2. Data Understanding

2.1. Importing necessary Libraries

In [1]:
```
import numpy as np
import pandas as pd
# Max Number of columns to display
pd.set_option("display.max_columns",None)
# seaborn and matplotlib for visualization
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
# warnings to filter future warnings
import warnings
warnings.filterwarnings("ignore")
```

2.2 Loading Data

In [2]:
```
df = pd.read_csv("telecom_customer_churn.csv",index_col=0)
```

2.3 General Data Overview

2.3.1 Displaying First five rows(Head)

In [3]:
```
head = df.head()
head
```

Out[3]:

Customer ID	Gender	Age	Married	Number of Dependents	City	Zip Code	Latitude	Longitude	Number of Referrals
0002-ORFBO	Female	37	Yes	0	Frazier Park	93225	34.827662	-118.999073	2
0003-MKNFE	Male	46	No	0	Glendale	91206	34.162515	-118.203869	0
0004-TLHLJ	Male	50	No	0	Costa Mesa	92627	33.645672	-117.922613	0
0011-IGKFF	Male	78	Yes	0	Martinez	94553	38.014457	-122.115432	1
0013-EXCHZ	Female	75	Yes	0	Camarillo	93010	34.227846	-119.079903	3

2.3.2 Displaying last five rows (Tail)

In [4]:
```
tail = df.tail()
tail
```

Out[4]:

ıt d	Monthly Charge	Total Charges	Total Refunds	Total Extra Data Charges	Total Long Distance Charges	Total Revenue	Customer Status	Churn Category	Churn Reason
it d	55.15	742.90	0.0	0	606.84	1349.74	Stayed	NaN	NaN
k al	85.10	1873.70	0.0	0	356.40	2230.10	Churned	Dissatisfaction	Product dissatisfaction
it d	50.30	92.75	0.0	0	37.24	129.99	Joined	NaN	NaN
it d	67.85	4627.65	0.0	0	142.04	4769.69	Stayed	NaN	NaN
k al	59.00	3707.60	0.0	0	0.00	3707.60	Stayed	NaN	NaN

From the head and tail the data has a consistent number of columns(38)

2.3.3 Columns Preview

In [5]:
```
#Columns in the dataset
df.columns
```

Out[5]:
```
Index(['Gender', 'Age', 'Married', 'Number of Dependents', 'City', 'Zip Code',
       'Latitude', 'Longitude', 'Number of Referrals', 'Tenure in Months',
       'Offer', 'Phone Service', 'Avg Monthly Long Distance Charges',
       'Multiple Lines', 'Internet Service', 'Internet Type',
       'Avg Monthly GB Download', 'Online Security', 'Online Backup',
       'Device Protection Plan', 'Premium Tech Support', 'Streaming TV',
       'Streaming Movies', 'Streaming Music', 'Unlimited Data', 'Contract',
       'Paperless Billing', 'Payment Method', 'Monthly Charge',
       'Total Charges', 'Total Refunds', 'Total Extra Data Charges',
       'Total Long Distance Charges', 'Total Revenue', 'Customer Status',
       'Churn Category', 'Churn Reason'],
      dtype='object')
```

In [6]:
```
# Column Descriptions
data_desc = pd.read_csv("telecom_data_dictionary.csv",encoding = "latin1",index_c
data_desc = data_desc.drop(['Table'],axis=1)
data_desc
```

Out[6]:

Field	Description
CustomerID	A unique ID that identifies each customer
Gender	The customer s gender: Male, Female
Age	The customer s current age, in years, at the t...
Married	Indicates if the customer is married: Yes, No
Number of Dependents	Indicates the number of dependents that live w...
City	The city of the customer s primary residence i...
Zip Code	The zip code of the customer s primary residence
Latitude	The latitude of the customer s primary residence
Longitude	The longitude of the customer s primary residence
Number of Referrals	Indicates the number of times the customer has...
Tenure in Months	Indicates the total amount of months that the ...
Offer	Identifies the last marketing offer that the c...
Phone Service	Indicates if the customer subscribes to home p...
Avg Monthly Long Distance Charges	Indicates the customer s average long distance...
Multiple Lines	Indicates if the customer subscribes to multip...
Internet Service	Indicates if the customer subscribes to Intern...
Internet Type	Indicates the customer's type of internet conn...
Avg Monthly GB Download	Indicates the customer s average download volu...
Online Security	Indicates if the customer subscribes to an add...
Online Backup	Indicates if the customer subscribes to an add...
Device Protection Plan	Indicates if the customer subscribes to an add...
Premium Tech Support	Indicates if the customer subscribes to an add...
Streaming TV	Indicates if the customer uses their Internet ...
Streaming Movies	Indicates if the customer uses their Internet ...
Streaming Music	Indicates if the customer uses their Internet ...
Unlimited Data	Indicates if the customer has paid an addition...
Contract	Indicates the customer s current contract type...
Paperless Billing	Indicates if the customer has chosen paperless...
Payment Method	Indicates how the customer pays their bill: Ba...
Monthly Charge	Indicates the customer s current total monthly...
Total Charges	Indicates the customer s total charges, calcul...

Field	Description
Total Refunds	Indicates the customer s total refunds, calcul...
Total Extra Data Charges	Indicates the customer s total charges for ext...
Total Long Distance Charges	Indicates the customer s total charges for lon...
Total Revenue	Indicates the company's total revenue from thi...
Customer Status	Indicates the status of the customer at the en...
Churn Category	A high-level category for the customer s reaso...
Churn Reason	A customer s specific reason for leaving the c...
Zip Code	The zip code of the customer s primary residence
Population	A current population estimate for the entire Z...

In [7]:
```
# Shape of the data
shape = df.shape
```

The dataset contains 7043 rows and 38 columns.

In [8]:
```
# Check the datatypes
info = df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7043 entries, 0002-ORFBO to 9995-HOTOH
Data columns (total 37 columns):
 #   Column                             Non-Null Count  Dtype  
---  ------                             --------------  -----  
 0   Gender                             7043 non-null   object 
 1   Age                                7043 non-null   int64  
 2   Married                            7043 non-null   object 
 3   Number of Dependents               7043 non-null   int64  
 4   City                               7043 non-null   object 
 5   Zip Code                           7043 non-null   int64  
 6   Latitude                           7043 non-null   float64
 7   Longitude                          7043 non-null   float64
 8   Number of Referrals                7043 non-null   int64  
 9   Tenure in Months                   7043 non-null   int64  
 10  Offer                              7043 non-null   object 
 11  Phone Service                      7043 non-null   object 
 12  Avg Monthly Long Distance Charges  6361 non-null   float64
 13  Multiple Lines                     6361 non-null   object 
 14  Internet Service                   7043 non-null   object 
 15  Internet Type                      5517 non-null   object 
 16  Avg Monthly GB Download            5517 non-null   float64
 17  Online Security                    5517 non-null   object 
 18  Online Backup                      5517 non-null   object 
 19  Device Protection Plan             5517 non-null   object 
 20  Premium Tech Support               5517 non-null   object 
 21  Streaming TV                       5517 non-null   object 
 22  Streaming Movies                   5517 non-null   object 
 23  Streaming Music                    5517 non-null   object 
 24  Unlimited Data                     5517 non-null   object 
 25  Contract                           7043 non-null   object 
 26  Paperless Billing                  7043 non-null   object 
 27  Payment Method                     7043 non-null   object 
 28  Monthly Charge                     7043 non-null   float64
 29  Total Charges                      7043 non-null   float64
 30  Total Refunds                      7043 non-null   float64
 31  Total Extra Data Charges           7043 non-null   int64  
 32  Total Long Distance Charges        7043 non-null   float64
 33  Total Revenue                      7043 non-null   float64
 34  Customer Status                    7043 non-null   object 
 35  Churn Category                     1869 non-null   object 
 36  Churn Reason                       1869 non-null   object 
dtypes: float64(9), int64(6), object(22)
memory usage: 2.0+ MB
```

The dataset contains 15 numerical columns 9 of which are floats and 6 are integers. It also has 23 categorical columns.

In [9]:
```
df.describe().T
```

Out[9]:

	count	mean	std	min	25%	50%	
Age	7043.0	46.509726	16.750352	19.000000	32.000000	46.000000	60.0
Number of Dependents	7043.0	0.468692	0.962802	0.000000	0.000000	0.000000	0.0
Zip Code	7043.0	93486.070567	1856.767505	90001.000000	92101.000000	93518.000000	95329.0
Latitude	7043.0	36.197455	2.468929	32.555828	33.990646	36.205465	38.1
Longitude	7043.0	-119.756684	2.154425	-124.301372	-121.788090	-119.595293	-117.9
Number of Referrals	7043.0	1.951867	3.001199	0.000000	0.000000	0.000000	3.0
Tenure in Months	7043.0	32.386767	24.542061	1.000000	9.000000	29.000000	55.0
Avg Monthly Long Distance Charges	6361.0	25.420517	14.200374	1.010000	13.050000	25.690000	37.6
Avg Monthly GB Download	5517.0	26.189958	19.586585	2.000000	13.000000	21.000000	30.0
Monthly Charge	7043.0	63.596131	31.204743	-10.000000	30.400000	70.050000	89.7
Total Charges	7043.0	2280.381264	2266.220462	18.800000	400.150000	1394.550000	3786.6
Total Refunds	7043.0	1.962182	7.902614	0.000000	0.000000	0.000000	0.0
Total Extra Data Charges	7043.0	6.860713	25.104978	0.000000	0.000000	0.000000	0.0
Total Long Distance Charges	7043.0	749.099262	846.660055	0.000000	70.545000	401.440000	1191.1
Total Revenue	7043.0	3034.379056	2865.204542	21.360000	605.610000	2108.640000	4801.1

3 Data Cleaning and Preparation

In [10]:
```
# Removing column spaces
df.columns = df.columns.str.replace(' ', '')
```

3.1 Working with Numerical Variables

In [11]:
```
# Separating numerical from categorical data
df_num = df.select_dtypes(include=["int64", "float64"])
df_num.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7043 entries, 0002-ORFBO to 9995-HOTOH
Data columns (total 15 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   Age                             7043 non-null   int64  
 1   NumberofDependents              7043 non-null   int64  
 2   ZipCode                         7043 non-null   int64  
 3   Latitude                        7043 non-null   float64
 4   Longitude                       7043 non-null   float64
 5   NumberofReferrals               7043 non-null   int64  
 6   TenureinMonths                  7043 non-null   int64  
 7   AvgMonthlyLongDistanceCharges   6361 non-null   float64
 8   AvgMonthlyGBDownload            5517 non-null   float64
 9   MonthlyCharge                   7043 non-null   float64
 10  TotalCharges                    7043 non-null   float64
 11  TotalRefunds                    7043 non-null   float64
 12  TotalExtraDataCharges           7043 non-null   int64  
 13  TotalLongDistanceCharges        7043 non-null   float64
 14  TotalRevenue                    7043 non-null   float64
dtypes: float64(9), int64(6)
memory usage: 880.4+ KB
```

In [12]:
```
# Checking for Missing values
Missing_num = df_num.isna().sum().sort_values(ascending = False)
Missing_num
```

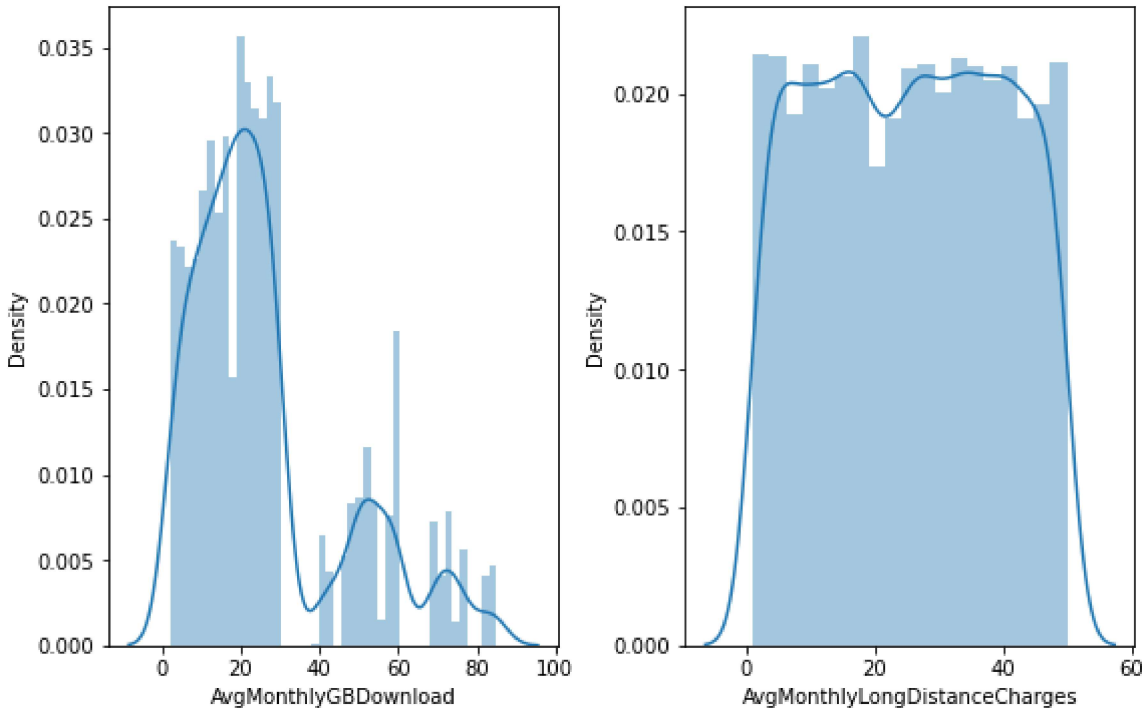
Out[12]:
```
AvgMonthlyGBDownload             1526
AvgMonthlyLongDistanceCharges     682
Age                                 0
NumberofDependents                  0
ZipCode                             0
Latitude                            0
Longitude                           0
NumberofReferrals                   0
TenureinMonths                      0
MonthlyCharge                       0
TotalCharges                        0
TotalRefunds                        0
TotalExtraDataCharges               0
TotalLongDistanceCharges            0
TotalRevenue                        0
dtype: int64
```

In [13]:
```
# Percentages of missing values
per_missing_vals = Missing_num*100/len(df)
per_missing_vals.sort_values(ascending=False)
```

Out[13]:
```
AvgMonthlyGBDownload              21.666903
AvgMonthlyLongDistanceCharges      9.683374
Age                                0.000000
NumberofDependents                 0.000000
ZipCode                            0.000000
Latitude                           0.000000
Longitude                          0.000000
NumberofReferrals                  0.000000
TenureinMonths                     0.000000
MonthlyCharge                      0.000000
TotalCharges                       0.000000
TotalRefunds                       0.000000
TotalExtraDataCharges              0.000000
TotalLongDistanceCharges           0.000000
TotalRevenue                       0.000000
dtype: float64
```

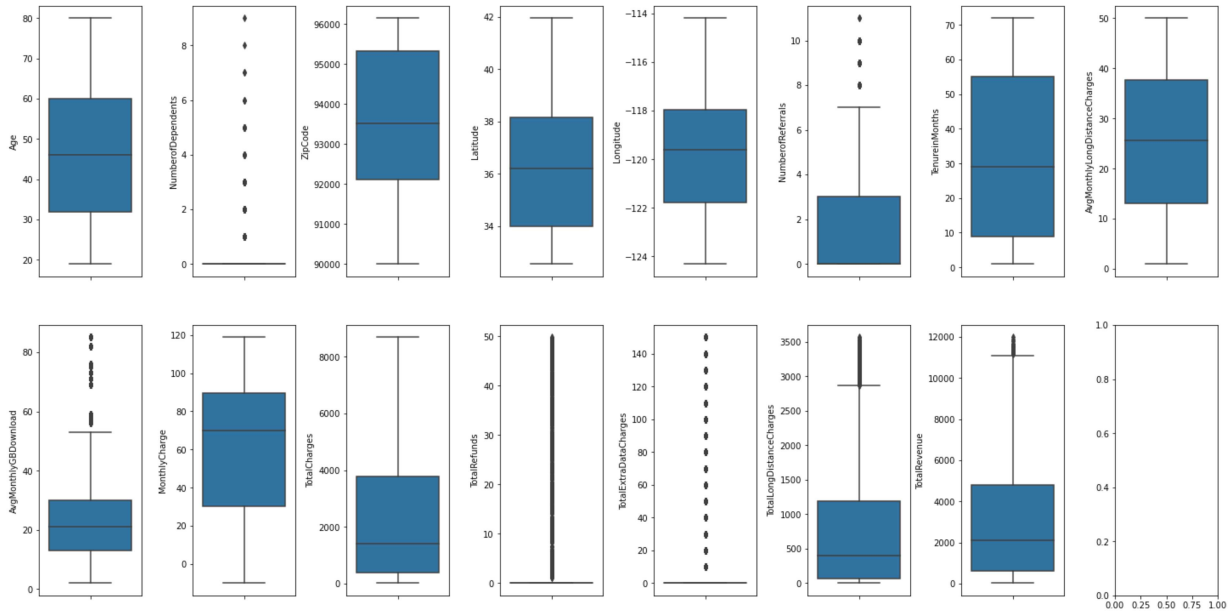
From the numerical features,only two(`Avg Monthly GB Download` , `Avg Monthly Long Distance Charges`) had missing values of ~21% and ~9% respectively. We check the distribution to determine what measure of central tendency to use in filling in the missing values.

In [14]:
```
# Creating a function to draw a distplot
num_miss = ['AvgMonthlyGBDownload','AvgMonthlyLongDistanceCharges']
fig,ax = plt.subplots(ncols=2,nrows = 1,figsize = (8,5))
index = 0
ax = ax.flatten()
for col,value in df[num_miss].items():
    sns.distplot(value,ax = ax[index])
    index += 1
plt.tight_layout(pad= 0.5,w_pad= 0.5,h_pad=3.0);
```



The `AveragemonthlyGBDownload` is highly skewed to the right prompting use of the median to fill in the missing values.For the `Averagemonthlylongdistancechargesdata` , it seems to somewhat take the normal distribution out of the close ranges of the data. For this reason, either the mean, median or mode works. For clarity we will check for outliers to best determine the best measure of central tendency for the `Averagemonthlylongdistancechargesdata` .

In [15]:
```
# Check for outliers in the dataset
fig,ax = plt.subplots(ncols=8,nrows =2,figsize = (20,10))
index = 0
ax = ax.flatten()
for col,value in df_num.items():
    sns.boxplot(y = col,data = df_num,ax = ax[index])
    index += 1
plt.tight_layout(pad= 0.5,w_pad= 0.7,h_pad=5.0);
```



Very few features have outliers. The `Average monthly long distance charges data` has no outliers hence the best measure to fill in the missing values would be the mean.

In [16]:
```
g in missing values
AvgMonthlyLongDistanceCharges'].fillna(df_num['AvgMonthlyLongDistanceCharges'].me
AvgMonthlyGBDownload'].fillna(df_num['AvgMonthlyGBDownload'].median(),inplace= Tr
```

In [17]:
```
df_num.isna().sum()
```

Out[17]:
```
Age                                0
NumberofDependents                 0
ZipCode                            0
Latitude                           0
Longitude                          0
NumberofReferrals                  0
TenureinMonths                     0
AvgMonthlyLongDistanceCharges      0
AvgMonthlyGBDownload               0
MonthlyCharge                      0
TotalCharges                       0
TotalRefunds                       0
TotalExtraDataCharges              0
TotalLongDistanceCharges           0
TotalRevenue                       0
dtype: int64
```

3.2 Working with Categorical data

In [18]:
```
df_cat = df.select_dtypes(exclude=["int64", "float64"])
df_cat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7043 entries, 0002-ORFBO to 9995-HOTOH
Data columns (total 22 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   Gender                7043 non-null   object
 1   Married               7043 non-null   object
 2   City                  7043 non-null   object
 3   Offer                 7043 non-null   object
 4   PhoneService          7043 non-null   object
 5   MultipleLines         6361 non-null   object
 6   InternetService       7043 non-null   object
 7   InternetType          5517 non-null   object
 8   OnlineSecurity        5517 non-null   object
 9   OnlineBackup          5517 non-null   object
 10  DeviceProtectionPlan  5517 non-null   object
 11  PremiumTechSupport    5517 non-null   object
 12  StreamingTV           5517 non-null   object
 13  StreamingMovies       5517 non-null   object
 14  StreamingMusic        5517 non-null   object
 15  UnlimitedData         5517 non-null   object
 16  Contract              7043 non-null   object
 17  PaperlessBilling      7043 non-null   object
 18  PaymentMethod         7043 non-null   object
 19  CustomerStatus        7043 non-null   object
 20  ChurnCategory         1869 non-null   object
 21  ChurnReason           1869 non-null   object
dtypes: object(22)
memory usage: 1.2+ MB
```

In [19]:
```
# Lets check for missing values
Missing_cat = df_cat.isna().sum().sort_values(ascending = False)
Missing_cat
```

Out[19]:
```
ChurnReason             5174
ChurnCategory           5174
OnlineBackup            1526
UnlimitedData           1526
StreamingMusic          1526
StreamingMovies         1526
StreamingTV             1526
DeviceProtectionPlan    1526
PremiumTechSupport      1526
OnlineSecurity          1526
InternetType            1526
MultipleLines            682
Married                    0
InternetService            0
PhoneService               0
Offer                      0
Contract                   0
PaperlessBilling           0
PaymentMethod              0
CustomerStatus             0
City                       0
Gender                     0
dtype: int64
```

In [20]:
```
# Missing Value percentages
per_missing_cat = Missing_cat*100/len(df)
per_missing_cat.sort_values(ascending=False)
```

Out[20]:
```
ChurnReason             73.463013
ChurnCategory           73.463013
OnlineBackup            21.666903
UnlimitedData           21.666903
StreamingMusic          21.666903
StreamingMovies         21.666903
StreamingTV             21.666903
DeviceProtectionPlan    21.666903
PremiumTechSupport      21.666903
OnlineSecurity          21.666903
InternetType            21.666903
MultipleLines            9.683374
Married                  0.000000
InternetService          0.000000
PhoneService             0.000000
Offer                    0.000000
Contract                 0.000000
PaperlessBilling         0.000000
PaymentMethod            0.000000
CustomerStatus           0.000000
City                     0.000000
Gender                   0.000000
dtype: float64
```

About 73% of the missing entries are found in the "churn category" and "church cause" columns. From this, it follows that a significant portion of customers who leave a telecommunications firm may fail to provide justification. Because of this, the vast majority of missing values in this instance represent the anticipated informal churning in which no input is provided. Since it is impossible to categorize a client who is leaving without providing a reason, the two columns are related in that they both contain a similar quantity of missing values. Due to the lack of feedback from this churning group, it is reasonable to mark the missing data as "No feedback" to indicate that the numbers are missing.

In [21]:
```
#Replace with No Feedback
df_cat['ChurnReason'].fillna(('nofeedback'),inplace=True)
df_cat['ChurnCategory'].fillna(('nofeedback'),inplace=True)
df_cat.isna().sum().sort_values(ascending = False)
```

Out[21]:
```
PremiumTechSupport      1526
UnlimitedData           1526
InternetType            1526
OnlineSecurity          1526
OnlineBackup            1526
DeviceProtectionPlan    1526
StreamingTV             1526
StreamingMovies         1526
StreamingMusic          1526
MultipleLines            682
ChurnCategory              0
CustomerStatus             0
PaymentMethod              0
PaperlessBilling           0
Contract                   0
Gender                     0
Married                    0
InternetService            0
PhoneService               0
Offer                      0
City                       0
ChurnReason                0
dtype: int64
```

In [22]:
```
# Define a function to fill in values
def filling_null_cat (data):
    values_to_fill = {
        col:data[col].mode()[0]
        for col in data.columns
    }
    return data.fillna(values_to_fill,inplace=True)
```

In [23]:
```
filling_null_cat(df_cat)
df_cat.isna().sum()
```

Out[23]:
```
Gender                 0
Married                0
City                   0
Offer                  0
PhoneService           0
MultipleLines          0
InternetService        0
InternetType           0
OnlineSecurity         0
OnlineBackup           0
DeviceProtectionPlan   0
PremiumTechSupport     0
StreamingTV            0
StreamingMovies        0
StreamingMusic         0
UnlimitedData          0
Contract               0
PaperlessBilling       0
PaymentMethod          0
CustomerStatus         0
ChurnCategory          0
ChurnReason            0
dtype: int64
```

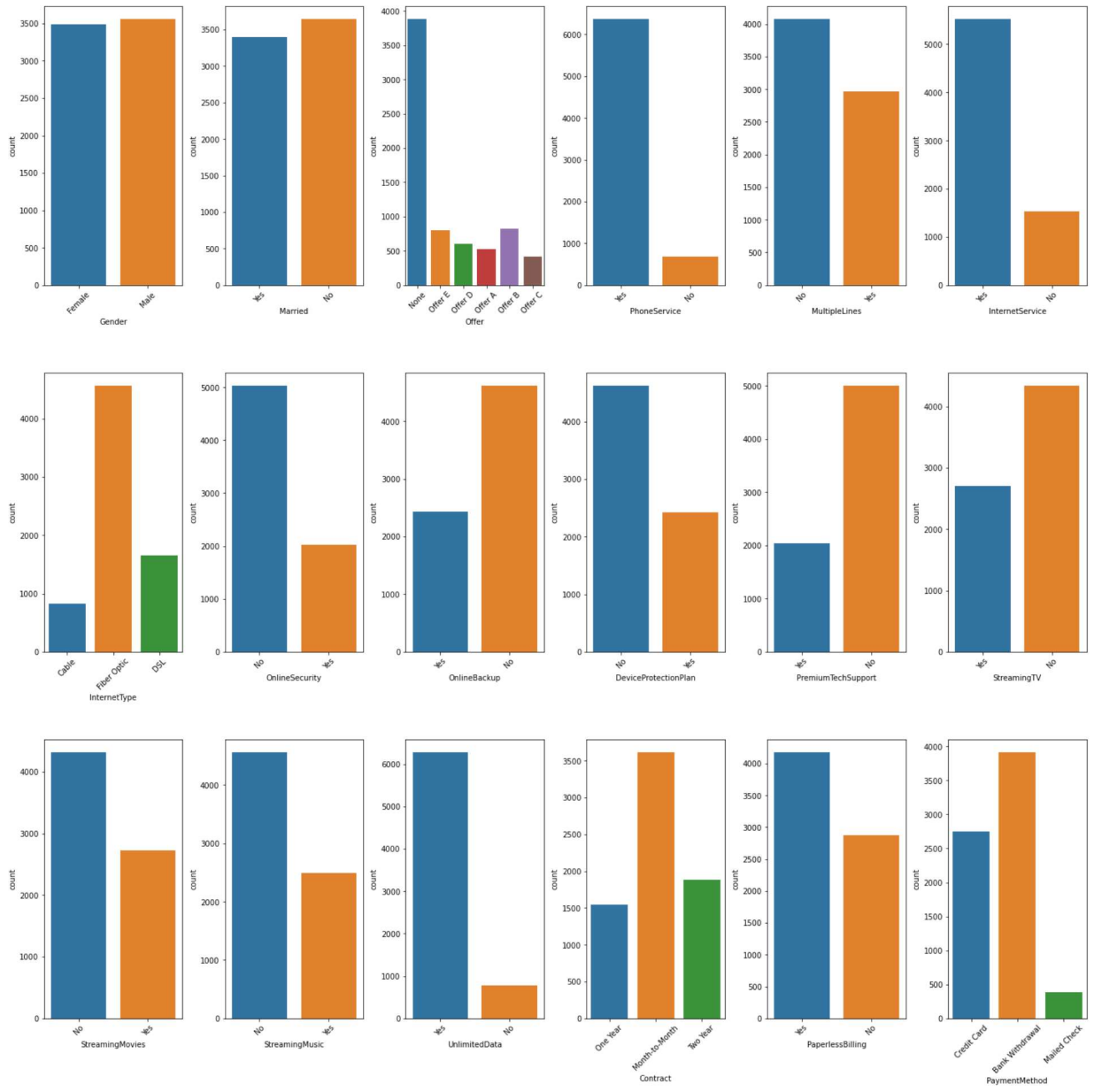
4. Exploratory Data Analysis

In order to better comprehend the patterns in the data and even develop some hypotheses, let's first begin by studying our data set. In order to find any significant trends, we will first examine the distribution of the various variables.

4.1 Univariate Analysis

4.1.1 Analysis of Categorical Features

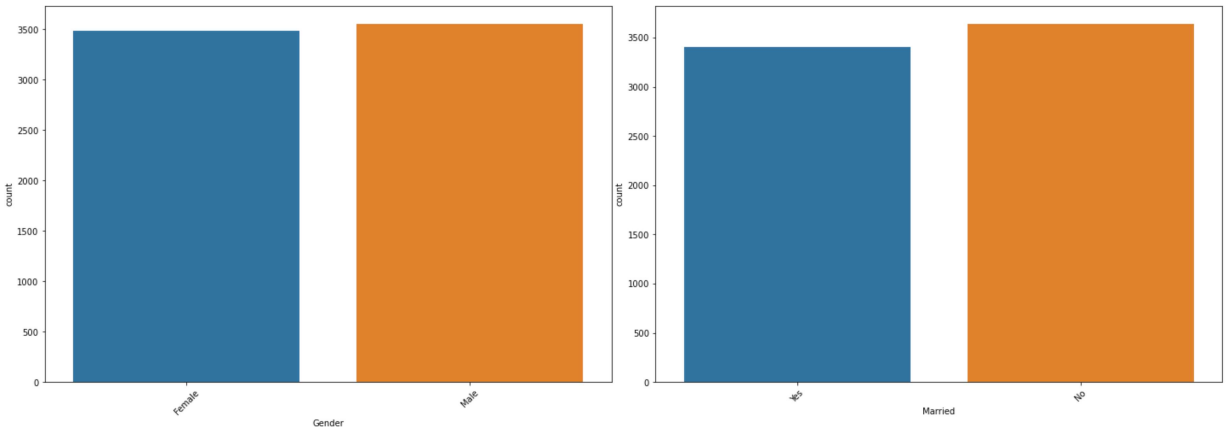
In [24]:
```
cat_interest = ['Gender','Married','Offer','PhoneService','MultipleLines','Interr
fig,ax = plt.subplots(ncols=6,nrows=3,figsize = (20,20))
index = 0
ax = ax.flatten()
for col,value in df_cat[cat_interest].items():
    p = sns.countplot(x = col,data=df_cat[cat_interest],ax = ax[index])
    p.tick_params(axis='x', rotation=45)
    plt.xticks(rotation = "45")
    index += 1
plt.tight_layout(pad= 0.5,w_pad= 0.7,h_pad=5.0);
```



In [25]:
```
# fig,ax = plt.subplots(ncols=5,nrows = 3,figsize = (20,20))
# index = 0
# ax = ax.flatten()
# for col,value in df_num.items():
#     sns.distplot(value,ax = ax[index],
#                  hist=True, kde=False,
#              bins=int(180/5), color = 'darkblue',
#              hist_kws={'edgecolor':'black'},
#              kde_kws={'linewidth': 4})
#     index += 1
# plt.tight_layout(pad= 0.5,w_pad= 0.7,h_pad=5.0);
```

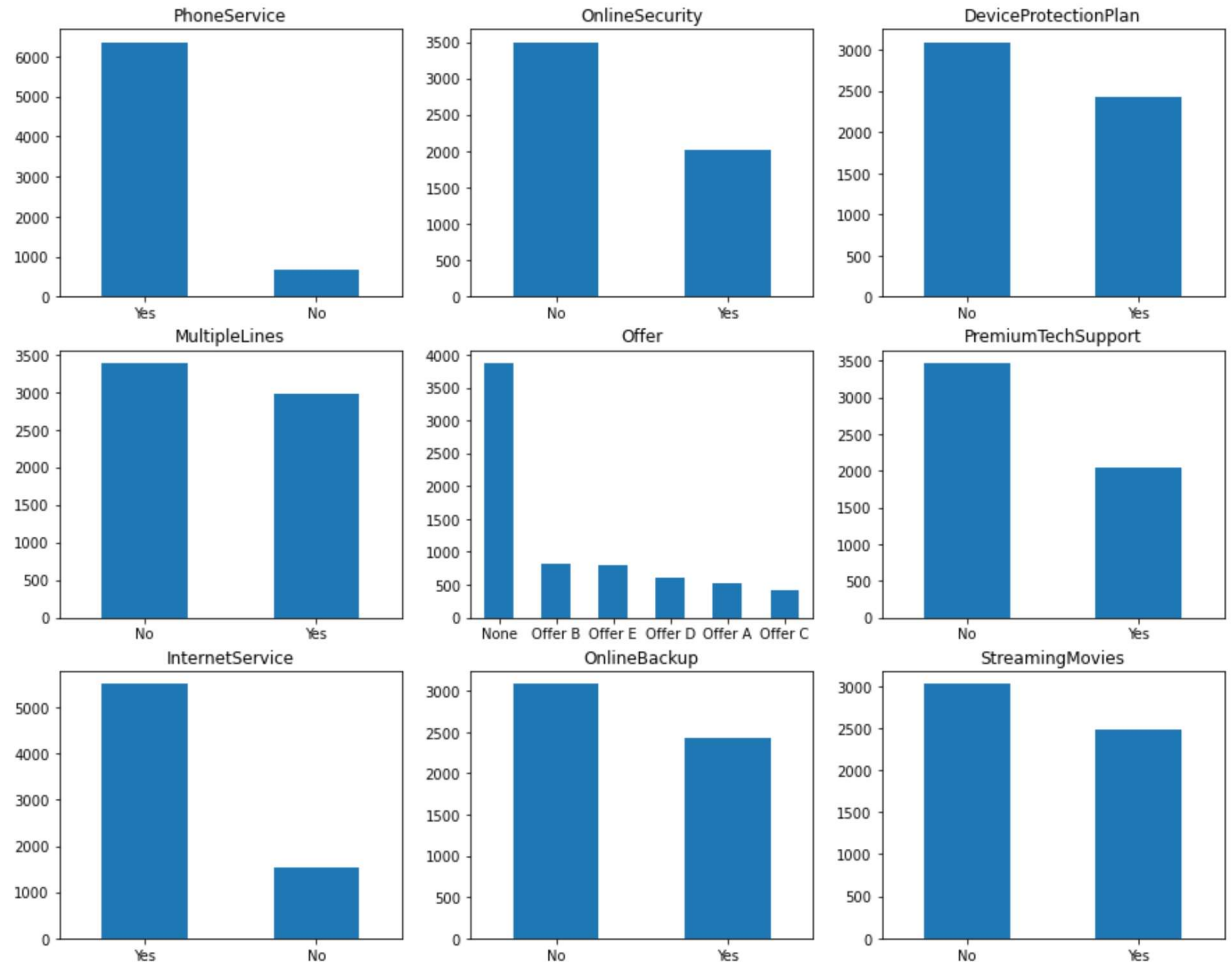
Demographic

In [26]:
```
demo = ['Gender','Married']
fig,ax = plt.subplots(ncols=2,nrows=1,figsize = (20,7))
index = 0
ax = ax.flatten()
for col,value in df_cat[demo].items():
    p = sns.countplot(x = col,data=df_cat[demo],ax = ax[index])
    p.tick_params(axis='x', rotation=45)
    plt.xticks(rotation = "45")
    index += 1
plt.tight_layout(pad= 0.5,w_pad= 0.5,h_pad=3.0);
```

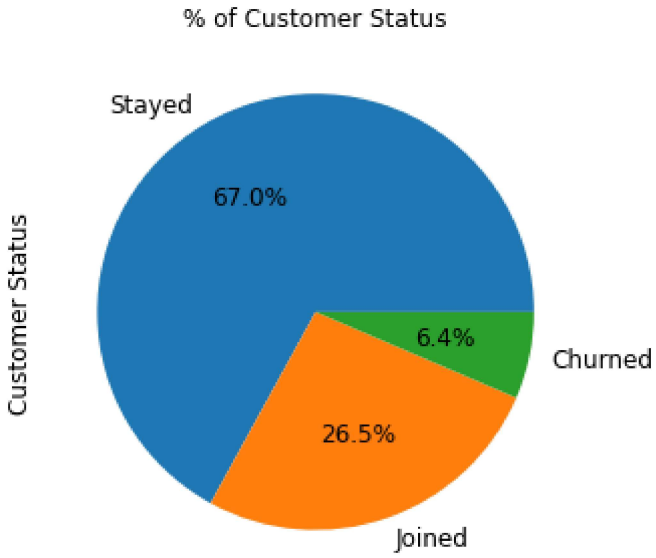


From the dataset the number of male customers was slightly higher than that of females. Most of the customers are also not married.

In [27]:
```
services = ['PhoneService','MultipleLines','InternetService','OnlineSecurity', '
            'OnlineBackup','DeviceProtectionPlan', 'PremiumTechSupport', 'Streamin

fig, axes = plt.subplots(nrows = 3,ncols = 3,figsize = (15,12))
for i, item in enumerate(services):
    if i < 3:
        ax = df[item].value_counts().plot(kind = 'bar',ax=axes[i,0],rot = 0)

    elif i >=3 and i < 6:
        ax = df[item].value_counts().plot(kind = 'bar',ax=axes[i-3,1],rot = 0)

    elif i < 9:
        ax = df[item].value_counts().plot(kind = 'bar',ax=axes[i-6,2],rot = 0)
    ax.set_title(item)
```



In [28]:
```
import matplotlib.ticker as mtick
ax = (df_cat['CustomerStatus'].value_counts()*100.0 /len(df_cat))\
.plot.pie(autopct='%.1f%%', labels = ['Stayed','Joined','Churned'],figsize =(5,5)
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.set_ylabel('Customer Status',fontsize = 12)
ax.set_title('% of Customer Status', fontsize = 12);
```



From the visualization above the dataset has class imbalance. From this, customers who stayed form the majority class and those who churned formed the minority class.

Feature Engineering and Data Preprocessing

In [29]:
```
df_cat.columns
```

Out[29]:
```
Index(['Gender', 'Married', 'City', 'Offer', 'PhoneService', 'MultipleLines',
       'InternetService', 'InternetType', 'OnlineSecurity', 'OnlineBackup',
       'DeviceProtectionPlan', 'PremiumTechSupport', 'StreamingTV',
       'StreamingMovies', 'StreamingMusic', 'UnlimitedData', 'Contract',
       'PaperlessBilling', 'PaymentMethod', 'CustomerStatus', 'ChurnCategory',
       'ChurnReason'],
      dtype='object')
```

In [30]:
```
#Dropping unnecessary features
df_cat = df_cat.drop(['ChurnReason','ChurnCategory','City'],axis=1)
```

In [31]:
```
# Changing categorical variables to numerics
data = pd.concat([df_cat, df_num], axis=1)
```

Train_Test Split

In [32]:
```
#To start out, we'll consider y to be the target variable and everything else to
X = data.drop(['CustomerStatus'],axis=1)
y = data['CustomerStatus']
```

In [33]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.25, random_s
```

Specifying a `random_state` means that we will get consistent results even if the kernel is restarted.

Preprocessing

Categorical

In [34]:
```
X_train_cat = X_train.select_dtypes(exclude=["int64", "float64"])
```

In [35]:
```
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(handle_unknown="ignore", sparse=False)

ohe.fit(X_train_cat)
X_train_ohe = pd.DataFrame(
    ohe.transform(X_train_cat),
    # index is important to ensure we can concatenate with other columns
    index=X_train_cat.index,
    # we are dummying multiple columns at once, so stack the names
    columns=np.hstack(ohe.categories_)
)
```

Numerical

Normalization

Normalizing your data is an additional crucial step in the data preparation process. That instance, if the features are scaled differently, some features may have a greater impact on the model than others. We frequently normalize all features to a constant scale of 0 to 1 to level the playing field.

In [36]:
```
X_train_num = X_train.select_dtypes(include=["int64", "float64"])
```

In [37]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

scaler.fit(X_train_num)
X_train_scaled = pd.DataFrame(
    scaler.transform(X_train_num),
    # index is important to ensure we can concatenate with other columns
    index=X_train_num.index,
    columns=X_train_num.columns
)
```

In [38]:
```
X_train_full = pd.concat([X_train_scaled, X_train_ohe], axis=1)
```

Target Variable

In [39]:
```
from imblearn.over_sampling import SMOTE
# Previous original class distribution
print('Original class distribution: \n')
print(y.value_counts())
smote = SMOTE()
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_full, y_train)
# Preview synthetic sample class distribution
print('----------------------------------------')
print('Synthetic sample class distribution: \n')
print(pd.Series(y_train_resampled).value_counts())
```

```
Original class distribution:

Stayed     4720
Churned    1869
Joined      454
Name: CustomerStatus, dtype: int64
----------------------------------------
Synthetic sample class distribution:

Stayed     3540
Churned    3540
Joined     3540
Name: CustomerStatus, dtype: int64
```

In [40]:
```
# Label encoding the target column
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()
y_train_label = label.fit_transform(y_train_resampled)
y_train_label = pd.Series(y_train_label)
```

Let's now preprocess our test data using the same methodology so that we can assess the model's performance on untested data.

In [41]:
```
# Handling categorical data
X_test_cat = X_test.select_dtypes(exclude=["int64", "float64"])
ohe.fit(X_test_cat)
X_test_ohe = pd.DataFrame(
    ohe.transform(X_test_cat),
    # index is important to ensure we can concatenate with other columns
    index=X_test_cat.index,
    # we are dummying multiple columns at once, so stack the names
    columns=np.hstack(ohe.categories_)
)
X_test_ohe

# Normalization
X_test_num = X_test.select_dtypes(include=["int64", "float64"])
X_test_scaled = pd.DataFrame(
    scaler.transform(X_test_num),
    index=X_test_num.index,
    columns=X_test_num.columns
)

# Concatenating categorical and numeric data
X_test_full = pd.concat([X_test_scaled, X_test_ohe], axis=1)
```

In [42]:
```
# label encoding y_test
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()
y_test_label = label.fit_transform(y_test)
y_test_label=pd.Series(y_test_label)
```

Modelling

In [110]:
```
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.model_selection import ShuffleSplit
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import Lasso
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.linear_model import LinearRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
```

In [127]:
```
from sklearn import svm
model_params = {

        'lasso': {
            'model': Lasso(),
            'params': {
                'alpha': [1,2],
                'selection': ['random', 'cyclic']
            }
        },
    'svm': {
        'model': svm.SVC(gamma='auto'),
        'params' : {
            'C': [1,10,20],
            'kernel': ['rbf','linear']
        }
    },
    'random_forest': {
        'model': RandomForestClassifier(),
        'params' : {
            'n_estimators': [1,5,10],
            'max_leaf_nodes': [10,20,30],
            'max_depth': [2,4,6]
        }
    },
    'logistic_regression' : {
        'model': LogisticRegression(solver='liblinear',multi_class='auto'),
        'params': {
            'C': [1,5,10]
        }
    },
    'logistic_regression' : {
        'model': LogisticRegression(solver='liblinear',multi_class='auto'),
        'params': {}
    },
    'naive_bayes_gaussian': {
        'model': GaussianNB(),
        'params': {}
    },
    'naive_bayes_multinomial': {
        'model': MultinomialNB(),
        'params': {}
    },
    'decision_tree': {
        'model': DecisionTreeClassifier(),
        'params': {
            'criterion': ['gini','entropy'],
            'max_leaf_nodes': [10,20,30],
            'max_depth': [2,4,6]
        }
    }


      }
```

In [128]:
```
from sklearn.model_selection import GridSearchCV
scores = []
cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
for model_name, mp in model_params.items():
    clf =  GridSearchCV(mp['model'], mp['params'], cv=cv, return_train_score=Fals
    clf.fit(X_train_resampled, y_train_label)
    scores.append({
        'model': model_name,
        'best_score': clf.best_score_,
        'best_params': clf.best_params_
    })

df = pd.DataFrame(scores,columns=['model','best_score','best_params'])
df
```

Out[128]:

	model	best_score	best_params
0	lasso	-0.001469	{'alpha': 1, 'selection': 'random'}
1	svm	0.852731	{'C': 20, 'kernel': 'rbf'}
2	random_forest	0.818738	{'max_depth': 6, 'max_leaf_nodes': 30, 'n_esti...
3	logistic_regression	0.785122	{}
4	naive_bayes_gaussian	0.762147	{}
5	naive_bayes_multinomial	0.716573	{}
6	decision_tree	0.819774	{'criterion': 'gini', 'max_depth': 6, 'max_lea...

The random forest model has the highest score (82.2%) in the models listed above. As a result, the model for evaluation will use random forest.

In [117]:
```
# AdaBoost Algorithm
from sklearn.ensemble import AdaBoostClassifier
model = AdaBoostClassifier()

# n_estimators = 50 (default value)
# base_estimator = DecisionTreeClassifier (default value)

model.fit(X_train_resampled,y_train_label)
preds = model.predict(X_test_full)
print(accuracy_score(y_test_label, preds))
```

```
0.6695059625212947
```

The random forest model has the highest score (90.07%) in the models listed above. As a result, the model for evaluation will use random forest.

Model Evaluation

In [124]:
```
model_rf = RandomForestClassifier(n_estimators= 10, oob_score = True, n_jobs = -1
                                  random_state =45, max_features = "auto",
                                  max_leaf_nodes = 30,max_depth=6)
model_rf.fit(X_train_resampled, y_train_label)

# Make predictions
prediction_test = model_rf.predict(X_test_full)
prediction_train = model_rf.predict(X_train_resampled)
print(classification_report(y_train_label, prediction_train))
print(classification_report(y_test_label, prediction_test))
```

```
              precision    recall  f1-score   support

           0       0.81      0.64      0.72      3540
           1       0.82      0.99      0.89      3540
           2       0.86      0.86      0.86      3540

    accuracy                           0.83     10620
   macro avg       0.83      0.83      0.82     10620
weighted avg       0.83      0.83      0.82     10620

              precision    recall  f1-score   support

           0       0.62      0.61      0.62       456
           1       0.58      0.96      0.72       125
           2       0.92      0.86      0.89      1180

    accuracy                           0.80      1761
   macro avg       0.71      0.81      0.74      1761
weighted avg       0.82      0.80      0.81      1761
```

Result Interpretation

Precision — What percent of your predictions were correct? Based on the above findings, the model accurately predicts that one will discontinue utilizing the company's services 61% of the time.

Recall captures What percent of the positive cases the model predicts. This model captures 91% of the positive cases.

F1 score answers the question of what percent of positive predictions were correct? On this, the model captures ~73% of correct positive predictions.

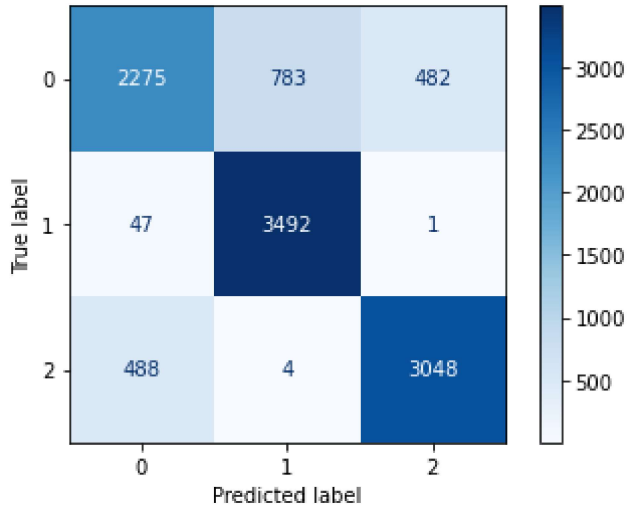
Accuracy, as its name implies, takes into account the model's overall accuracy, however this is only true when the model is balanced. This model balances for the situation at hand, accurately projecting the outcomes 81% of the time.

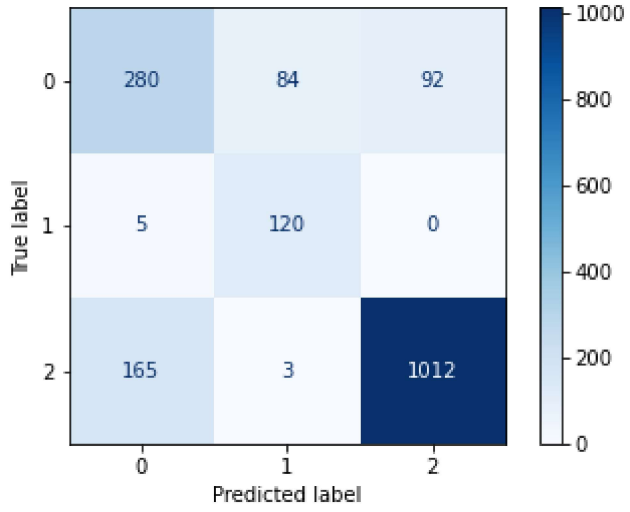
To evaluate the effectiveness of a machine learning classification, confusion matrix will be used. It takes the form of a matrix.

In [125]:
```
# Create a confusion matrix
cnf_train = confusion_matrix(y_train_label, prediction_train)
cnf_test = confusion_matrix(y_test_label, prediction_test)
print('Confusion Matrix:\n', cnf_train)
print('Confusion Matrix:\n', cnf_test)
```

```
Confusion Matrix:
 [[2275  783  482]
 [  47 3492    1]
 [ 488    4 3048]]
Confusion Matrix:
 [[ 280   84   92]
 [   5  120    0]
 [ 165    3 1012]]
```

In [126]:
```
# Visualize our confusion matrix
plot_confusion_matrix(model_rf, X_train_resampled, y_train_label,
                      cmap=plt.cm.Blues)
plot_confusion_matrix(model_rf, X_test_full, y_test_label,
                      cmap=plt.cm.Blues)
plt.show()
```





From the confusion matrix out of 456 customers that the model predicted will churn,280 were correct

Conclusions

When customers leave, they are going to the competition. Understanding churn factors will not only allow Telecom to understand why their customers are leaving but also to what extent the factors that lead to customer dissatisfaction affect the company. Overall, this will lead to the opportunity for Telecom to sharpen its attractiveness in the eyes of its customers by competing in the market well.

RECOMMENDATIONS

1.Telecom should recruit more customers to their offer packages because there is a clear indication that it reduces customer churning.

2.Telecom should improve its relationship with new customers because we found out that the highest churning rates occur in the first 10 months.

3.Telecom should make premium tech support more attractive to customers because customers on the cover have low churn rates.

4.Telecom should make changes to their internet provision services as they all have high

In [ ]:

In [ ]: