainable params: 3,649
Non-trainable params: 0
_________________________________________________________________
None
```

```
<ipython-input-125-b05a9e24e542>:2: DeprecationWarning: `np.float` is a
deprecated alias for the builtin `float`. To silence this warning, use `float`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.float64` here.
```

```
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  X_test=np.asarray(X_test).astype(np.float)
<ipython-input-125-b05a9e24e542>:4: DeprecationWarning: `np.float` is a
deprecated alias for the builtin `float`. To silence this warning, use `float`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  Y_test=np.asarray(Y_test).astype(np.float)

96/96 [==============================] - 1s 2ms/step - loss: 0.5317 - accuracy:
0.7625
```

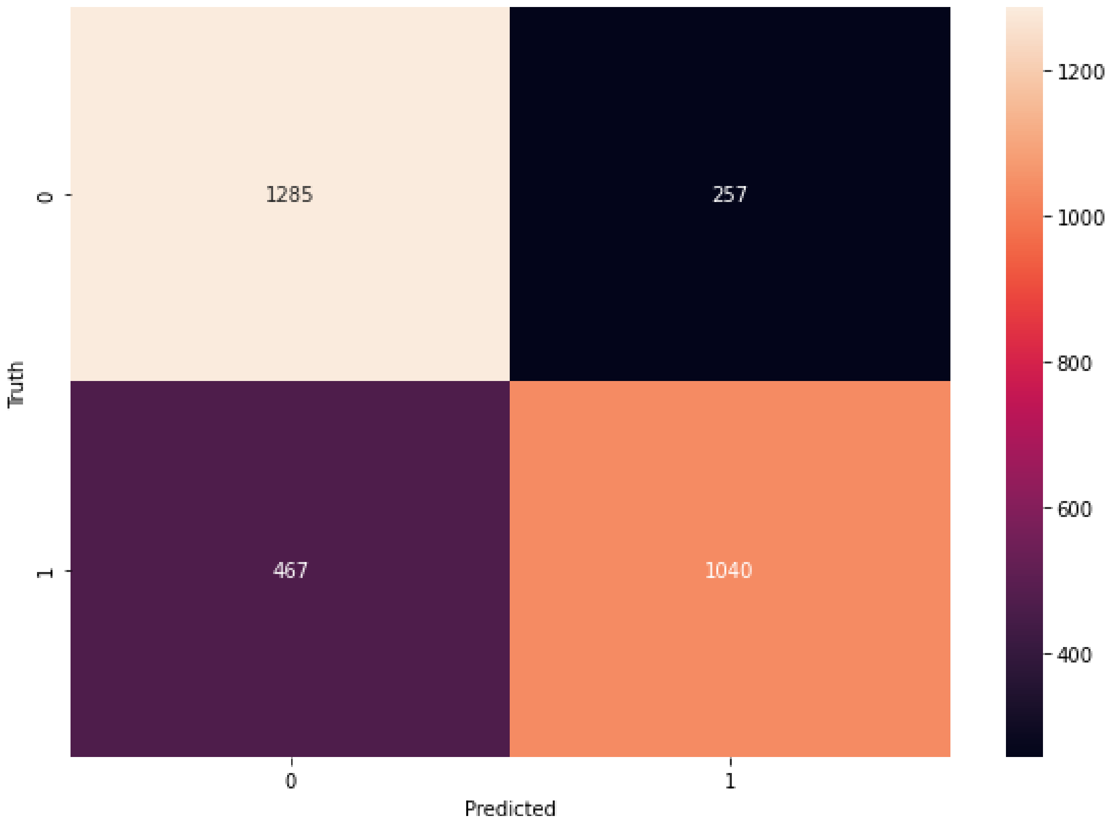
[125]:
```
[0.5317384004592896, 0.762545108795166]
```

[126]:
```python
from sklearn.metrics import confusion_matrix , classification_report
yp = model.predict(X_test)
Y_pred = []
for element in yp:
    if element > 0.5:
        Y_pred.append(1)
    else:
        Y_pred.append(0)
print(classification_report(Y_test,Y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.73      0.83      0.78      1542
         1.0       0.80      0.69      0.74      1507

    accuracy                           0.76      3049
   macro avg       0.77      0.76      0.76      3049
weighted avg       0.77      0.76      0.76      3049
```

[127]:
```python
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')
```

[127]:
```
Text(69.0, 0.5, 'Truth')
```



[ ]:
```
# Summary
# Very simple neural network with hidden layers

# monthly contract, tenure and total charges are the most important predictor␣
↪variables to predict churn.
# NN -Accuracy - 0.72 ,loss=0.75
# F1-Score: 0.76,0.69
```