# INSAID Hiring Exercise

## Important: Kindly go through the instructions mentioned below.

- The Sheet is structured in **4 steps**:

  1. Understanding data and manipulation
  2. Data visualization
  3. Implementing Machine Learning models(Note: It should be more than 1 algorithm)
  4. Model Evaluation and concluding with the best of the model.

- Try to break the codes in the **simplest form** and use number of code block with **proper comments** to them
- We are providing **h** different dataset to choose from(Note: You need to select any one of the dataset from this sample sheet only)
- The **interview calls** will be made solely based on how good you apply the **concepts**.
- Good Luck! Happy Coding!

### Importing the data

In [3]:

```
# use these links to do so:
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.cluster import KMeans
%matplotlib inline
```

```
C:\Users\Jais\new anaconda\lib\site-packages\statsmodels\tools\_testing.py:19: FutureWarning:
pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

### Understanding the data

In [4]:

```
telecom = pd.read_csv("C:/Users/Jais/Downloads/Churn.csv")
```

In [5]:

```
telecom.sample(5)
```

Out[5]:

| | customerID | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines | InternetService | OnlineSecurity | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **6448** | 3685-YLCMQ | Male | 0 | No | No | 58 | Yes | Yes | Fiber optic | No | ... |
| **5037** | 8943-URTMR | Female | 0 | No | No | 2 | Yes | No | Fiber optic | No | ... |
| **3581** | 7860-UXCRM | Male | 0 | Yes | Yes | 63 | Yes | No | DSL | Yes | ... |
| **1110** | 0343-QLUZP | Male | 0 | No | No | 60 | No | No phone service | DSL | Yes | ... |
| **4723** | 4274-OWWYO | Male | 0 | No | No | 1 | Yes | No | Fiber optic | No | ... |

5 rows × 21 columns

In [6]:

```python
telecom.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

In [7]:

```python
telecom.head()
```

Out[7]:

| | customerID | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines | InternetService | OnlineSecurity | ... | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 7590-VHVEG | Female | 0 | Yes | No | 1 | No | No phone service | DSL | No | ... | |
| **1** | 5575-GNVDE | Male | 0 | No | No | 34 | Yes | No | DSL | Yes | ... | |
| **2** | 3668-QPYBK | Male | 0 | No | No | 2 | Yes | No | DSL | Yes | ... | |
| **3** | 7795-CFOCW | Male | 0 | No | No | 45 | No | No phone service | DSL | Yes | ... | |
| **4** | 9237-HQITU | Female | 0 | No | No | 2 | Yes | No | Fiber optic | No | ... | |

5 rows × 21 columns

In [8]:

```python
telecom.tail()
```

Out[8]:

| | customerID | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines | InternetService | OnlineSecurity | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **7038** | 6840-RESVB | Male | 0 | Yes | Yes | 24 | Yes | Yes | DSL | Yes | ... |
| **7039** | 2234-XADUH | Female | 0 | Yes | Yes | 72 | Yes | Yes | Fiber optic | No | ... |
| **7040** | 4801-JZAZL | Female | 0 | Yes | Yes | 11 | No | No phone service | DSL | Yes | ... |
| **7041** | 8361-LTMKD | Male | 1 | Yes | No | 4 | Yes | Yes | Fiber optic | No | ... |

| | customerID | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines | InternetService | OnlineSecurity | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 7042 | 3186-AJIEK | Male | 0 | No | No | 66 | Yes | No | Fiber optic | Yes | ... |

5 rows × 21 columns

In [9]:

```
telecom.describe()
```

Out[9]:

| | 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 |

In [10]:

```
telecom.shape
```

Out[10]:

```
(7043, 21)
```

# Customer Id is not used so drop the column

# Data Manipulation

In [11]:

```
telecom.drop("customerID",axis="columns",inplace=True)
```

In [12]:

```
telecom.dtypes
```

Out[12]:

```
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
```

```
Churn                object
dtype: object
```

In [13]:

```
telecom.TotalCharges.values
```

Out[13]:

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

In [14]:

```
telecom.MonthlyCharges.values
```

Out[14]:

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

In [15]:

```
pd.to_numeric(telecom.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-15-7ebc859efd32> in <module>
----> 1 pd.to_numeric(telecom.TotalCharges)

~\new anaconda\lib\site-packages\pandas\core\tools\numeric.py in to_numeric(arg, errors, 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 [16]:

```
pd.to_numeric(telecom.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]:

```
pd.to_numeric(telecom.TotalCharges,errors="coerce").isnull()
```

Out[17]:

Out[17]:

```
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 [18]:

```
telecom[pd.to_numeric(telecom.TotalCharges,errors="coerce").isnull()]
```

Out[18]:

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

In [19]:

```
telecom.shape
```

Out[19]:

```
(7043, 20)
```

In [20]:

```
telecom.iloc[488]["TotalCharges"]
```

Out[20]:

```
' '
```

In [21]:

```
telecom1 = telecom[telecom.TotalCharges!=' ']
telecom1.shape
```

Out[21]:

```
(7032, 20)
```

In [22]:

```
telecom1.dtypes
```

Out[22]:

```
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 [56]:

```
telecom1.TotalCharges = pd.to_numeric(telecom1.TotalCharges)
```

```
C:\Users\Jais\new anaconda\lib\site-packages\pandas\core\generic.py:5168: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self[name] = value
```

In [57]:

```
telecom1.TotalCharges.dtypes
```

Out[57]:

```
dtype('float64')
```

In [58]:

```
#telecom1[telecom1.Churn=='No']
```

In [59]:

```
#telecom1[telecom1.Churn=='No'].tenure
#telecom1[telecom1.Churn=='Yes'].tenure
```

# Data Visualization

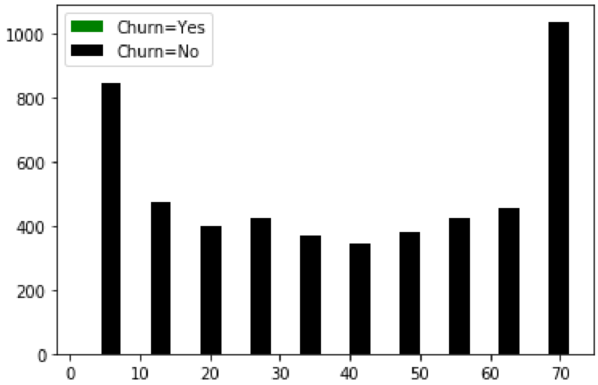
In [63]:

```
tenure_churn_no = telecom1[telecom1.Churn=='No'].tenure
tenure_churn_yes = telecom1[telecom1.Churn=='yes'].tenure

plt.hist([tenure_churn_yes, tenure_churn_no], color=['Green',
'black'],label=['Churn=Yes','Churn=No'])
```

```
plt.legend()
```

Out[63]:

```
<matplotlib.legend.Legend at 0x1da9dd55808>
```




In [ ]:

```

```

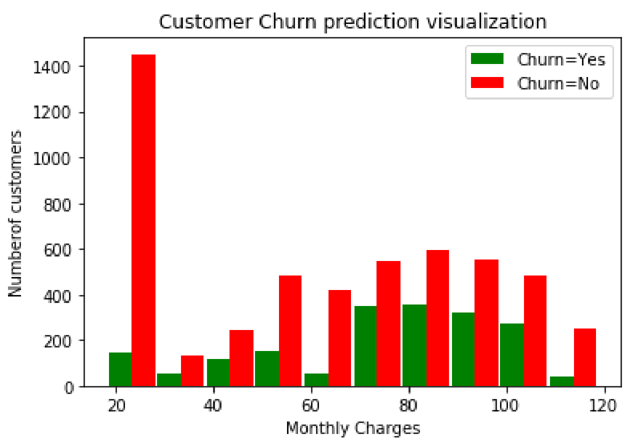
In [64]:

```
mc_churn_no = telecom1[telecom.Churn=='No'].MonthlyCharges
mc_churn_yes = telecom1[telecom1.Churn=='Yes'].MonthlyCharges

plt.xlabel("Monthly Charges")
plt.ylabel("Numberof customers")
plt.title("Customer Churn prediction visualization")

blood_suger_men = [113, 85, 90, 150, 149, 88, 93, 115, 135, 80, 77, 82, 129]
blood_suger_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=Yes', 'Churn
=No'])
plt.legend()
```

```
C:\Users\Jais\new anaconda\lib\site-packages\ipykernel_launcher.py:1: UserWarning: Boolean Series
key will be reindexed to match DataFrame index.
  """Entry point for launching an IPython kernel.
```

Out[64]:

```
<matplotlib.legend.Legend at 0x1daaa8cda48>
```




In [67]:

```
for column in telecom:
    if telecom[column].dtype=='object':
        print(f'{column} : {telecom[column].unique()}')
```

```
        print(f'{column} : {telecom[column].unique()}')
```

```
gender : ['Female' 'Male']
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 [68]:

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

In [69]:

```
print_unique_col_values(telecom1)
```

```
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 [70]:

```
telecom1.replace('No internet service','No',inplace=True)
telecom1.replace('No phone service','No',inplace=True)
```

```
C:\Users\Jais\new anaconda\lib\site-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_guide/indexing.html#returning-a-view-versus-a-copy
  method=method,
```

In [71]:

```
print_unique_col_values(telecom1)
```

```
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 [72]:

```
yes_no_columns =
['Partner','Dependents','PhoneService','MultipleLines','OnlineSecurity','OnlineBackup','DeviceProte
ction','TechSupport','StreamingTV','StreamingMovies','PaperlessBilling','Churn']
for col in yes_no_columns:
    telecom1[col].replace({'Yes': 1, 'No': 0},inplace=True)
```

```
C:\Users\Jais\new anaconda\lib\site-packages\pandas\core\series.py:4581: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  method=method,
```

In [73]:

```
for col in telecom1:
    print(f'{col}: {telecom1[col].unique()}')
```

```
gender: [1 0]
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 [74]:

```
telecom1['gender'].replace({'Female':1, 'Male':0},inplace=True)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-74-024d665ae2cd> in <module>
----> 1 telecom1['gender'].replace({'Female':1, 'Male':0},inplace=True)

~\new anaconda\lib\site-packages\pandas\core\series.py in replace(self, to_replace, value,
inplace, limit, regex, method)
   4579             limit=limit,
   4580             regex=regex,
-> 4581             method=method,
   4582         )
   4583

~\new anaconda\lib\site-packages\pandas\core\generic.py in replace(self, to_replace, value,
inplace, limit, regex, method)
```

```
   6499
   6500             return self.replace(
-> 6501                 to_replace, value, inplace=inplace, limit=limit, regex=regex
   6502             )
   6503         else:

~\new anaconda\lib\site-packages\pandas\core\series.py in replace(self, to_replace, value, inplace, limit, regex, method)
   4579             limit=limit,
   4580             regex=regex,
-> 4581             method=method,
   4582         )
   4583

~\new anaconda\lib\site-packages\pandas\core\generic.py in replace(self, to_replace, value, inplace, limit, regex, method)
   6545                         dest_list=value,
   6546                         inplace=inplace,
-> 6547                         regex=regex,
   6548                     )
   6549

~\new anaconda\lib\site-packages\pandas\core\internals\managers.py in replace_list(self, src_list, dest_list, inplace, regex)
    640         mask = ~isna(values)
    641
--> 642         masks = [comp(s, mask, regex) for s in src_list]
    643
    644         result_blocks = []

~\new anaconda\lib\site-packages\pandas\core\internals\managers.py in <listcomp>(.0)
    640         mask = ~isna(values)
    641
--> 642         masks = [comp(s, mask, regex) for s in src_list]
    643
    644         result_blocks = []

~\new anaconda\lib\site-packages\pandas\core\internals\managers.py in comp(s, mask, regex)
    634
    635             s = com.maybe_box_datetimelike(s)
--> 636             return _compare_or_regex_search(values, s, regex, mask)
    637
    638         # Calculate the mask once, prior to the call of comp

~\new anaconda\lib\site-packages\pandas\core\internals\managers.py in _compare_or_regex_search(a, b, regex, mask)
   1990     if is_datetimelike_v_numeric(a, b) or is_numeric_v_string_like(a, b):
   1991         # GH#29553 avoid deprecation warnings from numpy
-> 1992         _check_comparison_types(False, a, b)
   1993         return False
   1994

~\new anaconda\lib\site-packages\pandas\core\internals\managers.py in _check_comparison_types(result, a, b)
   1970
   1971             raise TypeError(
-> 1972                 f"Cannot compare types {repr(type_names[0])} and {repr(type_names[1])}"
   1973             )
   1974

TypeError: Cannot compare types 'ndarray(dtype=int64)' and 'str'
```

In [75]:

```python
telecom1['gender'].unique()
```

Out[75]:

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

In [76]:

```python
telecom2=pd.get_dummies(data=telecom1,columns=['InternetService','Contract','PaymentMethod'])
telecom2.columns
```

Out[76]:

```
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 [77]:

```
telecom2.sample(4)
```

Out[77]:

| | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines | OnlineSecurity | OnlineBackup | DeviceProtectio |
|---|---|---|---|---|---|---|---|---|---|---|
| **4881** | 0 | 0 | 1 | 0 | 28 | 1 | 0 | 0 | 0 | ( |
| **4788** | 0 | 0 | 0 | 0 | 59 | 1 | 1 | 0 | 0 | ' |
| **4717** | 0 | 0 | 1 | 0 | 38 | 1 | 1 | 1 | 0 | ' |
| **100** | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | ( |

4 rows × 27 columns

In [78]:

```
telecom2.dtypes
```

Out[78]:

```
gender                                       int64
SeniorCitizen                                int64
Partner                                      int64
Dependents                                   int64
tenure                                       int64
PhoneService                                 int64
MultipleLines                                int64
OnlineSecurity                               int64
OnlineBackup                                 int64
DeviceProtection                             int64
TechSupport                                  int64
StreamingTV                                  int64
StreamingMovies                              int64
PaperlessBilling                             int64
MonthlyCharges                             float64
TotalCharges                               float64
Churn                                        int64
InternetService_DSL                          uint8
InternetService_Fiber optic                  uint8
InternetService_No                           uint8
Contract_Month-to-month                      uint8
Contract_One year                            uint8
Contract_Two year                            uint8
PaymentMethod_Bank transfer (automatic)      uint8
PaymentMethod_Credit card (automatic)        uint8
PaymentMethod_Electronic check               uint8
PaymentMethod_Mailed check                   uint8
dtype: object
```

In [79]:

```
cols_to_scale = ['tenure','MonthlyCharges','TotalCharges']
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

telecom2[cols_to_scale] = scaler.fit_transform(telecom2[cols_to_scale])
```

In [80]:

```
telecom2.sample(3)
```

Out[80]:

| | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines | OnlineSecurity | OnlineBackup | DeviceProtect |
|---|---|---|---|---|---|---|---|---|---|---|
| **3138** | 1 | 0 | 0 | 0 | 0.000000 | 1 | 0 | 0 | 0 | |
| **3498** | 0 | 0 | 0 | 0 | 0.591549 | 1 | 0 | 1 | 0 | |
| **4887** | 1 | 0 | 1 | 1 | 0.521127 | 1 | 0 | 0 | 0 | |

3 rows × 27 columns

In [81]:

```
for col in telecom2:
    print(f'{col}: {telecom2[col].unique()}')
```

```
gender: [1 0]
SeniorCitizen: [0 1]
Partner: [1 0]
Dependents: [0 1]
tenure: [0.         0.46478873 0.01408451 0.61971831 0.09859155 0.29577465
 0.12676056 0.38028169 0.85915493 0.16901408 0.21126761 0.8028169
 0.67605634 0.33802817 0.95774648 0.71830986 0.98591549 0.28169014
 0.15492958 0.4084507  0.64788732 1.         0.22535211 0.36619718
 0.05633803 0.63380282 0.14084507 0.97183099 0.87323944 0.5915493
 0.1971831  0.83098592 0.23943662 0.91549296 0.11267606 0.02816901
 0.42253521 0.69014085 0.88732394 0.77464789 0.08450704 0.57746479
 0.47887324 0.66197183 0.3943662  0.90140845 0.52112676 0.94366197
 0.43661972 0.76056338 0.50704225 0.49295775 0.56338028 0.07042254
 0.04225352 0.45070423 0.92957746 0.30985915 0.78873239 0.84507042
 0.18309859 0.26760563 0.73239437 0.54929577 0.81690141 0.32394366
 0.6056338  0.25352113 0.74647887 0.70422535 0.35211268 0.53521127]
PhoneService: [0 1]
MultipleLines: [0 1]
OnlineSecurity: [0 1]
OnlineBackup: [1 0]
DeviceProtection: [0 1]
TechSupport: [0 1]
StreamingTV: [0 1]
StreamingMovies: [0 1]
PaperlessBilling: [1 0]
MonthlyCharges: [0.11542289 0.38507463 0.35422886 ... 0.44626866 0.25820896 0.60149254]
TotalCharges: [0.0012751  0.21586661 0.01031041 ... 0.03780868 0.03321025 0.78764136]
Churn: [0 1]
InternetService_DSL: [1 0]
InternetService_Fiber optic: [0 1]
InternetService_No: [0 1]
Contract_Month-to-month: [1 0]
Contract_One year: [0 1]
Contract_Two year: [0 1]
PaymentMethod_Bank transfer (automatic): [0 1]
PaymentMethod_Credit card (automatic): [0 1]
PaymentMethod_Electronic check: [1 0]
PaymentMethod_Mailed check: [0 1]
```

In [82]:

```
x = telecom2.drop('Churn',axis='columns')
y = telecom2['Churn']
```

In [83]:

```
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 [84]:

```
x_train.shape
```

Out[84]:

```
(5625, 26)
```

In [85]:

```
x_test.shape
```

Out[85]:

```
(1407, 26)
```

In [86]:

```
x_train[:10]
```

Out[86]:

| | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines | OnlineSecurity | OnlineBackup | DeviceProtect |
|---|---|---|---|---|---|---|---|---|---|---|
| **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 | |

10 rows × 26 columns

In [87]:

```
len(x_train.columns)
```

Out[87]:

```
26
```

In [ ]:

```

```

In [53]:

```
### Conclusion: What all did you understand from the above charts
```

### Implement Machine Learning Models

In [89]:

```
import tensorflow as tf
from tensorflow import keras


model = keras.Sequential([
```

```
model = keras.Sequential([
    keras.layers.Dense(20, input_shape=(26,), 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)
```

```
Train on 5625 samples
Epoch 1/100
5625/5625 [==============================] - 2s 307us/sample - loss: 0.5356 - accuracy: 0.7228
Epoch 2/100
5625/5625 [==============================] - 1s 200us/sample - loss: 0.4363 - accuracy: 0.7902
Epoch 3/100
5625/5625 [==============================] - 1s 128us/sample - loss: 0.4226 - accuracy: 0.8004
Epoch 4/100
5625/5625 [==============================] - 1s 147us/sample - loss: 0.4178 - accuracy: 0.8041
Epoch 5/100
5625/5625 [==============================] - 1s 119us/sample - loss: 0.4150 - accuracy: 0.8052
Epoch 6/100
5625/5625 [==============================] - 1s 167us/sample - loss: 0.4128 - accuracy: 0.8050
Epoch 7/100
5625/5625 [==============================] - 1s 148us/sample - loss: 0.4119 - accuracy: 0.8055
Epoch 8/100
5625/5625 [==============================] - 1s 161us/sample - loss: 0.4103 - accuracy: 0.8078
Epoch 9/100
5625/5625 [==============================] - 1s 187us/sample - loss: 0.4105 - accuracy: 0.8085
Epoch 10/100
5625/5625 [==============================] - 1s 186us/sample - loss: 0.4085 - accuracy: 0.8085
Epoch 11/100
5625/5625 [==============================] - 1s 101us/sample - loss: 0.4078 - accuracy: 0.8092
Epoch 12/100
5625/5625 [==============================] - 1s 125us/sample - loss: 0.4071 - accuracy: 0.8089
Epoch 13/100
5625/5625 [==============================] - 1s 147us/sample - loss: 0.4067 - accuracy: 0.8071
Epoch 14/100
5625/5625 [==============================] - 1s 146us/sample - loss: 0.4052 - accuracy: 0.8071
Epoch 15/100
5625/5625 [==============================] - 1s 139us/sample - loss: 0.4045 - accuracy: 0.8108
Epoch 16/100
5625/5625 [==============================] - 1s 147us/sample - loss: 0.4033 - accuracy: 0.8121
Epoch 17/100
5625/5625 [==============================] - 1s 144us/sample - loss: 0.4026 - accuracy: 0.8116
Epoch 18/100
5625/5625 [==============================] - 1s 101us/sample - loss: 0.4009 - accuracy: 0.8140
Epoch 19/100
5625/5625 [==============================] - 1s 146us/sample - loss: 0.4010 - accuracy: 0.8132
Epoch 20/100
5625/5625 [==============================] - 1s 142us/sample - loss: 0.4007 - accuracy: 0.8144
Epoch 21/100
5625/5625 [==============================] - 1s 139us/sample - loss: 0.4001 - accuracy: 0.8142
Epoch 22/100
5625/5625 [==============================] - 1s 109us/sample - loss: 0.3997 - accuracy: 0.8151
Epoch 23/100
5625/5625 [==============================] - 1s 140us/sample - loss: 0.3984 - accuracy: 0.8165
Epoch 24/100
5625/5625 [==============================] - 1s 140us/sample - loss: 0.3977 - accuracy: 0.8174
Epoch 25/100
5625/5625 [==============================] - 1s 139us/sample - loss: 0.3971 - accuracy: 0.8153
Epoch 26/100
5625/5625 [==============================] - 1s 137us/sample - loss: 0.3970 - accuracy: 0.8149
Epoch 27/100
5625/5625 [==============================] - 1s 134us/sample - loss: 0.3957 - accuracy: 0.8144
Epoch 28/100
5625/5625 [==============================] - 1s 156us/sample - loss: 0.3966 - accuracy: 0.8165
Epoch 29/100
5625/5625 [==============================] - 1s 155us/sample - loss: 0.3949 - accuracy: 0.8190
Epoch 30/100
5625/5625 [==============================] - 1s 133us/sample - loss: 0.3948 - accuracy: 0.8174
Epoch 31/100
5625/5625 [==============================] - 1s 114us/sample - loss: 0.3942 - accuracy: 0.8171
Epoch 32/100
5625/5625 [==============================] - 1s 110us/sample - loss: 0.3935 - accuracy: 0.8176
Epoch 33/100
5625/5625 [==============================] - 1s 107us/sample - loss: 0.3939 - accuracy: 0.8187
```

```
Epoch 34/100
5625/5625 [==============================] - 1s 111us/sample - loss: 0.3926 - accuracy: 0.8176
Epoch 35/100
5625/5625 [==============================] - 1s 124us/sample - loss: 0.3923 - accuracy: 0.8190
Epoch 36/100
5625/5625 [==============================] - 1s 131us/sample - loss: 0.3917 - accuracy: 0.8188
Epoch 37/100
5625/5625 [==============================] - 1s 129us/sample - loss: 0.3917 - accuracy: 0.8199
Epoch 38/100
5625/5625 [==============================] - 1s 132us/sample - loss: 0.3909 - accuracy: 0.8201
Epoch 39/100
5625/5625 [==============================] - 1s 125us/sample - loss: 0.3905 - accuracy: 0.8181
Epoch 40/100
5625/5625 [==============================] - 1s 127us/sample - loss: 0.3903 - accuracy: 0.8199
Epoch 41/100
5625/5625 [==============================] - 1s 137us/sample - loss: 0.3903 - accuracy: 0.8188
Epoch 42/100
5625/5625 [==============================] - 1s 133us/sample - loss: 0.3895 - accuracy: 0.8199
Epoch 43/100
5625/5625 [==============================] - 1s 132us/sample - loss: 0.3890 - accuracy: 0.8180
Epoch 44/100
5625/5625 [==============================] - 1s 126us/sample - loss: 0.3892 - accuracy: 0.8206
Epoch 45/100
5625/5625 [==============================] - 1s 134us/sample - loss: 0.3886 - accuracy: 0.8181
Epoch 46/100
5625/5625 [==============================] - 1s 125us/sample - loss: 0.3881 - accuracy: 0.8199 - l
oss: 0.3
Epoch 47/100
5625/5625 [==============================] - 1s 108us/sample - loss: 0.3879 - accuracy: 0.8197
Epoch 48/100
5625/5625 [==============================] - 1s 108us/sample - loss: 0.3878 - accuracy: 0.8220
Epoch 49/100
5625/5625 [==============================] - 1s 104us/sample - loss: 0.3869 - accuracy: 0.8210
Epoch 50/100
5625/5625 [==============================] - 1s 112us/sample - loss: 0.3877 - accuracy: 0.8217
Epoch 51/100
5625/5625 [==============================] - 1s 104us/sample - loss: 0.3871 - accuracy: 0.8213
Epoch 52/100
5625/5625 [==============================] - 1s 106us/sample - loss: 0.3862 - accuracy: 0.8217
Epoch 53/100
5625/5625 [==============================] - 1s 121us/sample - loss: 0.3865 - accuracy: 0.8220
Epoch 54/100
5625/5625 [==============================] - 1s 141us/sample - loss: 0.3860 - accuracy: 0.8194
Epoch 55/100
5625/5625 [==============================] - 1s 153us/sample - loss: 0.3854 - accuracy: 0.8204
Epoch 56/100
5625/5625 [==============================] - 1s 111us/sample - loss: 0.3862 - accuracy: 0.8208
Epoch 57/100
5625/5625 [==============================] - 1s 139us/sample - loss: 0.3852 - accuracy: 0.8203
Epoch 58/100
5625/5625 [==============================] - 1s 147us/sample - loss: 0.3849 - accuracy: 0.8213
Epoch 59/100
5625/5625 [==============================] - 1s 148us/sample - loss: 0.3852 - accuracy: 0.8212
Epoch 60/100
5625/5625 [==============================] - 1s 142us/sample - loss: 0.3847 - accuracy: 0.8213
Epoch 61/100
5625/5625 [==============================] - 1s 148us/sample - loss: 0.3838 - accuracy: 0.8204 - l
os
Epoch 62/100
5625/5625 [==============================] - 1s 149us/sample - loss: 0.3842 - accuracy: 0.8190
Epoch 63/100
5625/5625 [==============================] - 1s 142us/sample - loss: 0.3832 - accuracy: 0.8210
Epoch 64/100
5625/5625 [==============================] - 1s 135us/sample - loss: 0.3840 - accuracy: 0.8212
Epoch 65/100
5625/5625 [==============================] - 1s 142us/sample - loss: 0.3833 - accuracy: 0.8201
Epoch 66/100
5625/5625 [==============================] - 1s 139us/sample - loss: 0.3833 - accuracy: 0.8219
Epoch 67/100
5625/5625 [==============================] - 1s 145us/sample - loss: 0.3824 - accuracy: 0.8215
Epoch 68/100
5625/5625 [==============================] - 1s 148us/sample - loss: 0.3833 - accuracy: 0.8220
Epoch 69/100
5625/5625 [==============================] - 1s 109us/sample - loss: 0.3838 - accuracy: 0.8220
Epoch 70/100
5625/5625 [==============================] - 1s 119us/sample - loss: 0.3830 - accuracy: 0.8220
Epoch 71/100
```

```
5625/5625 [==============================] - 1s 132us/sample - loss: 0.3822 - accuracy: 0.8228
Epoch 72/100
5625/5625 [==============================] - 1s 132us/sample - loss: 0.3819 - accuracy: 0.8219
Epoch 73/100
5625/5625 [==============================] - 1s 106us/sample - loss: 0.3825 - accuracy: 0.8236
Epoch 74/100
5625/5625 [==============================] - 1s 106us/sample - loss: 0.3819 - accuracy: 0.8212
Epoch 75/100
5625/5625 [==============================] - 1s 105us/sample - loss: 0.3817 - accuracy: 0.8217
Epoch 76/100
5625/5625 [==============================] - 1s 133us/sample - loss: 0.3816 - accuracy: 0.8235
Epoch 77/100
5625/5625 [==============================] - 1s 177us/sample - loss: 0.3810 - accuracy: 0.8222
Epoch 78/100
5625/5625 [==============================] - 1s 134us/sample - loss: 0.3815 - accuracy: 0.8235
Epoch 79/100
5625/5625 [==============================] - 1s 153us/sample - loss: 0.3820 - accuracy: 0.8224
Epoch 80/100
5625/5625 [==============================] - 1s 104us/sample - loss: 0.3809 - accuracy: 0.8213
Epoch 81/100
5625/5625 [==============================] - 1s 153us/sample - loss: 0.3809 - accuracy: 0.8222
Epoch 82/100
5625/5625 [==============================] - 1s 144us/sample - loss: 0.3804 - accuracy: 0.8231
Epoch 83/100
5625/5625 [==============================] - 1s 138us/sample - loss: 0.3808 - accuracy: 0.8210
Epoch 84/100
5625/5625 [==============================] - 1s 142us/sample - loss: 0.3811 - accuracy: 0.8219
Epoch 85/100
5625/5625 [==============================] - 1s 125us/sample - loss: 0.3801 - accuracy: 0.8215
Epoch 86/100
5625/5625 [==============================] - 1s 130us/sample - loss: 0.3803 - accuracy: 0.8228
Epoch 87/100
5625/5625 [==============================] - 1s 111us/sample - loss: 0.3804 - accuracy: 0.8220
Epoch 88/100
5625/5625 [==============================] - 1s 142us/sample - loss: 0.3798 - accuracy: 0.8219
Epoch 89/100
5625/5625 [==============================] - 1s 139us/sample - loss: 0.3798 - accuracy: 0.8226
Epoch 90/100
5625/5625 [==============================] - 1s 142us/sample - loss: 0.3791 - accuracy: 0.8242
Epoch 91/100
5625/5625 [==============================] - 1s 183us/sample - loss: 0.3789 - accuracy: 0.8228
Epoch 92/100
5625/5625 [==============================] - 1s 186us/sample - loss: 0.3797 - accuracy: 0.8229
Epoch 93/100
5625/5625 [==============================] - 1s 140us/sample - loss: 0.3790 - accuracy: 0.8217
Epoch 94/100
5625/5625 [==============================] - 1s 109us/sample - loss: 0.3779 - accuracy: 0.8249
Epoch 95/100
5625/5625 [==============================] - 1s 128us/sample - loss: 0.3784 - accuracy: 0.8226
Epoch 96/100
5625/5625 [==============================] - 1s 109us/sample - loss: 0.3787 - accuracy: 0.8226
Epoch 97/100
5625/5625 [==============================] - 1s 111us/sample - loss: 0.3785 - accuracy: 0.8240
Epoch 98/100
5625/5625 [==============================] - 1s 123us/sample - loss: 0.3782 - accuracy: 0.8231
Epoch 99/100
5625/5625 [==============================] - 1s 140us/sample - loss: 0.3785 - accuracy: 0.8204
Epoch 100/100
5625/5625 [==============================] - 1s 146us/sample - loss: 0.3782 - accuracy: 0.8235
```

Out[89]:

```
<tensorflow.python.keras.callbacks.History at 0x1daabdb53c8>
```

In [91]:

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

Out[91]:

```
array([[0.15390295],
       [0.31169254],
       [0.00720591],
       [0.7977661 ],
```

[0.46280918]], dtype=float32)

## Model Evaluation

In [92]:

```
y_test[:10]
```

Out[92]:

```
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 [95]:

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

In [96]:

```
y_pred[:10]
```

Out[96]:

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

In [98]:

```
from sklearn.metrics import confusion_matrix , classification_report
print(classification_report(y_test,y_pred))
```

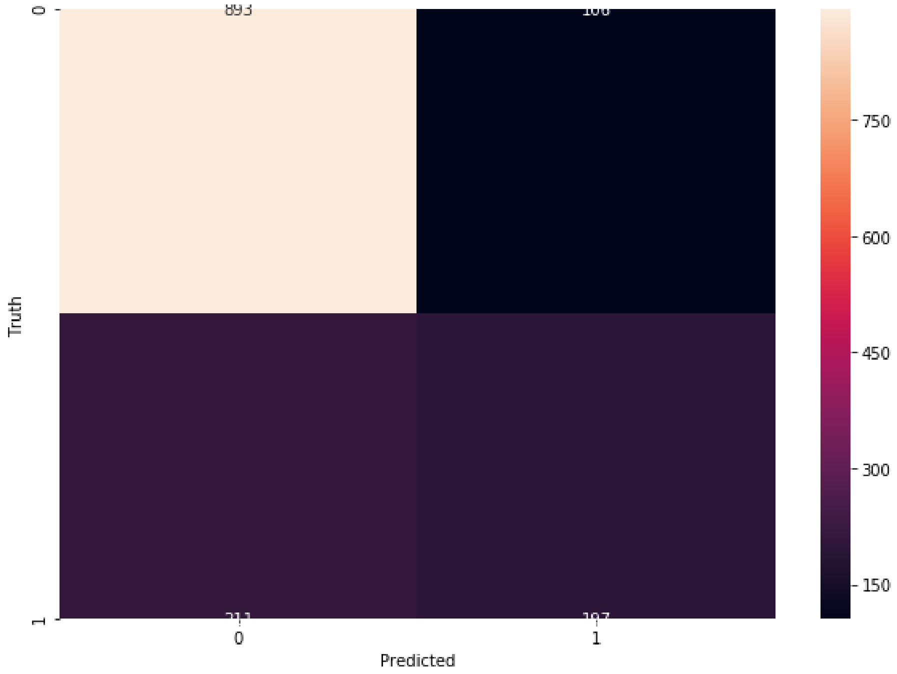
```
              precision    recall  f1-score   support

           0       0.81      0.89      0.85       999
           1       0.65      0.48      0.55       408

    accuracy                           0.77      1407
   macro avg       0.73      0.69      0.70      1407
weighted avg       0.76      0.77      0.76      1407
```

In [101]:

```
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[101]:

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




In [102]:

```
y_test.shape
```

Out[102]:

```
(1407,)
```

In [108]:

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

Out[108]:

```
0.78
```

precision for 0 class i.e. precision for customer who did not churn

In [109]:

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

Out[109]:

```
0.83
```

precision for 1 class i.e. precision for customer who actullay churn

In [110]:

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

Out[110]:

```
0.63
```

In [111]:

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

Out[111]:

```
0.86
```

### Final Conclusions

In [113]:

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

Out[113]:

```
0.56
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: