In [160…
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import RobustScaler,MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
```

In [163…
```
Df= pd.read_csv(r"C:\Users\awura\Downloads\Customer-Churn.csv")
Df
```

Out[163]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLin
0	7590-VHVEG	Female	0	Yes	No	1	No	No phor servi
1	5575-GNVDE	Male	0	No	No	34	Yes	N
2	3668-QPYBK	Male	0	No	No	2	Yes	N
3	7795-CFOCW	Male	0	No	No	45	No	No phor servi
4	9237-HQITU	Female	0	No	No	2	Yes	N
...	...	...	...	...	...	...	...	
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	Y
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	Y
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	No phor servi
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	Y
7042	3186-AJIEK	Male	0	No	No	66	Yes	N

7043 rows × 21 columns

In [165…
```
Df1= Df.copy()
Df1
```

Out[165]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLin
0	7590-VHVEG	Female	0	Yes	No	1	No	No phor servi
1	5575-GNVDE	Male	0	No	No	34	Yes	N
2	3668-QPYBK	Male	0	No	No	2	Yes	N
3	7795-CFOCW	Male	0	No	No	45	No	No phor servi
4	9237-HQITU	Female	0	No	No	2	Yes	N
...	...	...	...	...	...	...	...	
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	Y
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	Y
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	No phor servi
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	Y
7042	3186-AJIEK	Male	0	No	No	66	Yes	N

7043 rows × 21 columns

In [166...

```
Df2=Df.copy()
Df2
```

Out[166]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLin
0	7590-VHVEG	Female	0	Yes	No	1	No	No phor servi
1	5575-GNVDE	Male	0	No	No	34	Yes	N
2	3668-QPYBK	Male	0	No	No	2	Yes	N
3	7795-CFOCW	Male	0	No	No	45	No	No phor servi
4	9237-HQITU	Female	0	No	No	2	Yes	N
...	...	...	...	...	...	...	...	
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	Y
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	Y
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	No phor servi
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	Y
7042	3186-AJIEK	Male	0	No	No	66	Yes	N

7043 rows × 21 columns

In [167…

```
Df.dtypes
```

Out[167]:

```
customerID           object
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 [168…
```
Df["TotalCharges"]=pd.to_numeric(Df["TotalCharges"],errors="coerce")
Df["TotalCharges"]
```

Out[168]:
```
0         29.85
1       1889.50
2        108.15
3       1840.75
4        151.65
         ...
7038    1990.50
7039    7362.90
7040     346.45
7041     306.60
7042    6844.50
Name: TotalCharges, Length: 7043, dtype: float64
```

In [169…
```
Df.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      7032 non-null   float64
 20  Churn             7043 non-null   object
dtypes: float64(2), int64(2), object(17)
memory usage: 1.1+ MB
```

In [170…
```
Df.isnull().sum()
```

Out[170]:
```
customerID           0
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        11
Churn                0
dtype: int64
```

In [171…
```
Df[Df.duplicated()]
```

Out[171]:

customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	I

0 rows × 21 columns

In [172…
```
### 11 missing values from Total Charges
### No duplicates
```

In [173…
```
num=[]
cat=[]

for col in Df.columns:
    if Df[col].dtypes=="object":
        cat.append(col)
    else:
        num.append(col)
```

In [174…
```
num
```

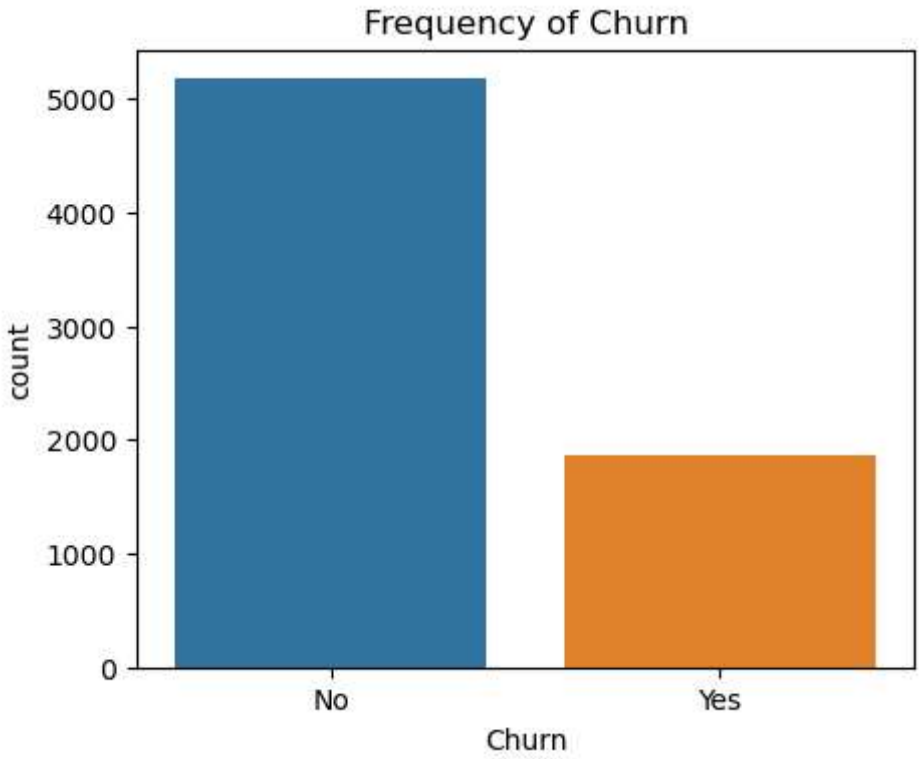
Out[174]:
```
['SeniorCitizen', 'tenure', 'MonthlyCharges', 'TotalCharges']
```

In [175…
```
cat
```

Out[175]:
```
['customerID',
 'gender',
 'Partner',
 'Dependents',
 'PhoneService',
 'MultipleLines',
 'InternetService',
 'OnlineSecurity',
 'OnlineBackup',
 'DeviceProtection',
 'TechSupport',
 'StreamingTV',
 'StreamingMovies',
 'Contract',
 'PaperlessBilling',
 'PaymentMethod',
 'Churn']
```

Univariate Analysis

In [176…
```
###looking for the count of chrun
plt.figure(figsize=(5,4))
sns.countplot(data=Df , x="Churn")

plt.title("Frequency of Churn")
plt.show();
```



In [177…
```
Df.groupby("Churn").count() ### The number of customers still with the company and
```

Out[177]:

Churn	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLi
No	5174	5174	5174	5174	5174	5174	5174	5
Yes	1869	1869	1869	1869	1869	1869	1869	1

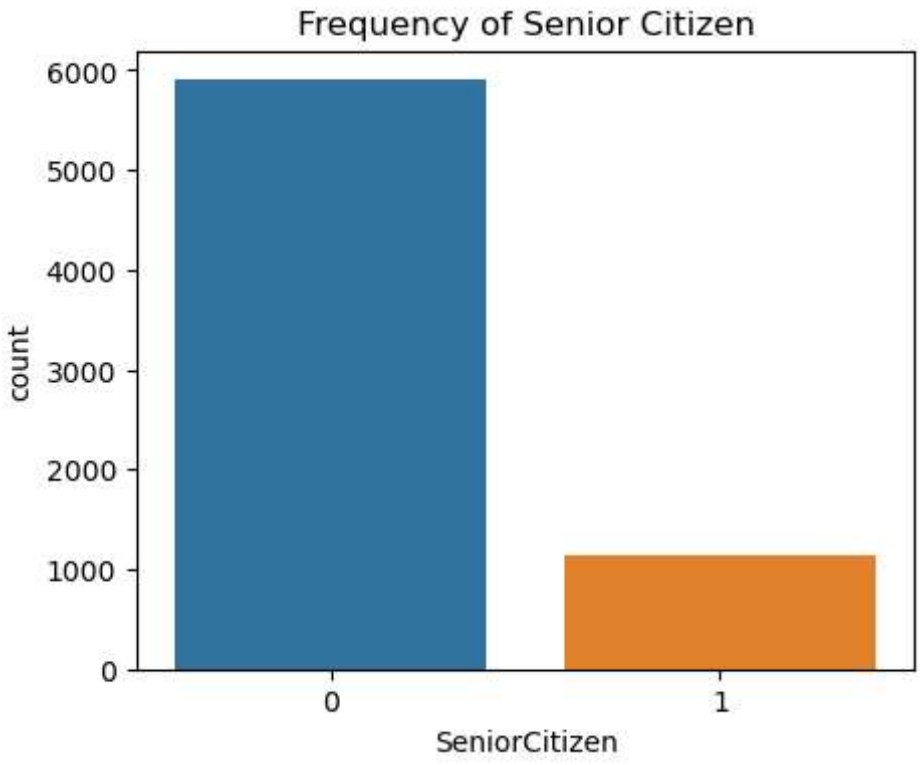
In [178…
```
Df.head()
```

Out[178]:

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

5 rows × 21 columns

In [179…
```
###looking for the count of chrun
plt.figure(figsize=(5,4))
sns.countplot(data=Df , x="SeniorCitizen")

plt.title("Frequency of Senior Citizen")
plt.show();
```

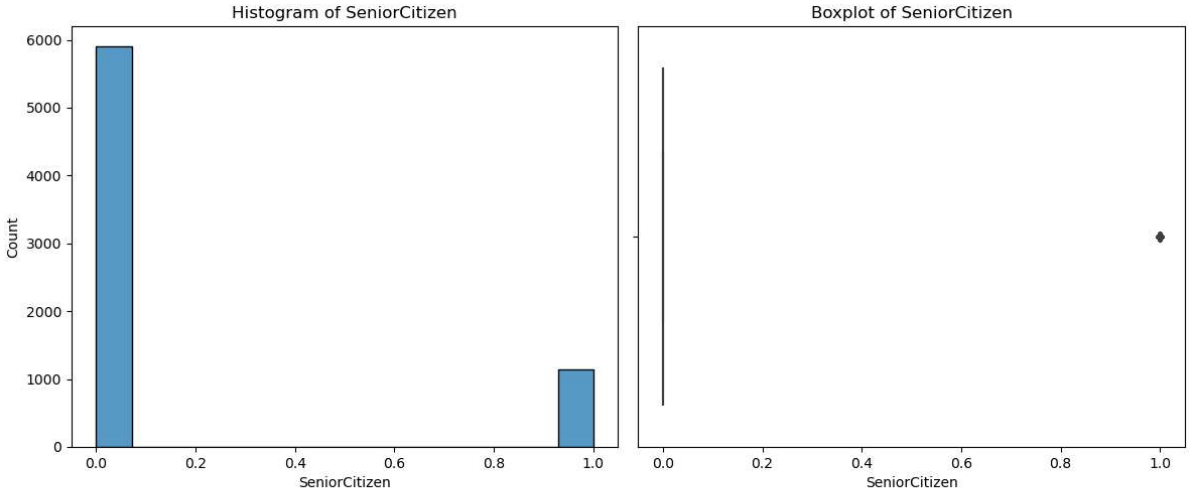


In [180…
```
for index, name in enumerate(num):
    print(index,name)
```

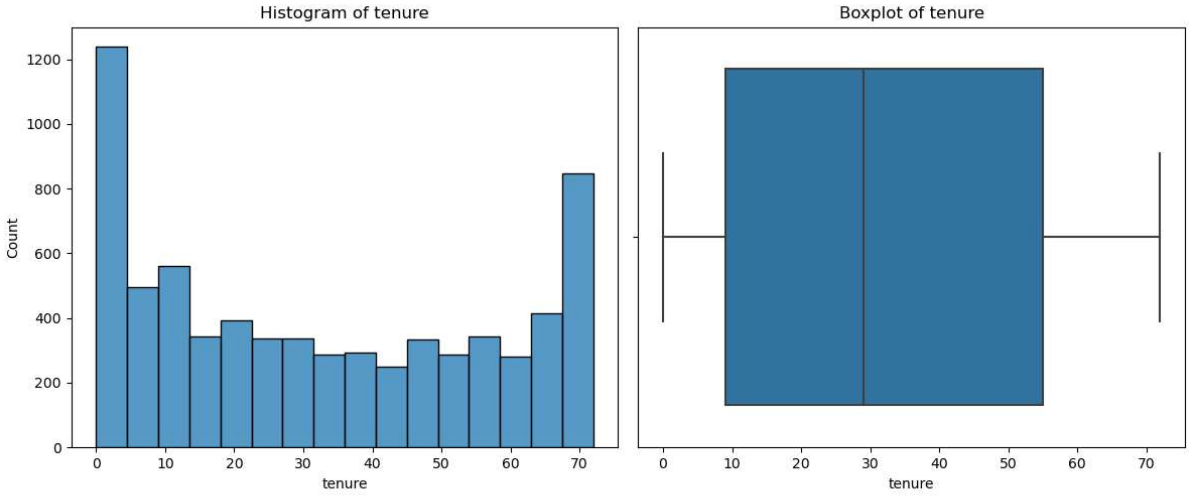
```
0 SeniorCitizen
1 tenure
2 MonthlyCharges
3 TotalCharges
```

In [181…

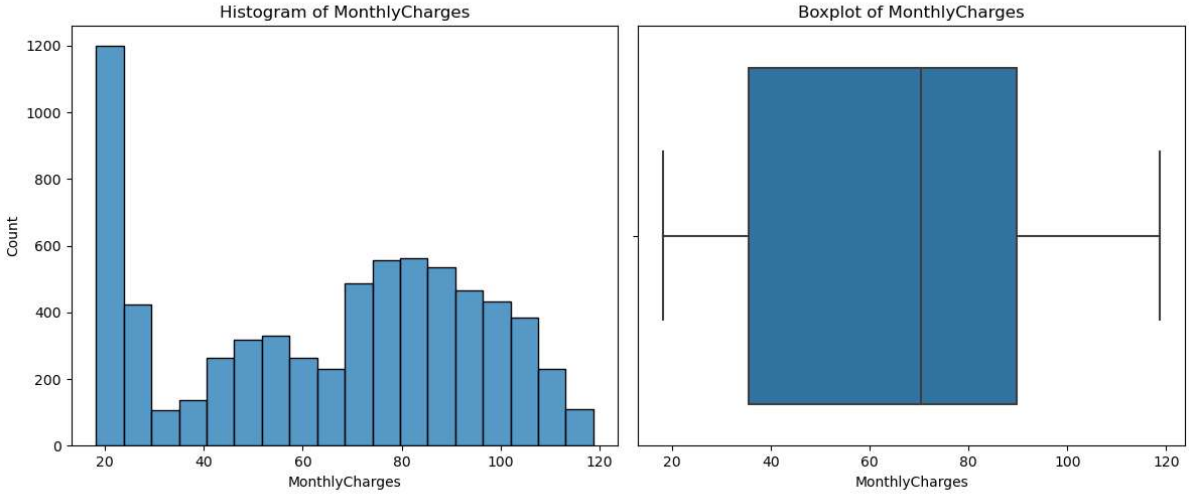
```python
fig, axes= plt.subplots(4,2, figsize=(12,20))
for index, col in enumerate(num):

    #histogram
    sns.histplot(data = Df, x = col, ax = axes[index, 0])
    axes[index, 0].set_title(f'Histogram of {col}')

    #boxplot
    sns.boxplot(data = Df, x = col, ax = axes[index, 1])
    axes[index, 1].set_title(f'Boxplot of {col}')


plt.tight_layout()
plt.show()
```


Histogram of SeniorCitizen
Count
SeniorCitizen

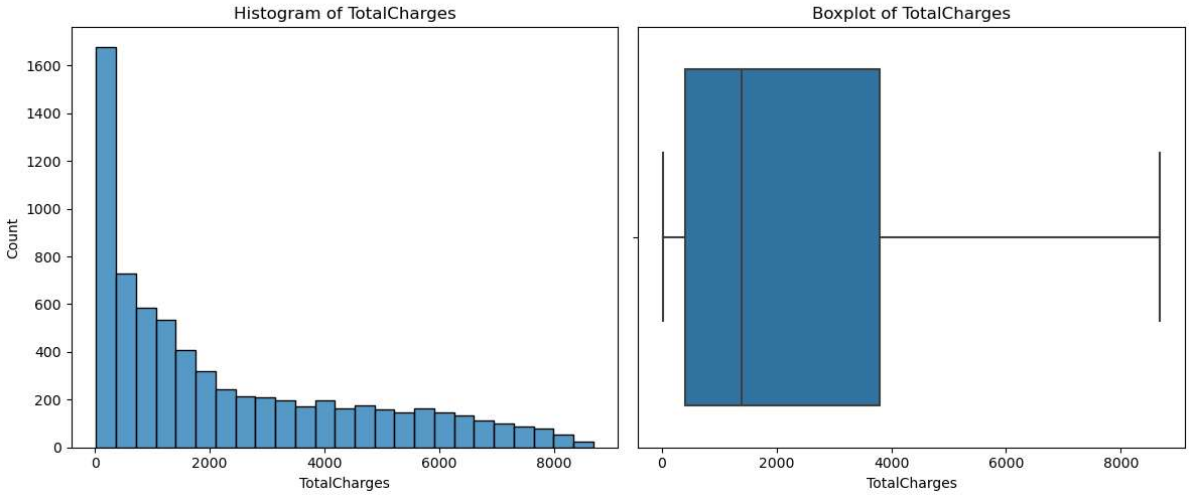
Boxplot of SeniorCitizen
SeniorCitizen


Histogram of tenure
Count
tenure

Boxplot of tenure
tenure


Histogram of MonthlyCharges
Count
MonthlyCharges

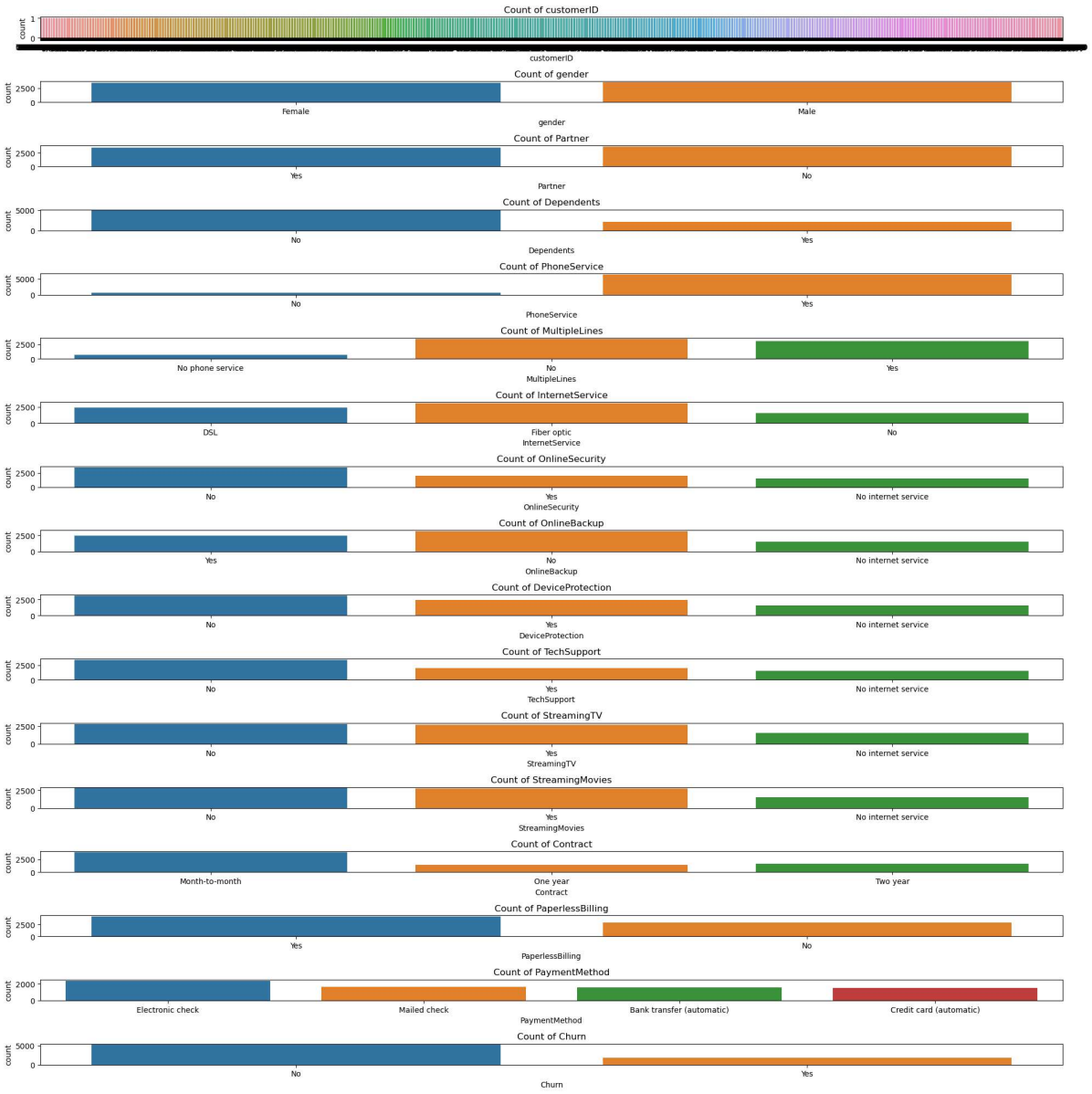
Boxplot of MonthlyCharges
MonthlyCharges


Histogram of TotalCharges
Count
TotalCharges

Boxplot of TotalCharges
TotalCharges

In [182…

```
fig, axes= plt.subplots(17,1, figsize=(20,20))
for index, col in enumerate(cat):


    sns.countplot(data = Df, x = col, ax = axes[index])
    axes[index].set_title(f'Count of {col}')


plt.tight_layout()
plt.show()
```



Bivariate Analysis

Analysis on impact on some attritubes on Churn

In [184…

```
### Impact of gender on Churn
Df.groupby("gender")["customerID"].count()
```

Out[184]:

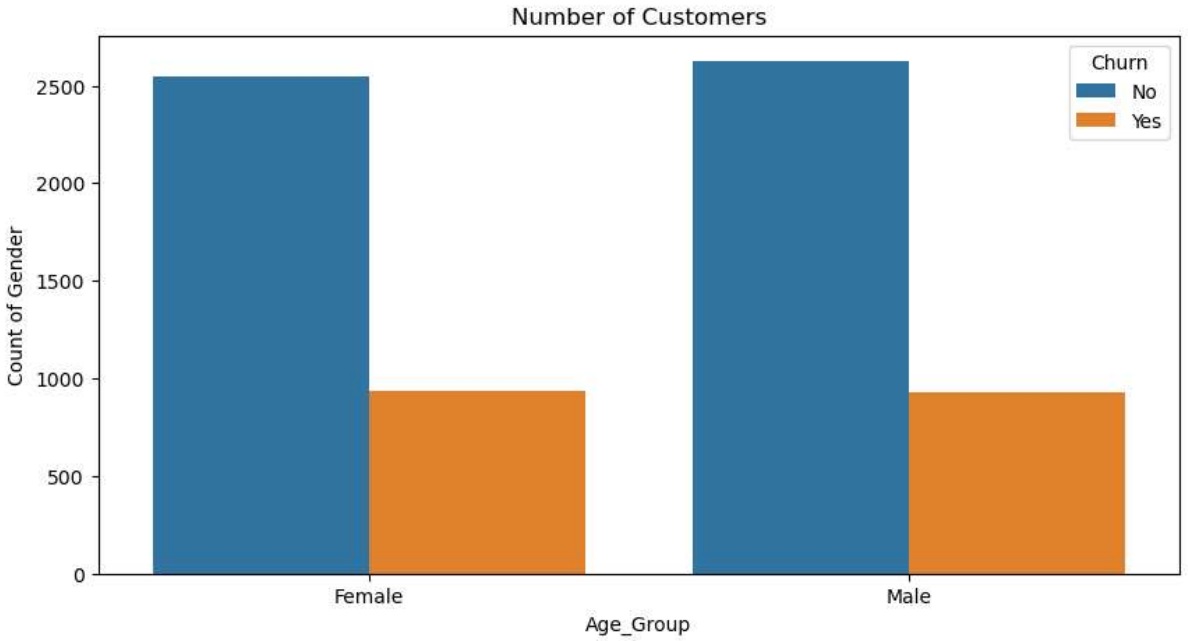
```
gender
Female    3488
Male      3555
Name: customerID, dtype: int64
```

In [185…

```
Df.groupby(['gender', 'Churn'])['customerID'].count()
```

Out[185]:
```
gender  Churn
Female  No       2549
        Yes       939
Male    No       2625
        Yes       930
Name: customerID, dtype: int64
```

In [186…
```
plt.figure(figsize=(10,5))
sns.countplot(data=Df, x="gender", hue="Churn")
plt.xlabel("Age_Group")
plt.ylabel("Count of Gender")
plt.title("Number of Customers");
plt.show()
```



MOre females are churned compared to the number of males

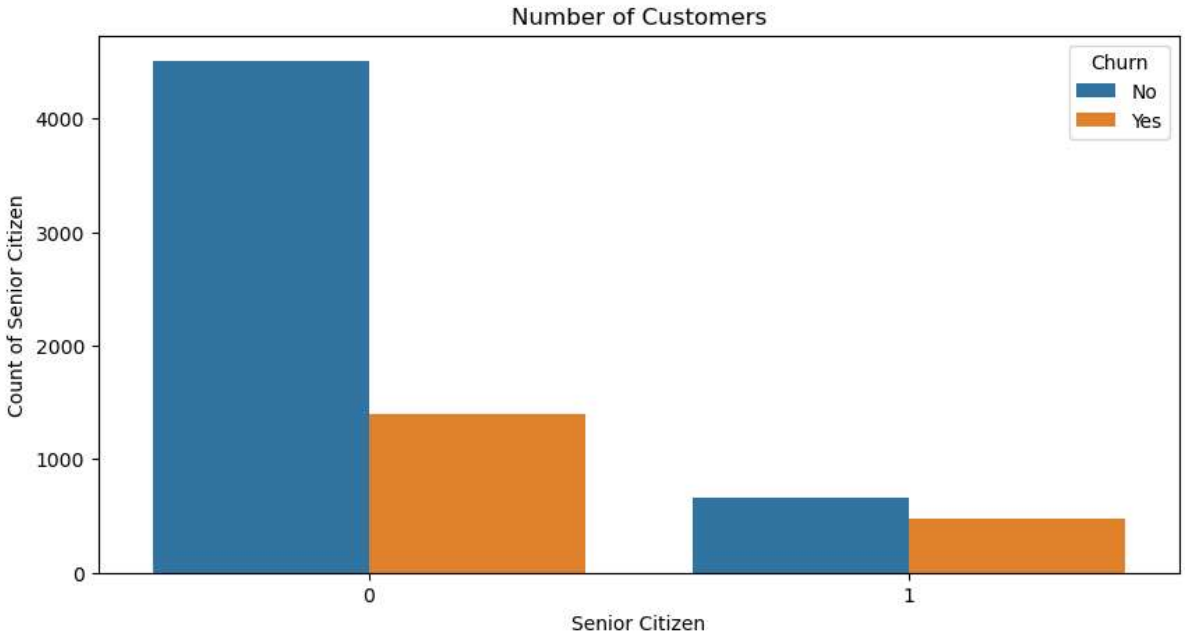
In [187…
```
##Impact of Senior Citizen on Churn
Df.groupby('SeniorCitizen')['customerID'].count()
```

Out[187]:
```
SeniorCitizen
0    5901
1    1142
Name: customerID, dtype: int64
```

In [188…
```
Df.groupby(['SeniorCitizen', 'Churn'])['customerID'].count()
```

Out[188]:
```
SeniorCitizen  Churn
0              No       4508
               Yes      1393
1              No        666
               Yes       476
Name: customerID, dtype: int64
```

In [189…
```
plt.figure(figsize=(10,5))
sns.countplot(x = 'SeniorCitizen', data = Df, hue = 'Churn')

plt.xlabel("Senior Citizen")
plt.ylabel("Count of Senior Citizen")
plt.title("Number of Customers");
plt.show()
```



From the analysis made, those who are senior citizen had higher rate of churned services relative to those who are not

In [190…

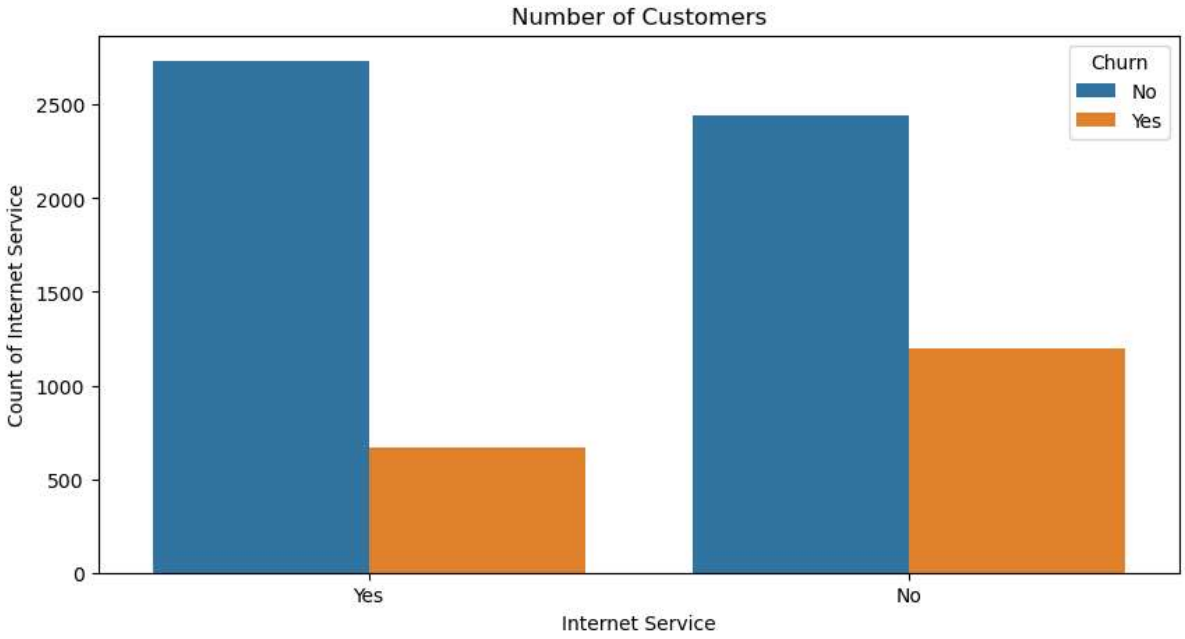
```
###Impact of InternetService on Churn
Df.groupby(['InternetService', 'Churn'])['customerID'].count()
```

Out[190]:

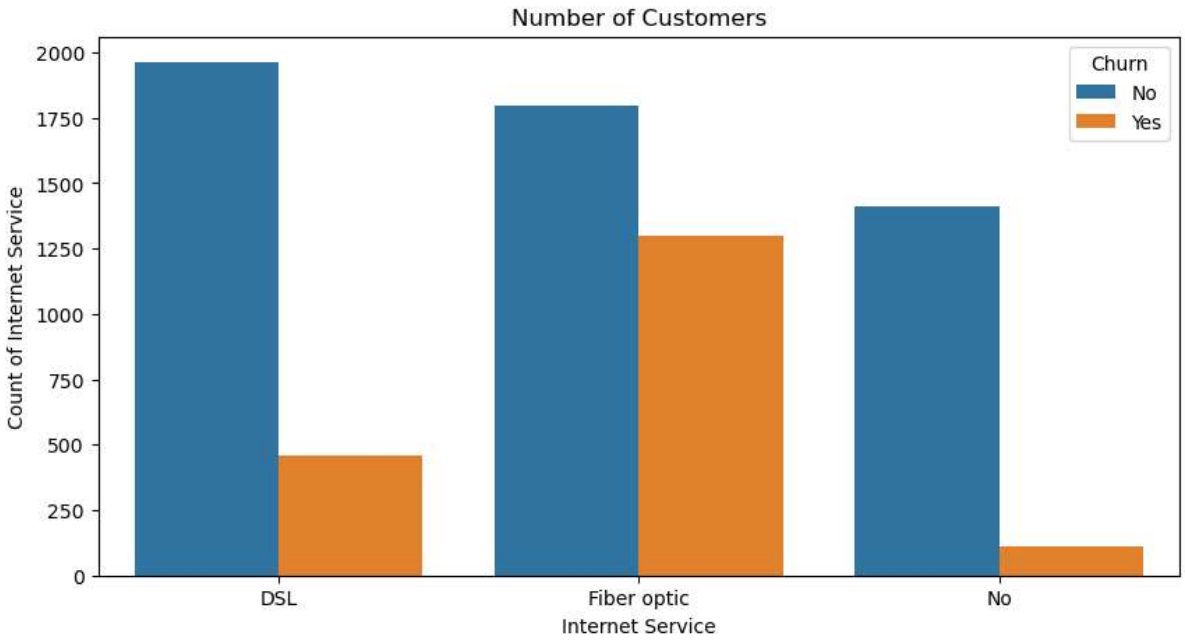
```
InternetService  Churn
DSL              No       1962
                 Yes       459
Fiber optic      No       1799
                 Yes      1297
No               No       1413
                 Yes       113
Name: customerID, dtype: int64
```

In [191…

```
plt.figure(figsize=(10,5))
sns.countplot(x = 'Partner', data = Df, hue = 'Churn')

plt.xlabel("Internet Service")
plt.ylabel("Count of Internet Service")
plt.title("Number of Customers");
plt.show()
```



In [64]:
```
plt.figure(figsize=(10,5))
sns.countplot(x = 'InternetService', data = Df, hue = 'Churn')

plt.xlabel("Internet Service")
plt.ylabel("Count of Internet Service")
plt.title("Number of Customers");
plt.show()
```



From the analysis on internet service, fiber optic had higher numbers and rate of churned services compared to the other

services

Hence, services on fiber optic needs to be improved to keep its customers

In [192…
```
## impact of online security on churn
Df.groupby('OnlineSecurity')['customerID'].count()
```

Out[192]:
```
OnlineSecurity
No                     3498
No internet service    1526
Yes                    2019
Name: customerID, dtype: int64
```

In [193…
```
Df.groupby(['OnlineSecurity', 'Churn'])['customerID'].count()
```

Out[193]:
```
OnlineSecurity       Churn
No                   No       2037
                     Yes      1461
No internet service  No       1413
                     Yes       113
Yes                  No       1724
                     Yes       295
Name: customerID, dtype: int64
```

In [194…
```
plt.figure(figsize=(10,5))
sns.countplot(x = "OnlineSecurity", data=Df, hue = 'Churn')

plt.xlabel("Online Security")
plt.ylabel("Count of Online Security")
plt.title("Number of Customers");
plt.show()
```

The projection shows that most of the churned services were from customers with no online security

In order to keep customers, the teleco has to work on providing online security to its customers

In [ ]:
```
##Impact on technical Support on churn
```

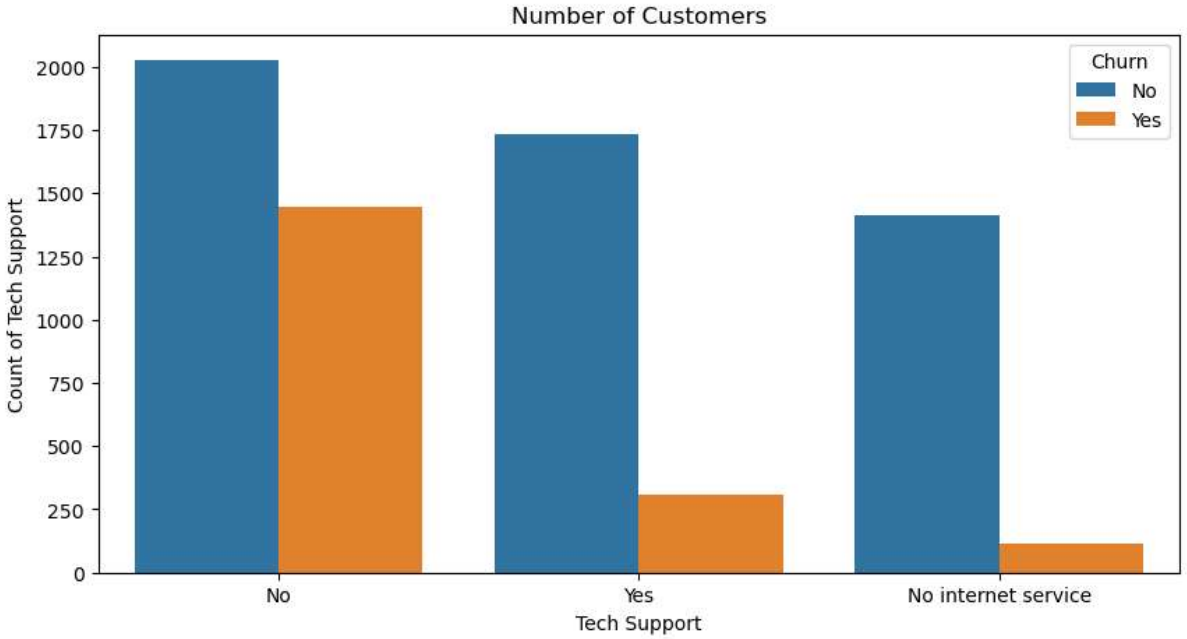
In [195…
```
Df.groupby('TechSupport')['customerID'].count()
```

Out[195]:
```
TechSupport
No                     3473
No internet service    1526
Yes                    2044
Name: customerID, dtype: int64
```

In [196…
```
Df.groupby(['TechSupport', 'Churn'])['customerID'].count()
```

Out[196]:
```
TechSupport          Churn
No                   No       2027
                     Yes      1446
No internet service  No       1413
                     Yes       113
Yes                  No       1734
                     Yes       310
Name: customerID, dtype: int64
```

In [197…
```
plt.figure(figsize=(10,5))
sns.countplot(x = 'TechSupport', data = Df, hue = 'Churn')

plt.xlabel("Tech Support")
plt.ylabel("Count of Tech Support")
plt.title("Number of Customers");
plt.show()
```



The visualization shows that large number of customers had no technical support and most of those customers churned services

The teleco has to work on improving and providing technical support to its customers

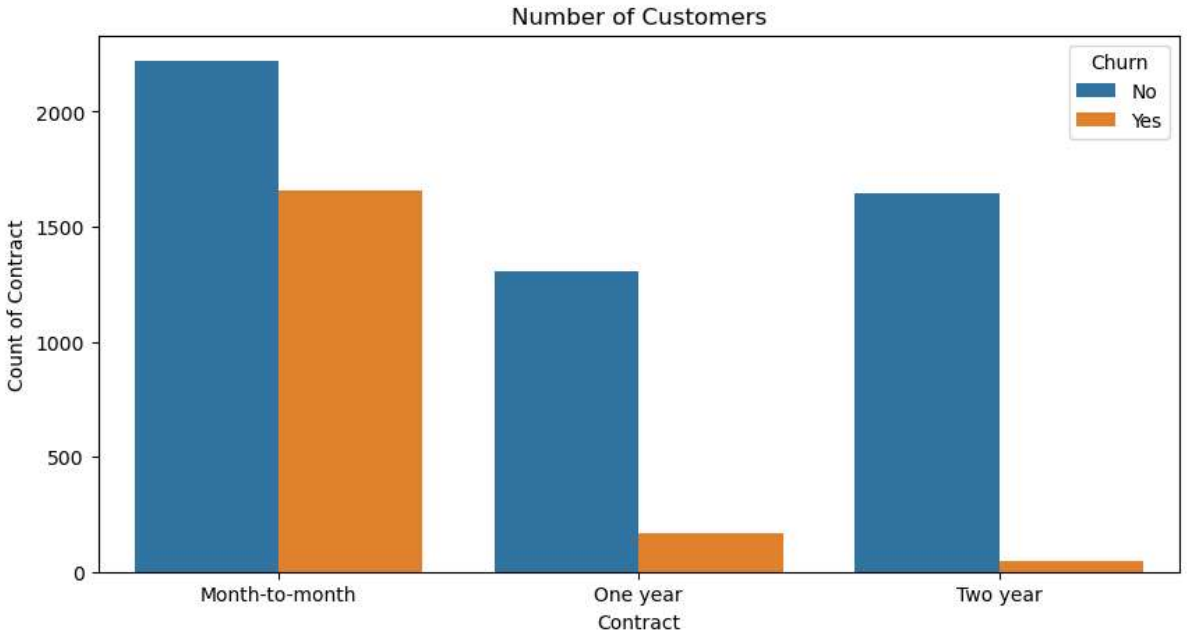
"""

In [ ]:
```
## the impact of contract on churn
```

In [198…
```
Df.groupby(['Contract', 'Churn'])['customerID'].count()
```

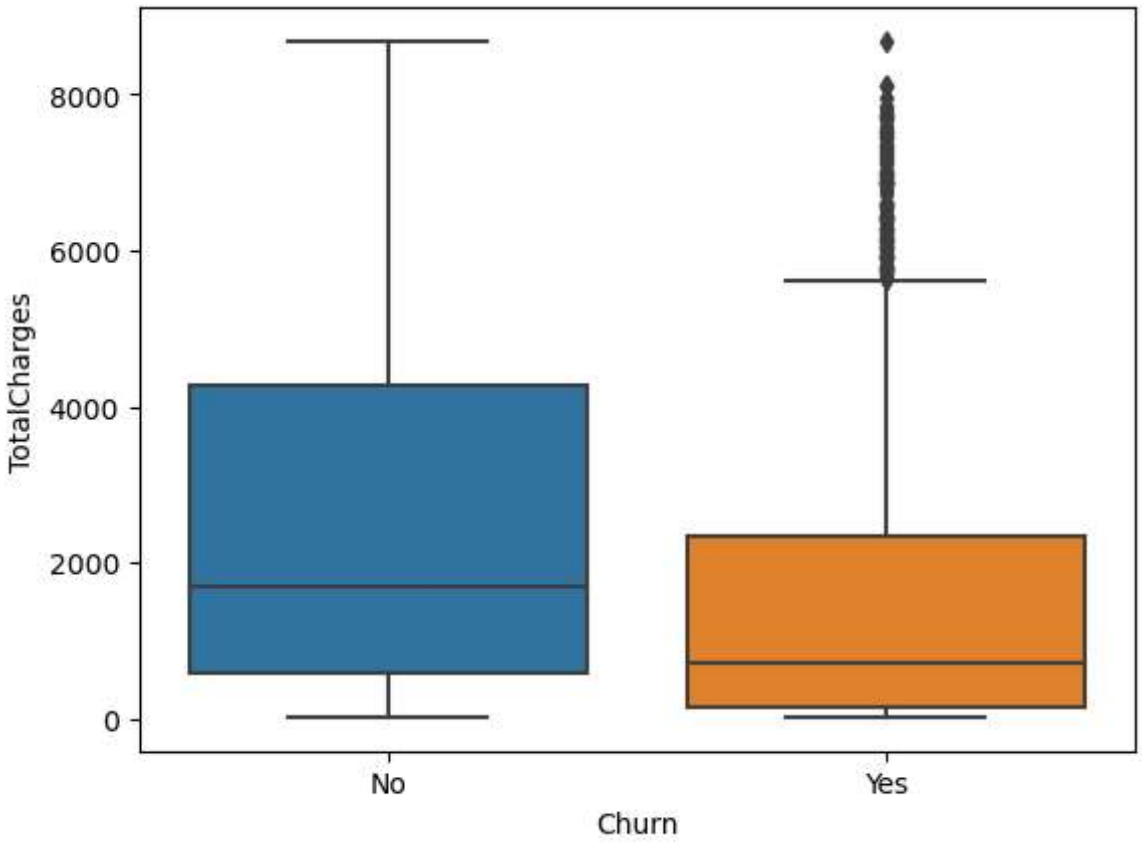
Out[198]:
```
Contract        Churn
Month-to-month  No       2220
                Yes      1655
One year        No       1307
                Yes       166
Two year        No       1647
                Yes        48
Name: customerID, dtype: int64
```

In [199…
```
plt.figure(figsize=(10,5))
sns.countplot(x = 'Contract', data = Df, hue = 'Churn')

plt.xlabel("Contract")
plt.ylabel("Count of Contract")
plt.title("Number of Customers");
plt.show()
```

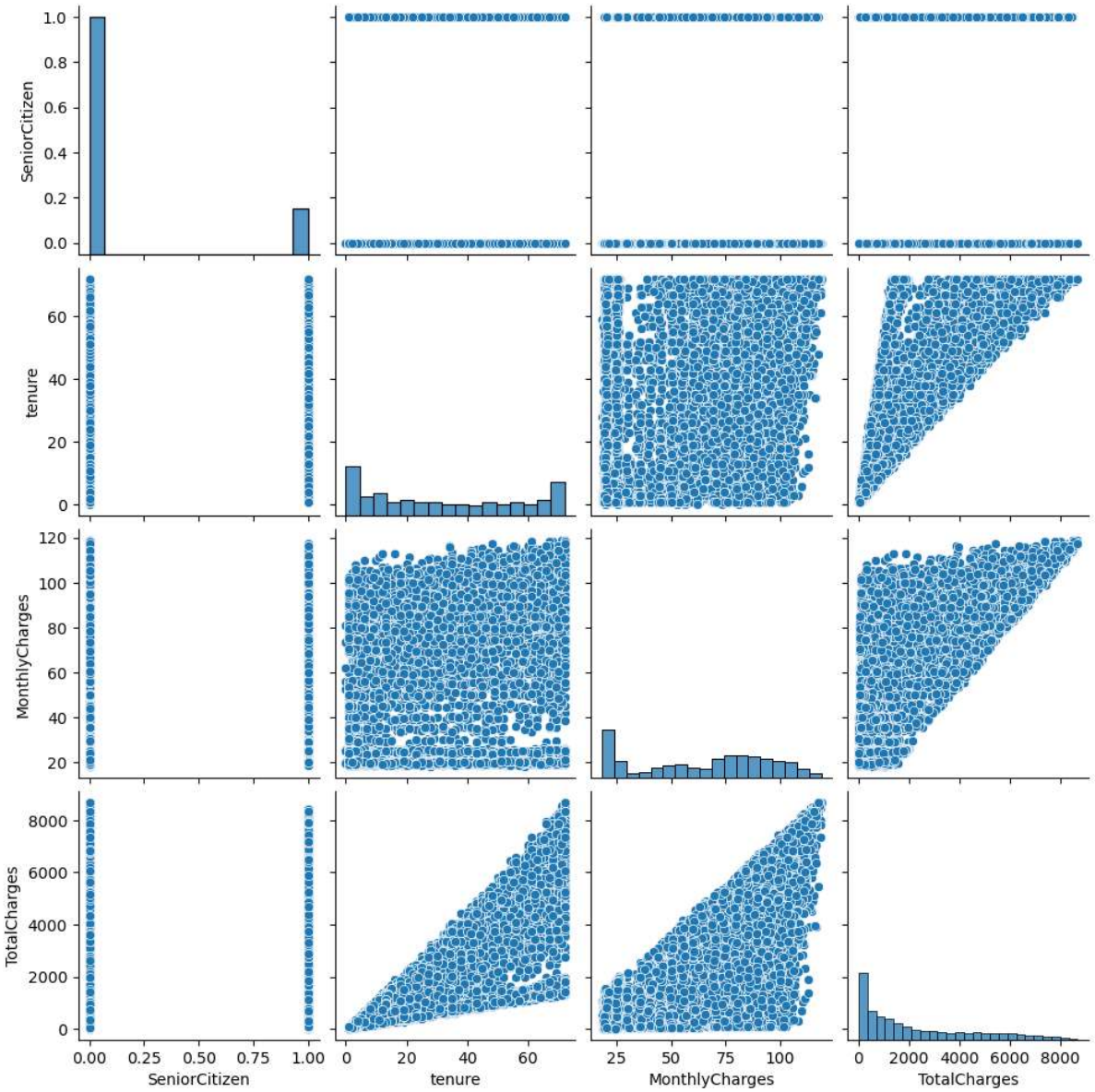


The projection shows that most of the churned services were from customers with month to month contract

In [200…
```
sns.boxplot(x = 'Churn', y = 'TotalCharges', data = Df)
plt.show()
```



In [71]:
```
sns.pairplot(Df, diag_kind = "hist");
```



Multivariate

In [202…

```
## correlation
Df.corr()
```

```
C:\Users\awura\AppData\Local\Temp\ipykernel_20408\752224673.py:2: FutureWarning: T
he default value of numeric_only in DataFrame.corr is deprecated. In a future vers
ion, it will default to False. Select only valid columns or specify the value of n
umeric_only to silence this warning.
  Df.corr()
```

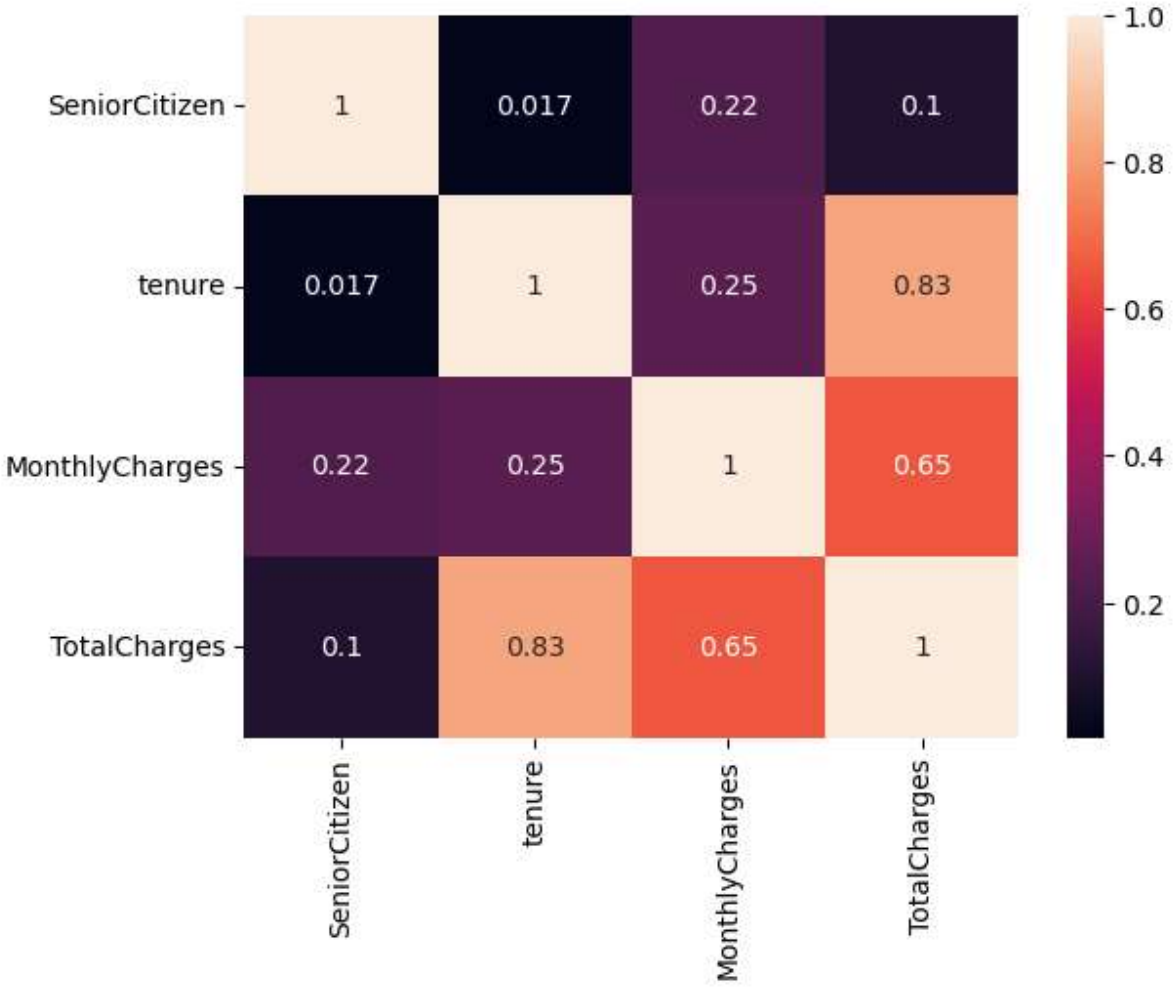
Out[202]:

	SeniorCitizen	tenure	MonthlyCharges	TotalCharges
SeniorCitizen	1.000000	0.016567	0.220173	0.102411
tenure	0.016567	1.000000	0.247900	0.825880
MonthlyCharges	0.220173	0.247900	1.000000	0.651065
TotalCharges	0.102411	0.825880	0.651065	1.000000

In [203…

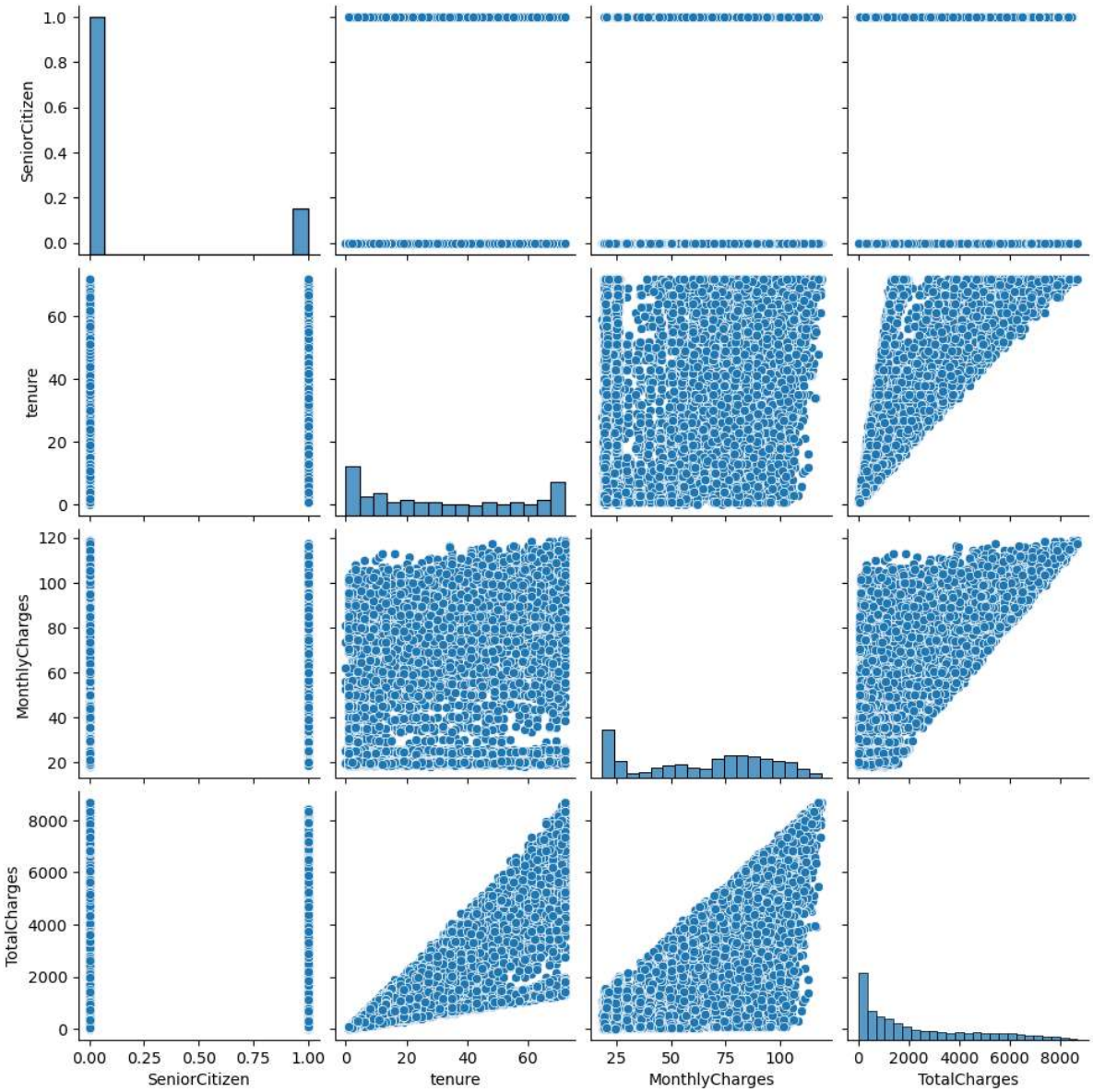
```
sns.heatmap(Df.corr(), annot = True)
plt.show()
```

```
C:\Users\awura\AppData\Local\Temp\ipykernel_20408\1824563537.py:1: FutureWarning:
The default value of numeric_only in DataFrame.corr is deprecated. In a future ver
sion, it will default to False. Select only valid columns or specify the value of
numeric_only to silence this warning.
  sns.heatmap(Df.corr(), annot = True)
```



""" Each attritube is highly correlated to itself Tenure and TotalCharges have the next high correlation SeniorCitizen and Tenure have the least correlation between each other """

In [70]:



Machine learning

In [ ]:
```
### Observtions from the EDA about the target variable and target features

### No missing features in the churn variable
### we can observe the presence of outliiers in one of the target features ie Total
### There are no duplicates in each of the columns
### There was 11 missing values found in one of the target features ie Total Charge
## Column, "Churn" is the target y-variable and is a categorical value: Yes vs. No
```

In [204…
```
Df.head()
```

Out[204]:

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

5 rows × 21 columns

In [206…

```
Df.dtypes
```

Out[206]:

```
customerID           object
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        float64
Churn                object
dtype: object
```

Feature Engineering

In [207…

```
## Creating a new column to covert the churn into an integer
def label(x):
    if x== "Yes":
        return 1
    else:
        return 0
Df["Churn_label"]=Df["Churn"].apply(label)
Df
```

Out[207]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLin
0	7590-VHVEG	Female	0	Yes	No	1	No	No phor servi
1	5575-GNVDE	Male	0	No	No	34	Yes	N
2	3668-QPYBK	Male	0	No	No	2	Yes	N
3	7795-CFOCW	Male	0	No	No	45	No	No phor servi
4	9237-HQITU	Female	0	No	No	2	Yes	N
...	...	...	...	...	...	...	...	
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	Y
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	Y
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	No phor servi
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	Y
7042	3186-AJIEK	Male	0	No	No	66	Yes	N

7043 rows × 22 columns

In [209…

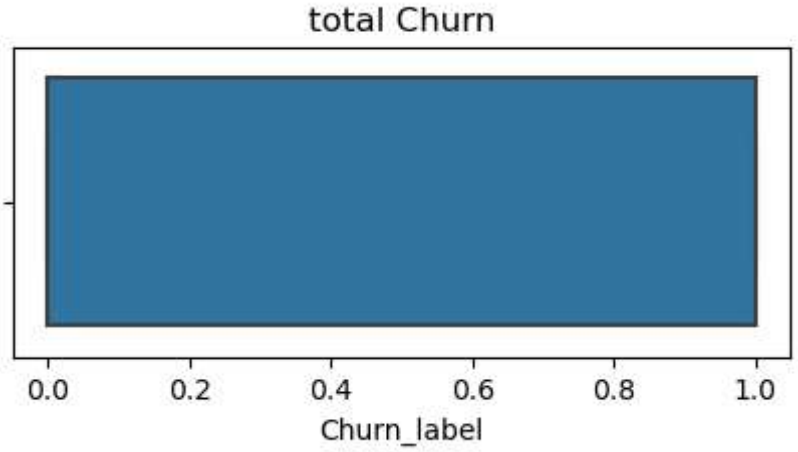
```
Df.dtypes
```

Out[209]:

```
customerID           object
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        float64
Churn                object
Churn_label           int64
dtype: object
```

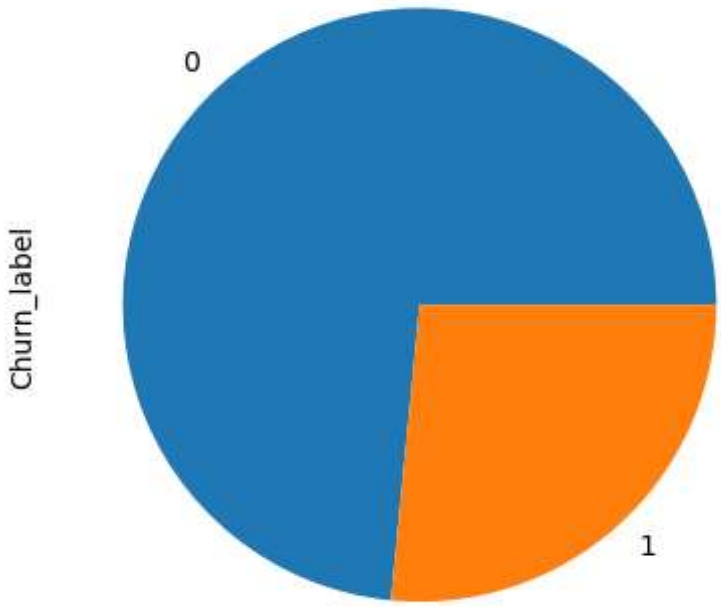
In [211…

```
plt.figure(figsize=(5,2))
sns.boxplot(data=Df,x='Churn_label')
plt.title("total Churn")
plt.show()
## There is no outlier in the target label
```



In [212…

```
Df["Churn_label"].value_counts().plot(kind='pie');
```



In [ ]:

In [215…
```
### drop missing values
Df.dropna(inplace=True)
```

In [216…
```
Df.isnull().sum()
```

Out[216]:
```
customerID          0
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
Churn_label         0
dtype: int64
```

In [ ]:
```
### segement Data
```

In [217…
```
Target_label= Df.pop("Churn_label")
```

In [218…
```
Target_label
```

Out[218]:
```
0       0
1       0
2       1
3       0
4       1
       ..
7038    0
7039    0
7040    0
7041    1
7042    0
Name: Churn_label, Length: 7032, dtype: int64
```

In [222…
```
Df
```

Out[222]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetS
0	Female	0	Yes	No	1	No	No phone service	
1	Male	0	No	No	34	Yes	No	
2	Male	0	No	No	2	Yes	No	
3	Male	0	No	No	45	No	No phone service	
4	Female	0	No	No	2	Yes	No	Fiber
...	...	...	...	...	...	...	...	
7038	Male	0	Yes	Yes	24	Yes	Yes	
7039	Female	0	Yes	Yes	72	Yes	Yes	Fiber
7040	Female	0	Yes	Yes	11	No	No phone service	
7041	Male	1	Yes	No	4	Yes	Yes	Fiber
7042	Male	0	No	No	66	Yes	No	Fiber

7032 rows × 20 columns

In [ ]:
```
```

In [223…
```
#dropping off redundant data
Df.drop("Churn", axis=1,inplace=True)
```

In [224…
```
Df.head(2)
```

Out[224]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetServi
0	Female	0	Yes	No	1	No	No phone service	D
1	Male	0	No	No	34	Yes	No	D

In [ ]:

Encoding categorical variables

In [225…
```
Df.dtypes
```

Out[225]:
```
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       float64
dtype: object
```

In [226…
```
Df1= pd.get_dummies(Df)
Df1
```

Out[226]:

	SeniorCitizen	tenure	MonthlyCharges	TotalCharges	gender_Female	gender_Male	Partner_
0	0	1	29.85	29.85	1	0	
1	0	34	56.95	1889.50	0	1	
2	0	2	53.85	108.15	0	1	
3	0	45	42.30	1840.75	0	1	
4	0	2	70.70	151.65	1	0	
...	...	...	...	...	...	...	
7038	0	24	84.80	1990.50	0	1	
7039	0	72	103.20	7362.90	1	0	
7040	0	11	29.60	346.45	1	0	
7041	1	4	74.40	306.60	0	1	
7042	0	66	105.65	6844.50	0	1	

7032 rows × 45 columns

In [227…

```
Df1.dtypes
```

Out[227]:

```
SeniorCitizen                              int64
tenure                                     int64
MonthlyCharges                           float64
TotalCharges                             float64
gender_Female                              uint8
gender_Male                                uint8
Partner_No                                 uint8
Partner_Yes                                uint8
Dependents_No                              uint8
Dependents_Yes                             uint8
PhoneService_No                            uint8
PhoneService_Yes                           uint8
MultipleLines_No                           uint8
MultipleLines_No phone service             uint8
MultipleLines_Yes                          uint8
InternetService_DSL                        uint8
InternetService_Fiber optic                uint8
InternetService_No                         uint8
OnlineSecurity_No                          uint8
OnlineSecurity_No internet service         uint8
OnlineSecurity_Yes                         uint8
OnlineBackup_No                            uint8
OnlineBackup_No internet service           uint8
OnlineBackup_Yes                           uint8
DeviceProtection_No                        uint8
DeviceProtection_No internet service       uint8
DeviceProtection_Yes                       uint8
TechSupport_No                             uint8
TechSupport_No internet service            uint8
TechSupport_Yes                            uint8
StreamingTV_No                             uint8
StreamingTV_No internet service            uint8
StreamingTV_Yes                            uint8
StreamingMovies_No                         uint8
StreamingMovies_No internet service        uint8
StreamingMovies_Yes                        uint8
Contract_Month-to-month                    uint8
Contract_One year                          uint8
Contract_Two year                          uint8
PaperlessBilling_No                        uint8
PaperlessBilling_Yes                       uint8
PaymentMethod_Bank transfer (automatic)    uint8
PaymentMethod_Credit card (automatic)      uint8
PaymentMethod_Electronic check             uint8
PaymentMethod_Mailed check                 uint8
dtype: object
```

In [228…

```
# Dealing with outliers in Total Charges and normalizing the data set
## only Total charges had outliers but wasnt sure of the rest so i scaled everythir
scaler=MinMaxScaler()
Df1["scaled_tenure"]=scaler.fit_transform(Df["tenure"].values.reshape(-1,1)) ### th
Df1["scaled_MonthlyCharges"]=scaler.fit_transform(Df["MonthlyCharges"].values.resha
Df1["scaled_TotalCharges"]=scaler.fit_transform(Df["TotalCharges"].values.reshape(-

Df1.drop(["tenure","MonthlyCharges","TotalCharges" ],axis=1 , inplace=True)## drop
Df1.head()
```

Out[228]:

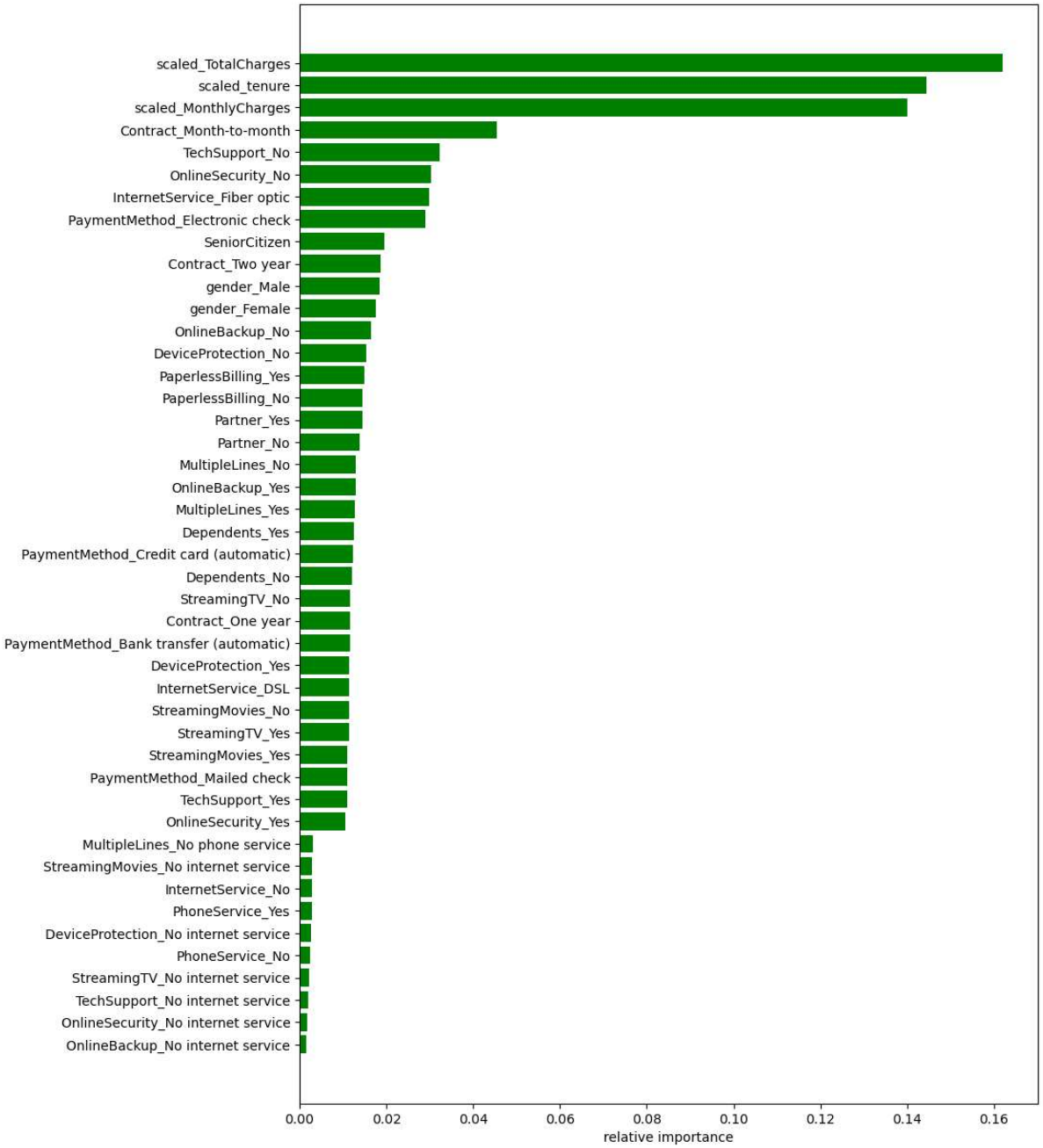
	SeniorCitizen	gender_Female	gender_Male	Partner_No	Partner_Yes	Dependents_No	Depende
0	0	1	0	0	1	1	
1	0	0	1	1	0	1	
2	0	0	1	1	0	1	
3	0	0	1	1	0	1	
4	0	1	0	1	0	1	

5 rows × 45 columns

Model Selection, Training and validation

In [229…

```
## to assign a score to input features based on how useful they are at predicting d


feat_model = RandomForestClassifier()

#fits model
feat_model.fit(Df1,Target_label)
feature_names = list(Df1.columns)
importance = feat_model.feature_importances_
indices = np.argsort(importance)

plt.figure(figsize=(10,15))
plt.barh(range(len(indices)),importance[indices],color='green', align="center")
plt.yticks(range(len(indices)),[feature_names[i] for i in indices])
plt.xlabel("relative importance")
plt.show()
```



In [ ]:
```
### from this we can see that Total charges is the most colset to the target label
```

In [230…
```
# split the dataset into training and testing sets  x=Questions, y=Answers
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(Df1,Target_label,test_size=0.2,rando
```

In [231…
```
Df1.shape
```

Out[231]:
```
(7032, 45)
```

In [233…
```
Target_label.shape
```

Out[233]:
```
(7032,)
```

In [235…
```
x_train.shape,x_test.shape
```

Out[235]:
```
((5625, 45), (1407, 45))
```

In [270…

Out[270]:
```
2481    0
6784    0
6125    1
3052    0
4099    0
       ..
1733    0
5250    0
5465    0
5851    0
3984    0
Name: Churn_label, Length: 1407, dtype: int64
```

In [236…
```
y_train.shape,y_test.shape
```

Out[236]:
```
((5625,), (1407,))
```

In [277…
```
### Building Model
#importing the predictive models
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
```

In [240…
```
x_train
```

Out[240]:

	SeniorCitizen	gender_Female	gender_Male	Partner_No	Partner_Yes	Dependents_No	Depe
6030	0	1	0	1	0	1	
3410	0	0	1	1	0	1	
5483	0	1	0	0	1	1	
5524	0	0	1	0	1	0	
6337	0	1	0	0	1	0	
...	...	...	...	...	...	...	
3778	0	0	1	1	0	1	
5199	0	1	0	1	0	1	
5235	0	0	1	1	0	1	
5399	0	1	0	1	0	1	
862	1	0	1	0	1	1	

5625 rows × 45 columns

In [241…
```
y_train
```

Out[241]:
```
6030    1
3410    1
5483    1
5524    0
6337    0
       ..
3778    0
5199    0
5235    0
5399    0
862     1
Name: Churn_label, Length: 5625, dtype: int64
```

In [246…

In [ ]:

In [ ]:

In [ ]:

In [250…

In [ ]:

In [251…

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [255…
```
# another model buliding
# RandomForest for Classification
# instantiating the model
RFC=RandomForestClassifier()
```

In [256…
```
## Training the model
RFC.fit(x_train,y_train)
```

Out[256]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [257…
```
## creating a prediction file for the model
RFC_pred= RFC.predict(x_test)
```

In [258…
```
RFC_pred
```

Out[258]:
```
array([0, 0, 1, ..., 0, 0, 0], dtype=int64)
```

In [ ]:
```
### Model Evaluation and interpretation
```

In [259…
```
#importing evaluation metrics
from sklearn.metrics import classification_report, confusion_matrix
```

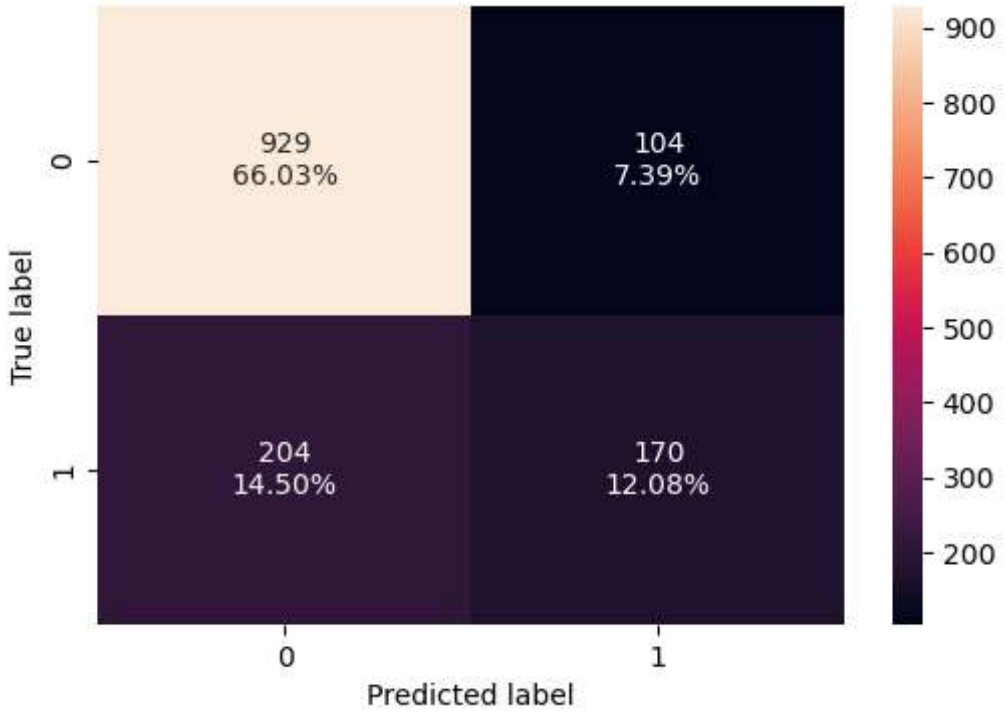
In [262…
```
def confusion_matrix_sklearn(model, predictors, target):
    y_pred = model.predict(predictors)
    cm = confusion_matrix(target, y_pred)
    labels = np.asarray(
        [
            ["{0:0.0f}".format(item) + "\n{0:.2%}".format(item / cm.flatten().sum()
            for item in cm.flatten()
        ]
    ).reshape(2, 2)

    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=labels, fmt="")
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

In [273…
```
## classification report
print("classification report for the Random Forest Classifier model \n",classificat
```

```
classification report for the Random Forest Classifier model
              precision    recall  f1-score   support

           0       0.82      0.90      0.86      1033
           1       0.62      0.45      0.52       374

    accuracy                           0.78      1407
   macro avg       0.72      0.68      0.69      1407
weighted avg       0.77      0.78      0.77      1407
```

In [ ]:
```
```

In [265…
```
confusion_matrix_sklearn(RFC,x_test,y_test)
```



In [278…
```
# another model buliding
# Decision Tree Classifier
```

```
# instantiating the model
DTC=DecisionTreeClassifier()
```

In [279…
```
## Training the model
DTC.fit(x_train,y_train)
```

Out[279]:
```
▾ DecisionTreeClassifier

DecisionTreeClassifier()
```

In [283…
```
## creating a prediction file for the model
DTC_pred= DTC.predict(x_test)
```

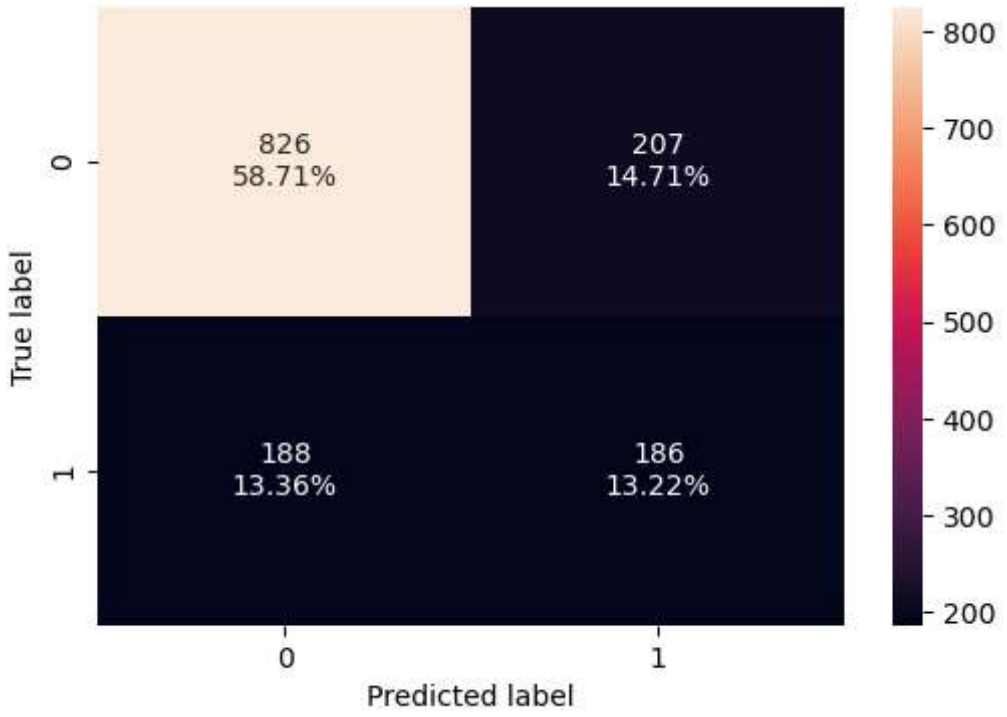
In [284…
```
DTC_pred
```

Out[284]:
```
array([0, 0, 1, ..., 0, 0, 0], dtype=int64)
```

In [285…
```
## classification report
print("classification report for the Random Forest Classifier model \n",classificat
```

```
classification report for the Random Forest Classifier model
              precision    recall  f1-score   support

           0       0.81      0.80      0.81      1033
           1       0.47      0.50      0.49       374

    accuracy                           0.72      1407
   macro avg       0.64      0.65      0.65      1407
weighted avg       0.72      0.72      0.72      1407
```

In [287…
```
confusion_matrix_sklearn(DTC,x_test,y_test)
```



In [290…
```
# another model buliding
# Logistic Regression
# instantiating the model
LR=LogisticRegression()
```

In [292…
```
#trainning the model
LR.fit(x_train,y_train)
```

```
C:\Users\awura\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[292]:
```
▾ LogisticRegression

LogisticRegression()
```

In [295…
```
#creating a prediction file for the model
LR_pred = LR.predict(x_test)
```

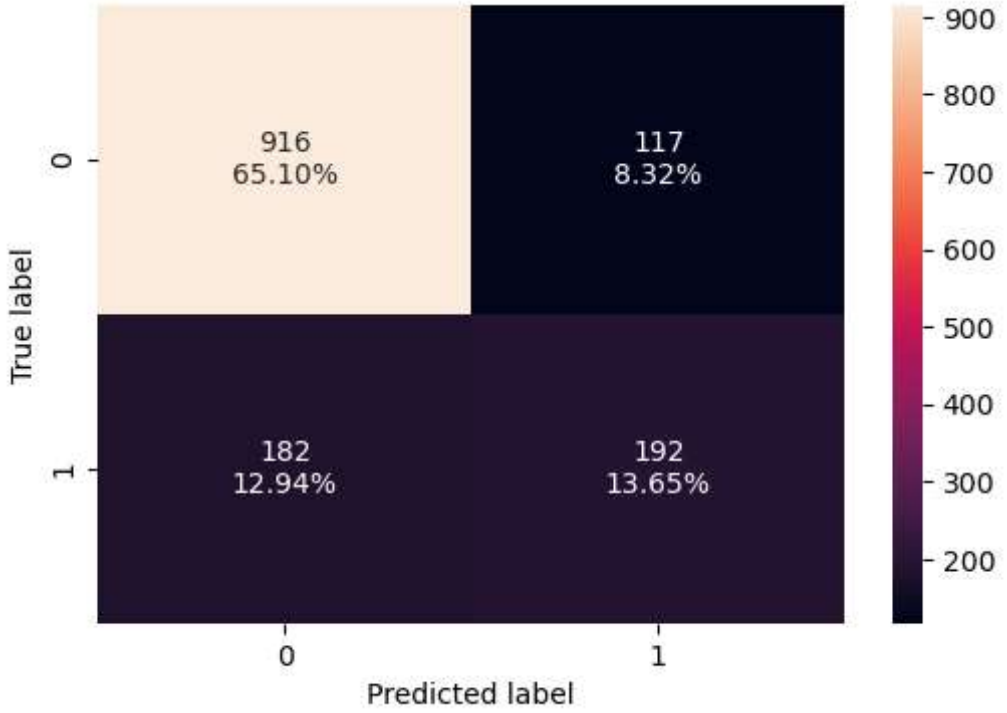
In [296…
```
LR_pred
```

Out[296]:
```
array([0, 0, 1, ..., 0, 0, 0], dtype=int64)
```

In [297…
```
#classification_report
print("classification report for the logistic regression model \n",classification_r
```

```
classification report for the logistic regression model
               precision    recall  f1-score   support

           0       0.83      0.89      0.86      1033
           1       0.62      0.51      0.56       374

    accuracy                           0.79      1407
   macro avg       0.73      0.70      0.71      1407
weighted avg       0.78      0.79      0.78      1407
```

In [298…
```
confusion_matrix_sklearn(LR,x_test,y_test)
```



In [ ]:
```
"""from the three supervised machine learning model trained and test on the target
the highest percentage of true positives (ie Customers Retained) of 66.03% and leas
It predicted more of type II error (Customers that are still with the company but p
This model performance can be improved futher.
```

In [ ]:

In [ ]:

In [ ]:

In [288…

In [304…

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[304], line 31
     25 # shuffle the parameter combinations
     26 #random.shuffle(params)
     27
     28 #find the best parameter combination
     29 #score = make_scorer(r2_score)
     30 gridsearch = GridSearchCV(opt_model,best_params,scoring='accuracy',cv=3,re
turn_train_score=True,refit=True)
---> 31 gridsearch.fit(X_train,y_train)
     32 print("Best params combination:",gridsearch.best_params_,"\n")

NameError: name 'X_train' is not defined
```

In [ ]:

In [ ]: