Customer Churn Prediction report - 12/7/2023

Project title: Predictive Analytics for customer churn in telecom industry

Problem description: The telecommunications sector has encountered a notable obstacle in recent times, characterized by a pronounced issue known as customer churn. Customer churn, which involves the departure of customers to rival firms or the cessation of services, poses a considerable financial burden on telecommunications companies.

Objective: Our objective is to address the challenge of accurately forecasting and alleviating customer churn within this industry by identifying customers at risk of churn and taking proactive measures like offers and discounts can help reduce revenue loss and improve customer retention.

Data Set source: Kaggle

Step 1: Data exploration

- Imported the libraries necessary including pandas, numpy, matplotlib, seaborn and SciKitlearn as seen in Figure 1 below

```
In [ ]:  #Imported the libraries necessary
         import numpy as np
         import pandas as pd
         import matplotli.pyplot as plt
         import seaborn as sns
         from sklearn.model_selection import train_test_split

In [ ]:
```

Figure 1

- I loaded the data set using the pandas library and read the csv file
- I checked the head of the read ad_data using .head() and the data set has 5 rows and 21 Columns as seen in Figure 2 below

```
#Loading the data
telco_data = pd.read_csv('Customer_Churn.csv')
```

```
# Display head of data
telco_data.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLine
0	7590-VHVEG	Female	0	Yes	No	1	No	No phon servic
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 phon servic
4	9237-HQITU	Female	0	No	No	2	Yes	N

5 rows × 21 columns

Figure 2

- I described the data using info() to view the count, mean, std and min
- I checked for the shape of the data using .shape

```
telco_data.shape
```

```
(7043, 21)
```

```
telco_data.describe()
```

	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7033.000000
mean	0.162147	32.371149	64.759434
std	0.368612	24.559481	30.096565
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

Figure 3

Step 2: Data Preprocessing

- I got the sum of all the missing values using isnull() and sum()
- I established the total of missing values in 'TotalCharges' is 10 entries as seen in the Figure 4 below

```
# sum of all the missing values using isnull() and sum()
telco_data.isnull().sum()
```

```
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      10
TotalCharges         0
Churn                0
dtype: int64
```

Figure 4

- I calculated the mean of 'MonthlyCharges' using mean()
- I chose to round it off to 1 decimal place to look like the rest of the age entries
- Then I filled the missing values in the 'Age' column with the rounded mean as seen in Figure 5 below

```
#calculated the mean of 'MonthlyCharges' using mean() and round()
mean_charges=round(telco_data['MonthlyCharges'].mean(),1)
```

```
mean_charges
```

```
]: 64.8
```

```
#Fill in the missing entries with the calculated mean
telco_data['MonthlyCharges'].fillna(mean_charges, inplace=True)
```

Figure 5

- After I double checked to confirm that all missing values are filled using isnull() and sum() and the data frame returned no missing values in any columns as seen in Figure 6 below

```
#After I double checked to confirm that all missing values
telco_data.isnull().sum()
```

```
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
dtype: int64
```

Figure 6

Step 3: Data Visualization

- Created an interactive Histogram comparing count relationship between the gender and Churn Using plotly.express
- A total for 939 females are likely to churn while 2549 are not like to Churn
- A total of 930 males are like;y to churn while 2625 females are not likely to churn

```
#Created an interactive Histogram comparing the count relationship between th
telco_hist=px.histogram(telco_data, x='gender', color='Churn', marginal='box'
telco_hist.update_layout(bargap=0.2)
```

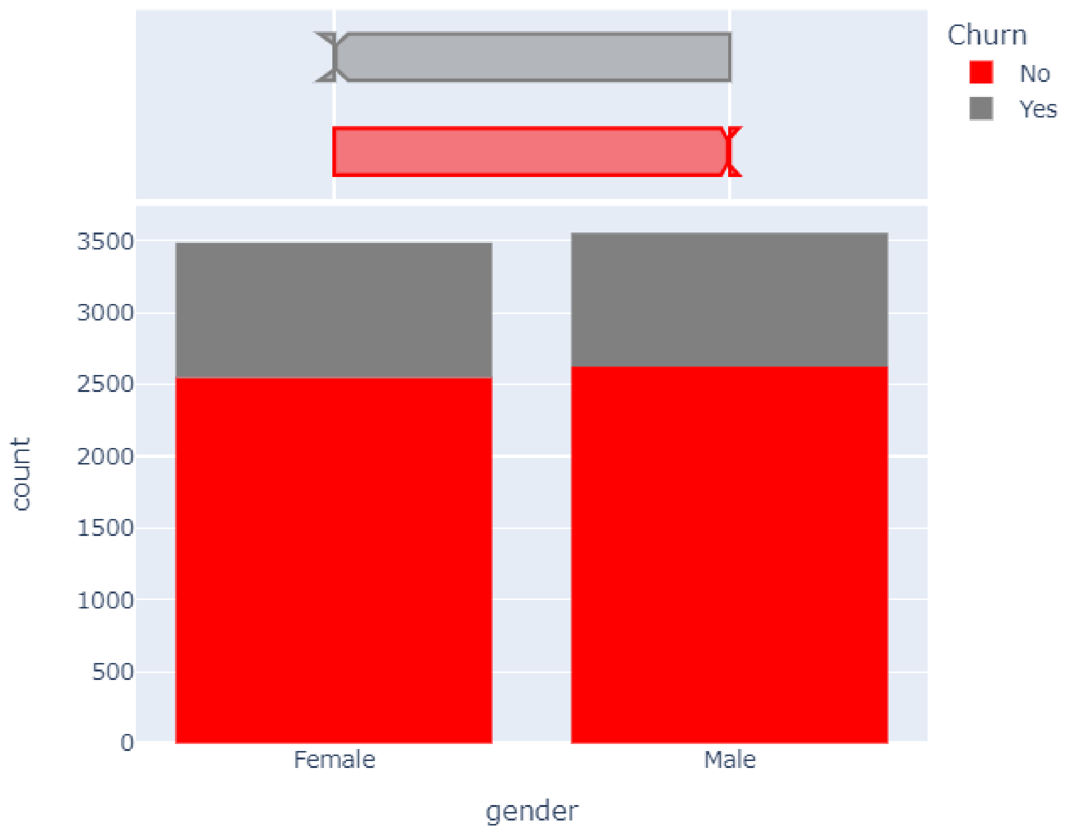

Figure 7

- I Created a bar showing showing relationship between ‘gender’ and ‘Churn’ as seen in Figure 8 below

```
#I Created a bar showing showing relationship between ‘gender’ and ‘Churn’
plt.bar(telco_data['gender'],telco_data['Churn'])
```

```
<BarContainer object of 7043 artists>
```

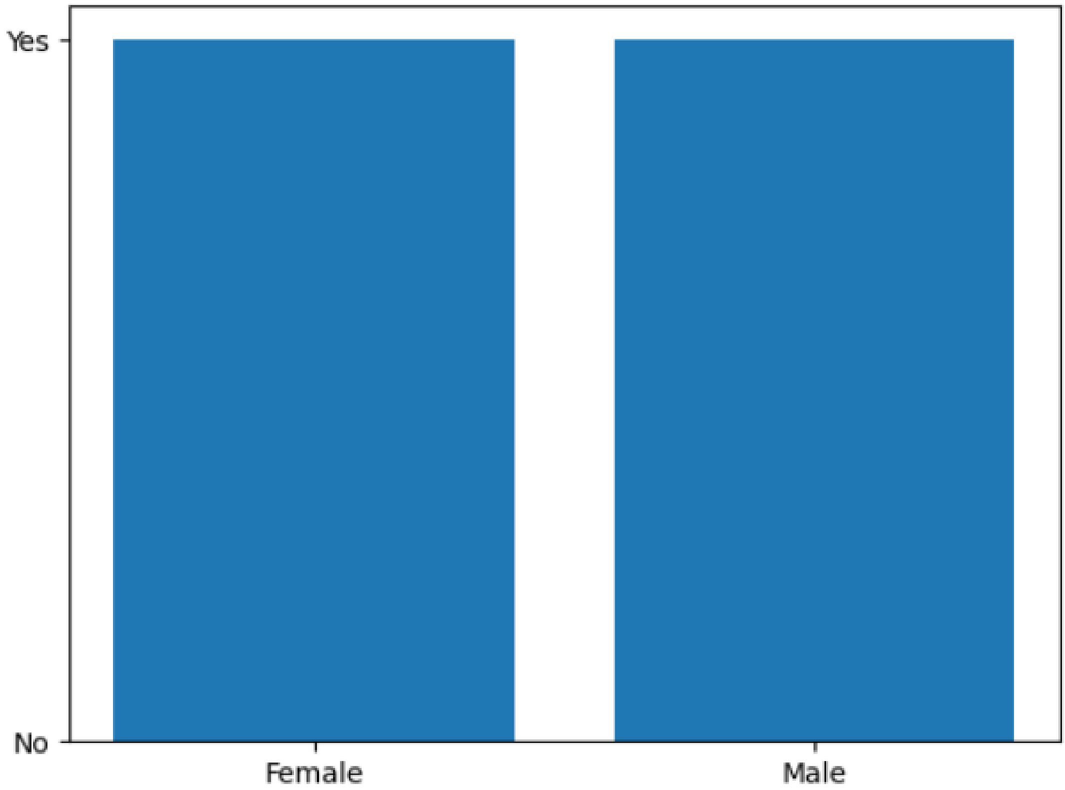

Figure 8

- Created histogram sub plots for ‘SenoirCitizen’, ‘tenure’, ‘MonthlyCharges’ as seen in the below Figure 9

```
telco_data.hist(bins=30, figsize=(20,15))
```

```
array([[<AxesSubplot:title={'center':'SeniorCitizen'}>,
        <AxesSubplot:title={'center':'tenure'}>],
       [<AxesSubplot:title={'center':'MonthlyCharges'}>, <AxesSubplot:>]],
      dtype=object)
```

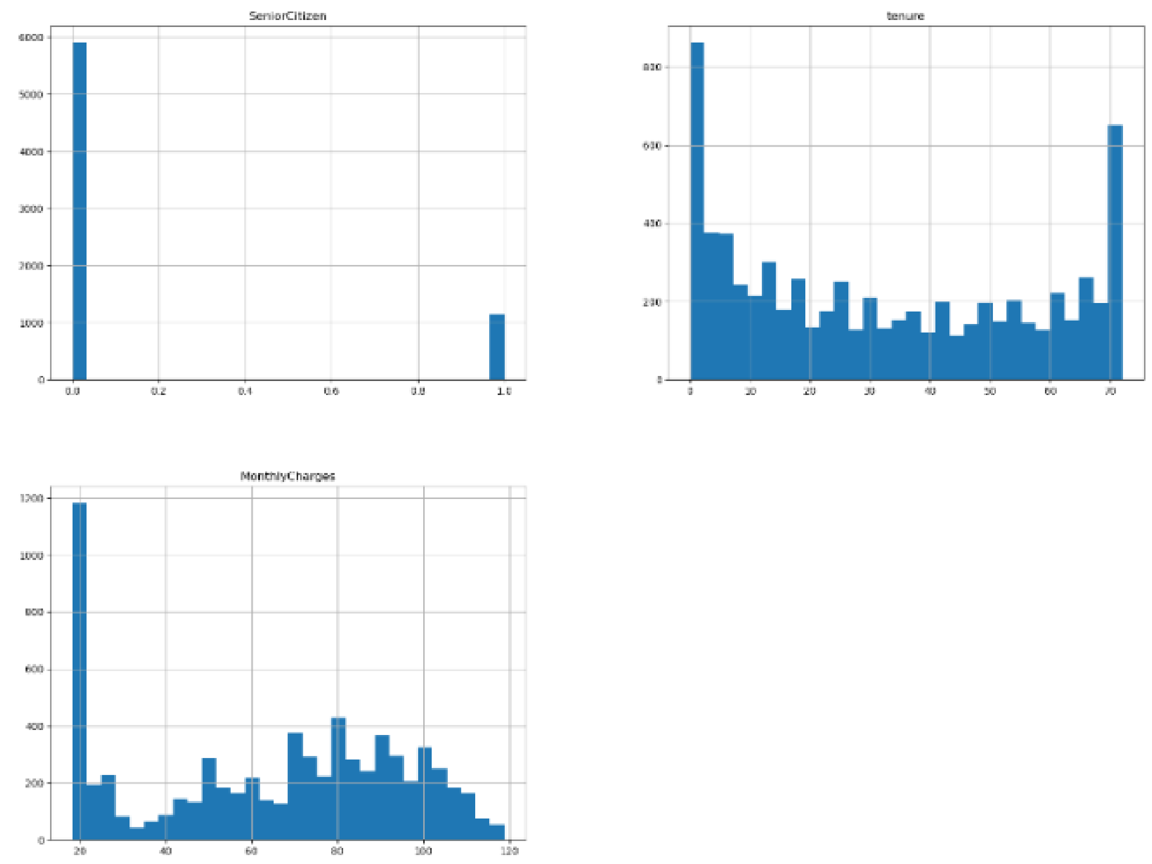

Figure 9

- Finally, I created a pairplot with the hue defined by the ‘Clicked on Ad column feature
- This finds a relationship between all the numerical columns and ‘Churn’ as seen in the Figure 10 below

```
#I created a  pairplot with the hue defined by the ‘Churn' column feature
sns.pairplot(telco_data,hue='Churn')
```

```
<seaborn.axisgrid.PairGrid at 0x1a7e13ea1f0>
```

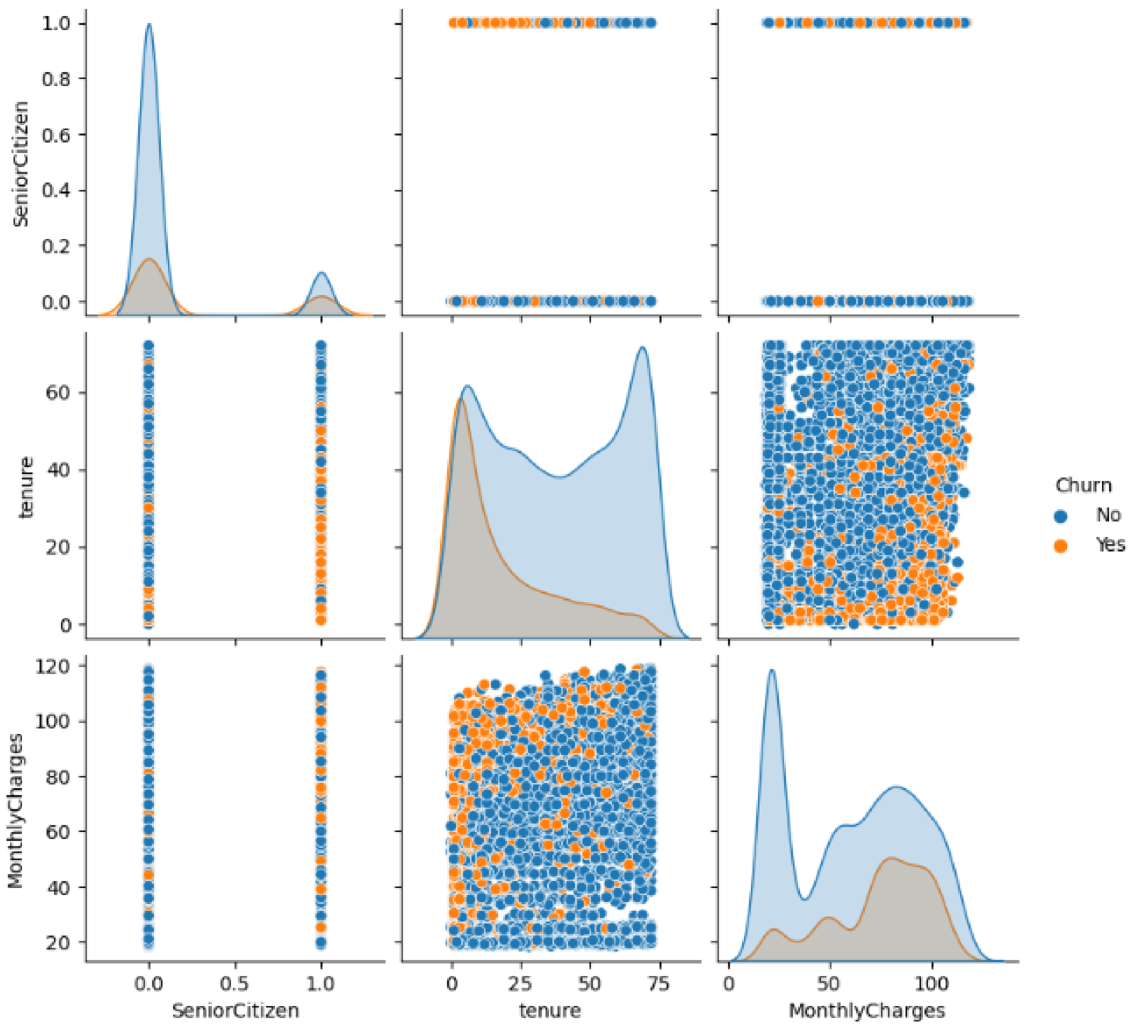

Figure 10

Step 4: Data Cleaning using drop()

- By removing 'gender', 'customerID', and 'tenure' as they are not useful in our analysis
- I showed a pairplot excluding the removed columns above as shown in Figure 11 below

```
#Data Cleaning By removing 'gender', 'customerID', and 'tenure'

col =['gender','customerID','tenure']
telco_data = telco_data.drop(col,axis=1)
```

```
sns.pairplot(telco_data)
```

```
<seaborn.axisgrid.PairGrid at 0x1a7e1050580>
```

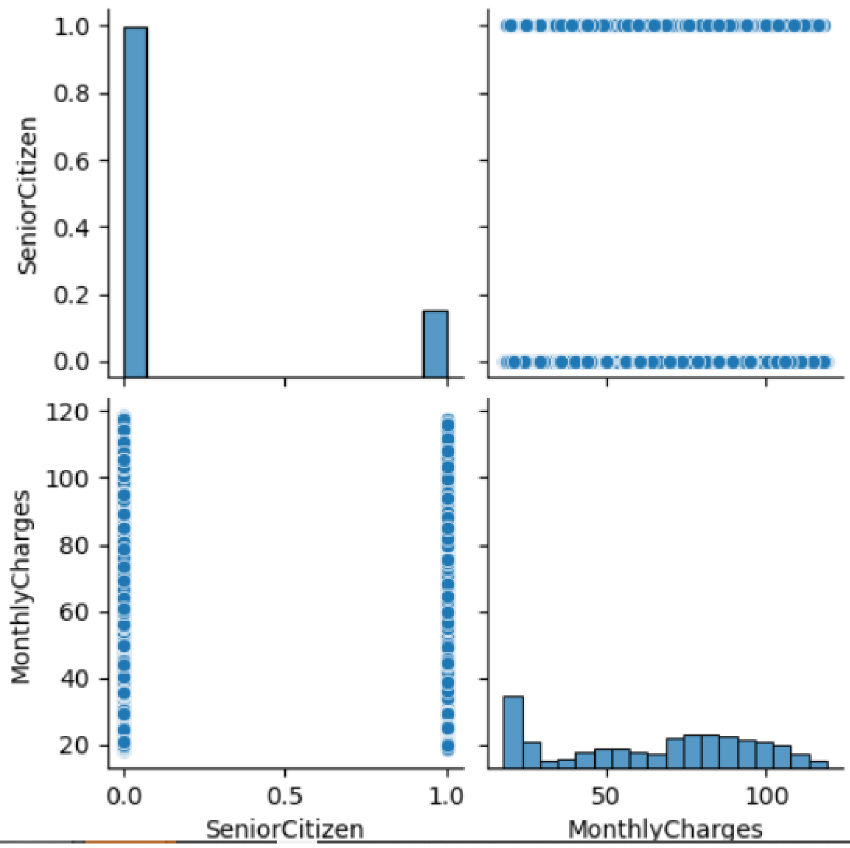

Figure 11

Step5: Data type conversions

- Describe 'TotalCharges', we have to convert it from object
- Describe 'MonthlyCharges' as seen in figure 12 below

```
telco_data['TotalCharges'].describe()
```

```
count      7043
unique     6531
top
freq         11
Name: TotalCharges, dtype: object
```

```
telco_data['MonthlyCharges'].describe()
```

```
count    7043.000000
mean       64.759492
std        30.075189
min        18.250000
25%        35.525000
50%        70.300000
75%        89.850000
max       118.750000
Name: MonthlyCharges, dtype: float64
```

Figure 12

- ‘TotalCharges’ contains a string(“ “) at 488 position, I have to remove/ replace it using np.nan
- Coerce will replace all the non-numeric values
- Then I will drop all the rows in which there is any null values
- We can now describe ‘TotalCharges ’and confirm that it's now a float as seen in figure 13 below

```
#‘TotalCharges’ contains a  string(“ “) at 488 position, I have to remove/ re
telco_data['TotalCharges']=telco_data['TotalCharges'].replace(" ",np.nan)
#coerce will replace all the non-numeric values
telco_data['TotalCharges']=pd.to_numeric(telco_data['TotalCharges'],errors='c
```

```
#dropping all the rows in which there is a null values
# Removing all the rows which have null value in it
telco_data = telco_data.dropna(how='any', axis =0)
```

```
telco_data['TotalCharges'].describe()
```

```
count    7032.000000
mean     2283.300441
std      2266.771362
min        18.800000
25%       401.450000
50%      1397.475000
75%      3794.737500
max      8684.800000
Name: TotalCharges, dtype: float64
```

Figure 13

- Describe column ‘Churn’ and It has only 2 unique values as seen in Figure 14 below

```
telco_data['Churn'].describe()
```

```
count      7032
unique        2
top          No
freq       5163
Name: Churn, dtype: object
```

Figure 14

- I visualized the data comparing Churn to TotalCharges and Monthly Charges as seen in the below Figure 15(a) Figure 15(b) Figure (15c) Figure (15d)

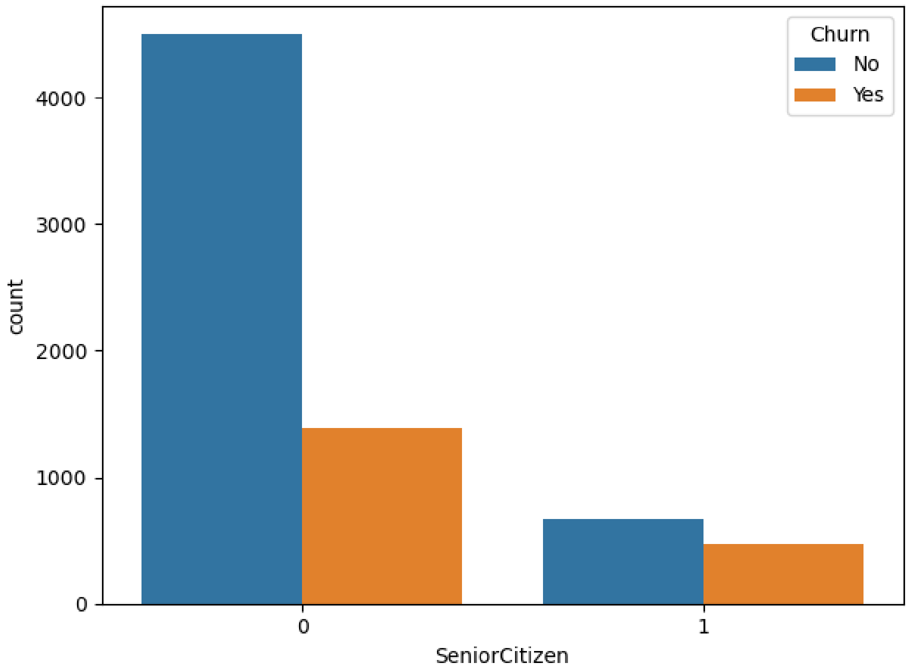

Figure 15(a)

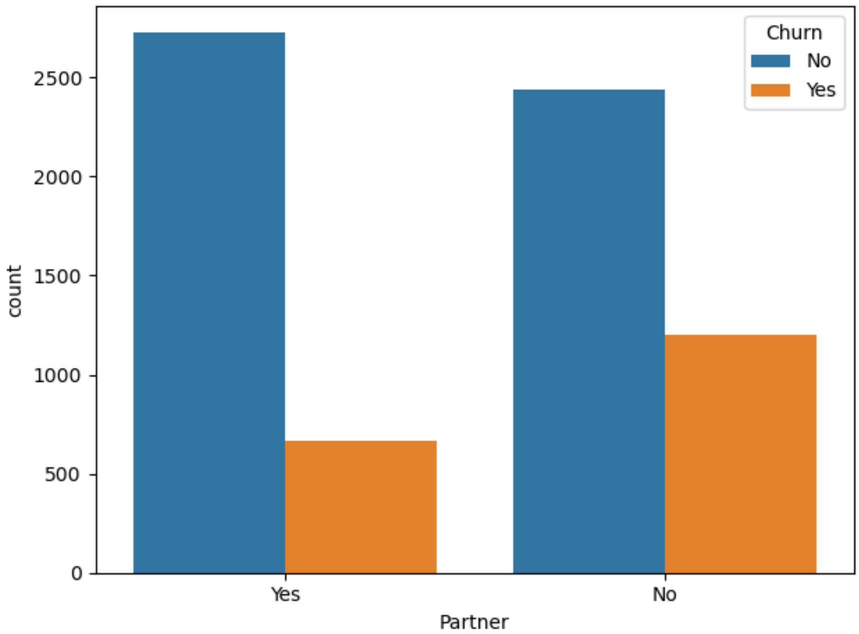

Figure 15(b)

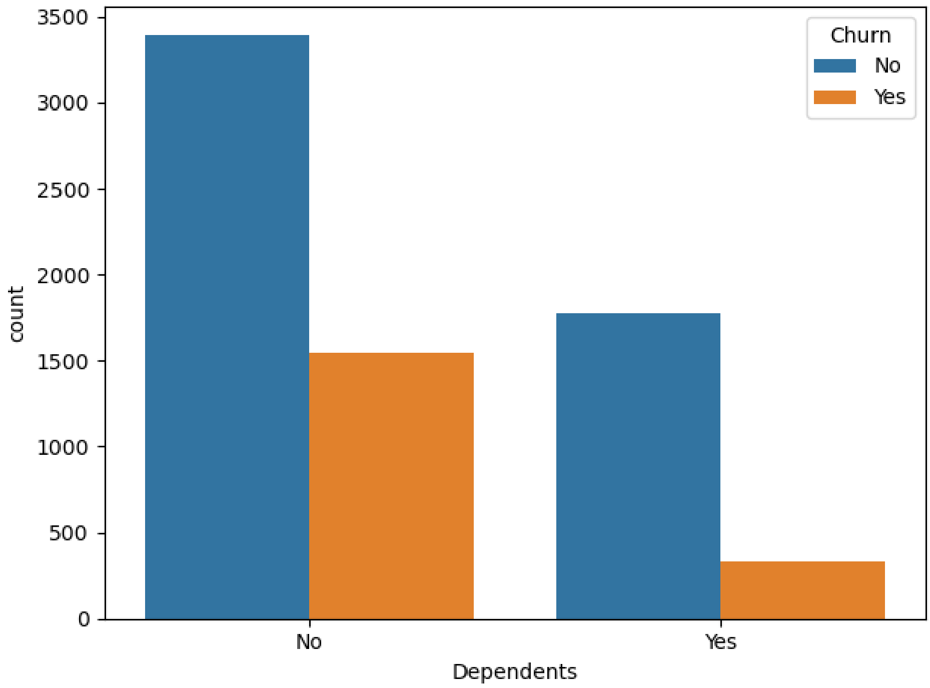

Figure (15c)

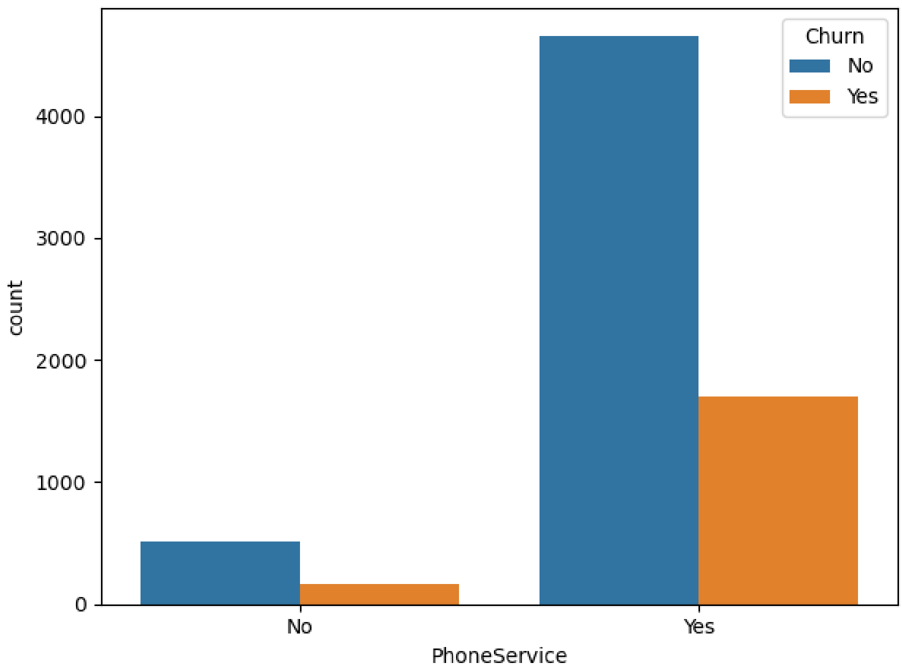

Figure (15d)

Step6: Dealing with categorical data

- Converting Yes as 1 and No as 0
- Created the dummy variables as seen in the image below
- This helps the machine learning algorithm to understand it and be able to use it i the prediction as seen in Figure 16 below

```
telco_data_dummies = pd.get_dummies(telco_data)
print(telco_data_dummies)
```

```
      SeniorCitizen  MonthlyCharges  TotalCharges  Churn  Partner_No  \
0                 0           29.85         29.85      0           0
1                 0           56.95       1889.50      0           1
2                 0           53.85        108.15      1           1
3                 0           64.80       1840.75      0           1
4                 0           70.70        151.65      1           1
...             ...             ...           ...    ...         ...
7038              0           84.80       1990.50      0           0
7039              0          103.20       7362.90      0           0
7040              0           29.60        346.45      0           0
7041              1           64.80        306.60      1           0
7042              0          105.65       6844.50      0           1

      Partner_Yes  Dependents_No  Dependents_Yes  PhoneService_No  \
0               1              1               0                1
1               0              1               0                0
2               0              1               0                0
3               0              1               0                1
4               0              1               0                0
...           ...            ...             ...              ...
7038            1              0               1                0
7039            1              0               1                0
7040            1              0               1                1
7041            1              1               0                0
7042            0              1               0                0

      PhoneService_Yes  ...  StreamingMovies_Yes  Contract_Month-to-month
\
0                    0  ...                    0                        1
1                    1  ...                    0                        0
2                    1  ...                    0                        1
```

Figure 16

- Now we are changing the correlation between all the data using corr_matrix
- High Churn is seen in case of monthly contracts, no online security, no technical support, first year subscription and fiber optic internet
- Low churn is in case of long contracts, subscriptions without internet service and customers with contracts for more than 5 years
- Factors such as gender, phone service availability, and multiple lines almost have no impact on Churn as seen in figure 17 below

```
# Now we are changing the correlation between all the data
churn_corr_matrix= telco_data_dummies.corr()
```

```
matrix['Churn'].sort_values(ascending=False).plot(kind='bar', figsize=(15,10))
```

<AxesSubplot:>

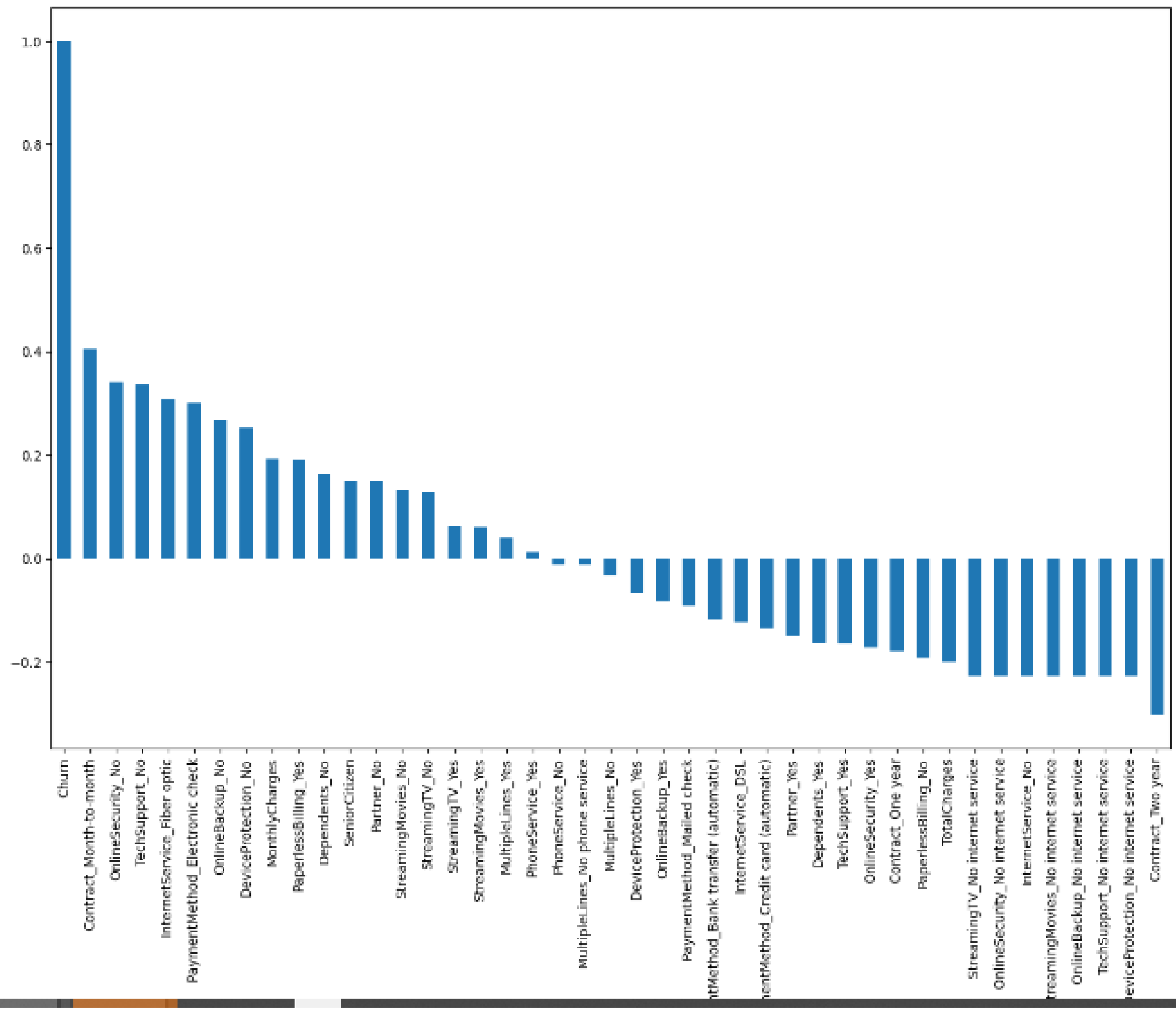

Figure 17

- Checking the Correlation between all the columns as seen in figure 18 below

```
In [60]:  #Checking the Correlation between all the columns
          churn_corr_matrix['Churn'].sort_values(ascending = False)
```

```
Out[60]: Churn                                      1.000000
         Contract_Month-to-month                    0.404565
         OnlineSecurity_No                          0.342235
         TechSupport_No                             0.336877
         InternetService_Fiber optic                0.307463
         PaymentMethod_Electronic check             0.301455
         OnlineBackup_No                            0.267595
         DeviceProtection_No                        0.252056
         MonthlyCharges                             0.192848
         PaperlessBilling_Yes                       0.191454
         Dependents_No                              0.163128
         SeniorCitizen                              0.150541
         Partner_No                                 0.149982
         StreamingMovies_No                         0.130920
         StreamingTV_No                             0.128435
         StreamingTV_Yes                            0.063254
         StreamingMovies_Yes                        0.060860
         MultipleLines_Yes                          0.040033
         PhoneService_Yes                           0.011691
         PhoneService_No                            0.011691
```

Figure 18

- Drop Churn where axis = 1
- Create a variable X as seen in the figure 19 below

```
#Drop Churn where axis =1
X=telco_data_dummies.drop('Churn',axis=1)
```

```
X
```

	SeniorCitizen	MonthlyCharges	TotalCharges	Partner_No	Partner_Yes	Dependents_No
0	0	29.85	29.85	0	1	1
1	0	56.95	1889.50	1	0	1
2	0	53.85	108.15	1	0	1
3	0	64.80	1840.75	1	0	1
4	0	70.70	151.65	1	0	1
...	...	...	...	...	...	...
7038	0	84.80	1990.50	0	1	0
7039	0	103.20	7362.90	0	1	0
7040	0	29.60	346.45	0	1	0
7041	1	64.80	306.60	0	1	1
7042	0	105.65	6844.50	1	0	1

7032 rows × 42 columns

Figure 19

- Create y variable as seen in figure 20 below

```
#create y variable
y=telco_data_dummies['Churn']
```

```
y
```

```
0       0
1       0
2       1
3       0
4       1
       ..
7038    0
7039    0
7040    0
7041    1
7042    0
Name: Churn, Length: 7032, dtype: int64
```

Figure 20

Step 7: Variable Imbalance

- Since y consists of data if the customer is going to Churn or not(as 0, 1)
- I considered Chur values in y using value_counts() and notice 5163 for 0s and 1869 for 1s which is a variable imbalance which will lead to wrong results as seen in the figure 21 below

```
y.shape
```

```
(7032,)
```

```
y.value_counts()
```

```
0    5163
1    1869
Name: Churn, dtype: int64
```

Figure 21

- I considered using SMOTE for imbalance Classification
- I installed a module Imbalanced-learn and balanced the entries in the X and y variables so now the total rows is 1036 and 42 columns as seen in the figure 22 below

```
from imblearn.over_sampling import SMOTE
```

```
smote=SMOTE(random_state=0)
```

```
X_resampled_smote, y_resampled_smote=smote.fit_resample(X,y)
```

```
y_resampled_smote.value_counts()
```

```
0    5163
1    5163
Name: Churn, dtype: int64
```

```
X_resampled_smote
```

	SeniorCitizen	MonthlyCharges	TotalCharges	Partner_No	Partner_Yes	Dependents_No
0	0	29.850000	29.850000	0	1	1
1	0	56.950000	1889.500000	1	0	1
2	0	53.850000	108.150000	1	0	1
3	0	64.800000	1840.750000	1	0	1
4	0	70.700000	151.650000	1	0	1
...	...	...	...	...	...	...
10321	0	103.976753	242.804921	0	1	1
10322	0	35.824447	35.824447	1	0	1
10323	0	44.493077	1061.960339	0	0	0
10324	0	19.363055	19.363055	1	0	1
10325	0	96.922890	96.922890	1	0	1

10326 rows × 42 columns

Figure 22

Step 8: Logistic Regression Model training

- I trained the model using the balanced data
- I divided into into train data and test data as seen in the figure 23 below

```
from sklearn.linear_model import LogisticRegression
```

```
test_split(X_resampled_smote,y_resampled_smote,test_size=0.2,random_state=42)
```

```
LogReg=LogisticRegression()
```

```
LogReg.fit(X_smote_train,y_smote_train)
```

Figure 28

Step 9: Making the Churn prediction as seen in the figure 29 below

```
y_smote_pred=LogReg.predict(X_smote_test)
```

```
y_smote_pred
```

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

Figure 29

Step 10: Model Evaluation

- I classification_report, confusion_matrix from sklearn
- I created a confusion matrix and a classification report with:
- Precision of 83%
- Recall of 83%
- F1 Score of 83% as seen in Figure 30 below

```
#Create Classification report and Confusion matrix
from sklearn.metrics import classification_report, confusion_matrix
```

```
# Classification Report
classification_rep = classification_report(y_smote_test, y_smote_pred)
print("Classification Report:\n", classification_rep)
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.82      0.84      0.83      1037
           1       0.84      0.81      0.82      1029

    accuracy                           0.83      2066
   macro avg       0.83      0.83      0.83      2066
weighted avg       0.83      0.83      0.83      2066
```

```
# Confusion Matrix
conf_matrix = confusion_matrix(y_smote_test, y_smote_pred)
print("Confusion Matrix:\n", conf_matrix)
```

```
Confusion Matrix:
 [[872 165]
 [191 838]]
```

Figure 30

Conclusion

- From the analysis of customer churn, 83% were actual churn cases. It measures the accuracy of the positive predictions.
- An 83% recall indicates that the model correctly identified 83% of the actual customer churn cases. It measures the ability of the model to capture all the positive instances.
- The F1 Score is the weighted average of Precision and Recall. It considers both false positives and false negatives.
- An F1 Score of 83% reflects a balance between precision and recall, providing a single metric that considers both false positives and false negatives. It is particularly useful when there is an uneven class distribution.