The percentage of customers who ceased using a company's product or service over a specific period of time is known as customer churn. Given that it is significantly less costly to keep current customers than it is to acquire new ones, customer churn is one of the most crucial metrics for a growing company to monitor. Customers in the telecom sector have a variety of service providers to choose from and can actively move between them. In this fiercely competitive market, the telecommunications industry experiences a yearly churn rate of 15 to 25 percent.

In [1]:
```
#Telco Churn Analysis
```

In [2]:
```
#import the required libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
%matplotlib inline
```

In [3]:
```
telco_base_data = pd.read_csv('Telco_Customer_Churn.csv')
```

In [4]:
```
telco_base_data.head()
```

Out[4]:

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

5 rows × 21 columns

In [5]:
```
telco_base_data.shape
```

Out[5]:
```
(7043, 21)
```

In [6]:
```
telco_base_data.columns.values
```

Out[6]:
```
array(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
       'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
       'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges',
       'TotalCharges', 'Churn'], dtype=object)
```

In [7]:
```
# Checking the data types of all the columns
telco_base_data.dtypes
```

Out[7]:

```
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 [8]:

```
# Check the descriptive statistics of numeric variables
telco_base_data.describe()
```

Out[8]:

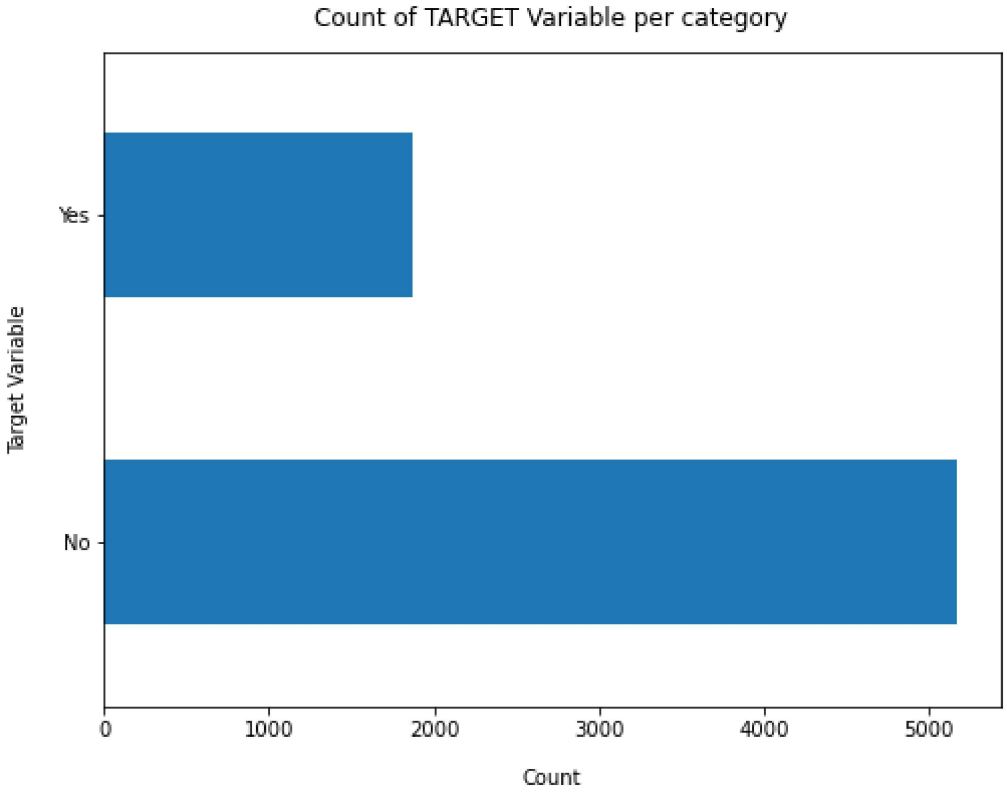
	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

In [9]:
```
#SeniorCitizen is actually a categorical hence the 25%-50%-75% distribution is not propoer

#75% customers have tenure less than 55 months

#Average Monthly charges are USD 64.76 whereas 25% customers pay more than USD 89.85 per month
```

In [10]:
```
telco_base_data['Churn'].value_counts().plot(kind='barh', figsize=(8, 6))
plt.xlabel("Count", labelpad=14)
plt.ylabel("Target Variable", labelpad=14)
plt.title("Count of TARGET Variable per category", y=1.02);
```



In [11]:
```
100*telco_base_data['Churn'].value_counts()/len(telco_base_data['Churn'])
```

Out[11]:
```
No     73.463013
Yes    26.536987
Name: Churn, dtype: float64
```

In [12]:
```
telco_base_data['Churn'].value_counts()
```

Out[12]:
```
No     5174
Yes    1869
Name: Churn, dtype: int64
```

In [13]:
```
#Data is highly imbalanced, ratio = 73:27
#So we analyse the data with other features while taking the target values separately to get some insights.
```

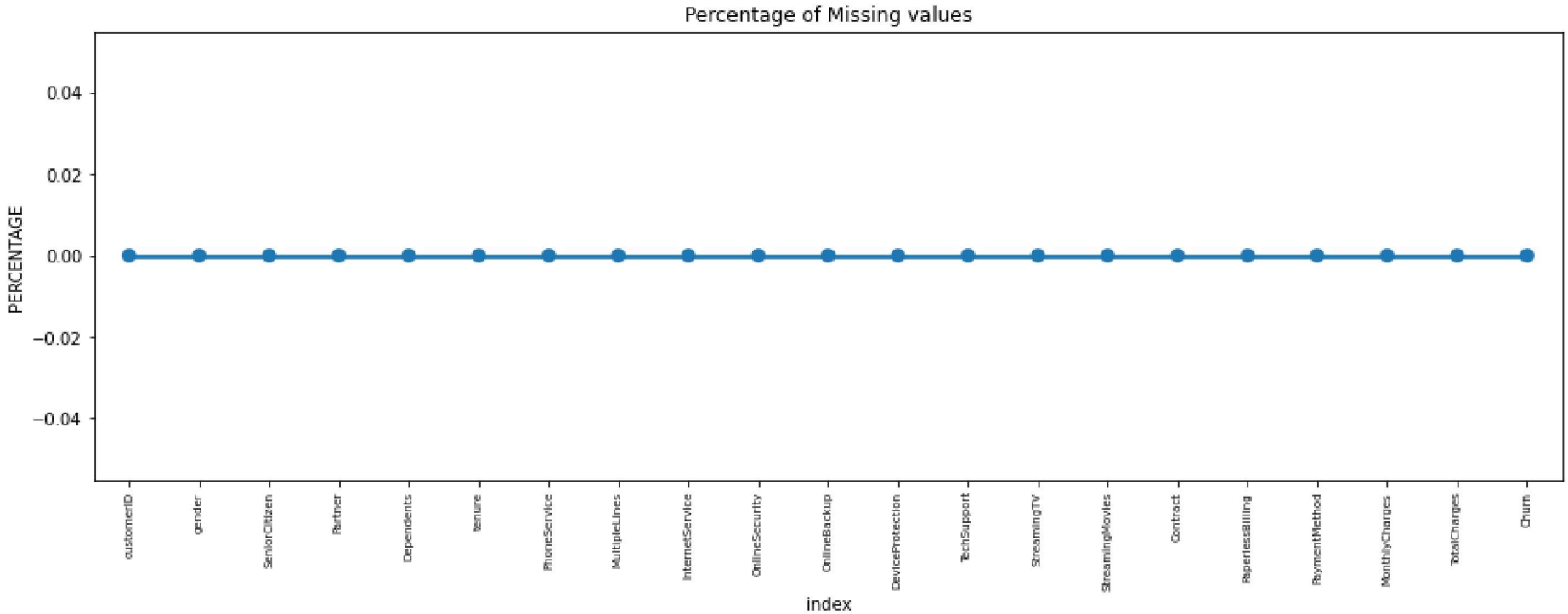
In [14]:
```
# Concise Summary of the dataframe, as we have too many columns, we are using the verbose = True mode
telco_base_data.info(verbose = True)
```

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

In [15]:
```
missing = pd.DataFrame((telco_base_data.isnull().sum())*100/telco_base_data.shape[0]).reset_index()
plt.figure(figsize=(16,5))
```

```
ax = sns.pointplot('index',0,data=missing)
plt.xticks(rotation =90,fontsize =7)
plt.title("Percentage of Missing values")
plt.ylabel("PERCENTAGE")
plt.show()
```

```
C:\Users\ACER\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword arg
s: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit k
eyword will result in an error or misinterpretation.
  warnings.warn(
```



In [16]:
```
#Missing Data - Initial Intuition
#Here, we don't have any missing data.
#General Thumb Rules:

#For features with less missing values- can use regression to predict the missing values or fill with the mean of the values pres
#For features with very high number of missing values- it is better to drop those columns as they give very less insight on analy
#As there's no thumb rule on what criteria do we delete the columns with high number of missing values, but generally you can del
```

Data Cleaning

In [30]:
```
#1. Create a copy of base data for manupulation & processing
telco_data = telco_base_data.copy()
```

In [31]:
```
#2 Total Charges should be numeric amount. Let's convert it to numerical data type
telco_data.TotalCharges = pd.to_numeric(telco_data.TotalCharges, errors='coerce')
telco_data.isnull().sum()
```

Out[31]:
```
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 [32]:
```
# 3 As we can see there are 11 missing values in TotalCharges column. Let's check these records
telco_data.loc[telco_data ['TotalCharges'].isnull() == True]
```

Out[32]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtectio
488	4472-LVYGI	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	...	Ye
753	3115-CZMZD	Male	0	No	Yes	0	Yes	No	No	No internet service	...	No interne servic
936	5709-LVOEQ	Female	0	Yes	Yes	0	Yes	No	DSL	Yes	...	Ye
1082	4367-NUYAO	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service	...	No interne servic
1340	1371-DWPAZ	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	...	Ye
3331	7644-OMVMY	Male	0	Yes	Yes	0	Yes	No	No	No internet service	...	No interne servic
3826	3213-VVOLG	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service	...	No interne servic
4380	2520-SGTTA	Female	0	Yes	Yes	0	Yes	No	No	No internet service	...	No interne servic
5218	2923-ARZLG	Male	0	Yes	Yes	0	Yes	No	No	No internet service	...	No interne servic
6670	4075-WKNIU	Female	0	Yes	Yes	0	Yes	Yes	DSL	No	...	Ye
6754	2775-SEFEE	Male	0	No	Yes	0	Yes	Yes	DSL	Yes	...	N

11 rows × 21 columns

In [33]:
```
#4 Missing Value Treatement
#Since the % of these records compared to total dataset is very low ie 0.15%, it is safe to ignore them from further processing.

#Removing missing values
telco_data.dropna(how = 'any', inplace = True)

#telco_data.fillna(0)
```

In [34]:
```
#5. Divide customers into bins based on tenure e.g. for tenure < 12 months: assign a tenure group if 1-12, for tenure between 1 t

# Get the max tenure
print(telco_data['tenure'].max()) #72
```

```
72
```

In [35]:
```
# Group the tenure in bins of 12 months
labels = ["{0} - {1}".format(i, i + 11) for i in range(1, 72, 12)]

telco_data['tenure_group'] = pd.cut(telco_data.tenure, range(1, 80, 12), right=False, labels=labels)
```

In [36]:
```
telco_data['tenure_group'].value_counts()
```

Out[36]:
```
1 - 12     2175
61 - 72    1407
13 - 24    1024
25 - 36     832
49 - 60     832
37 - 48     762
Name: tenure_group, dtype: int64
```

In [37]:
```
#6. Remove columns not required for processing
#drop column customerID and tenure

telco_data.drop(columns= ['customerID','tenure'], axis=1, inplace=True)
telco_data.head()
```

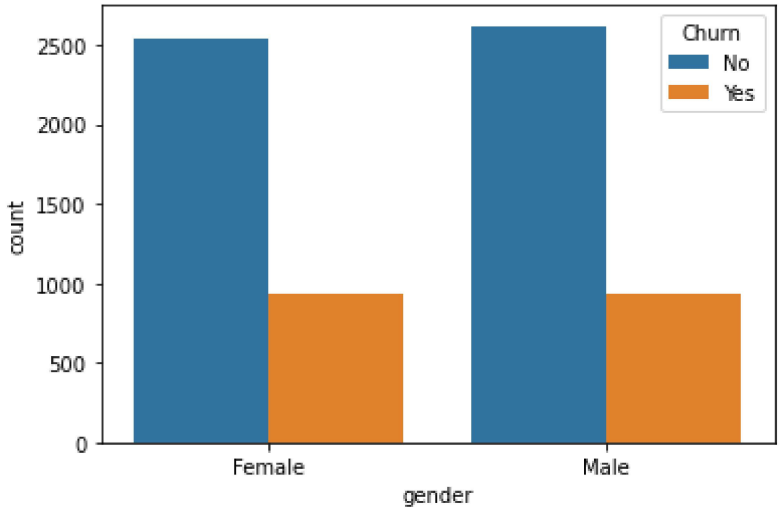
Out[37]:

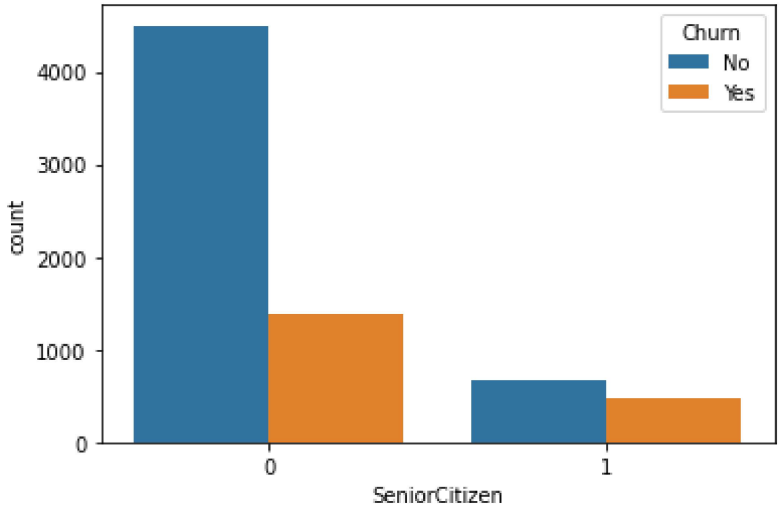
	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSuppo
0	Female	0	Yes	No	No	No phone service	DSL	No	Yes	No	N
1	Male	0	No	No	Yes	No	DSL	Yes	No	Yes	N
2	Male	0	No	No	Yes	No	DSL	Yes	Yes	No	N
3	Male	0	No	No	No	No phone service	DSL	Yes	No	Yes	Y
4	Female	0	No	No	Yes	No	Fiber optic	No	No	No	N

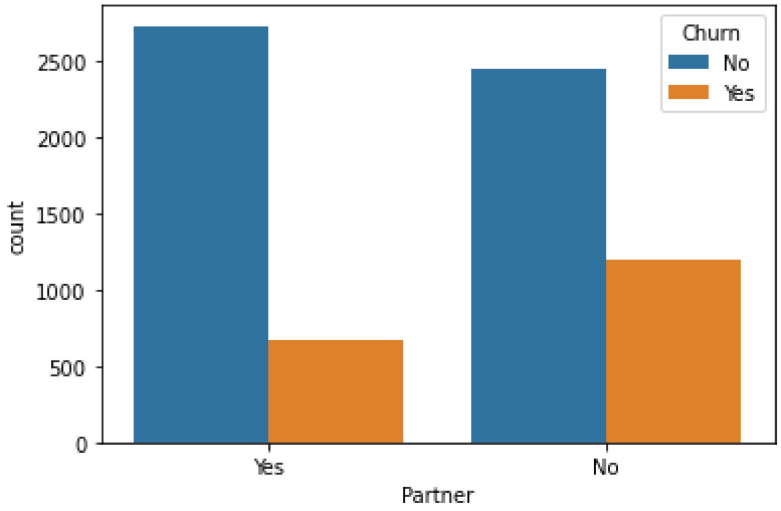
Data Exploration

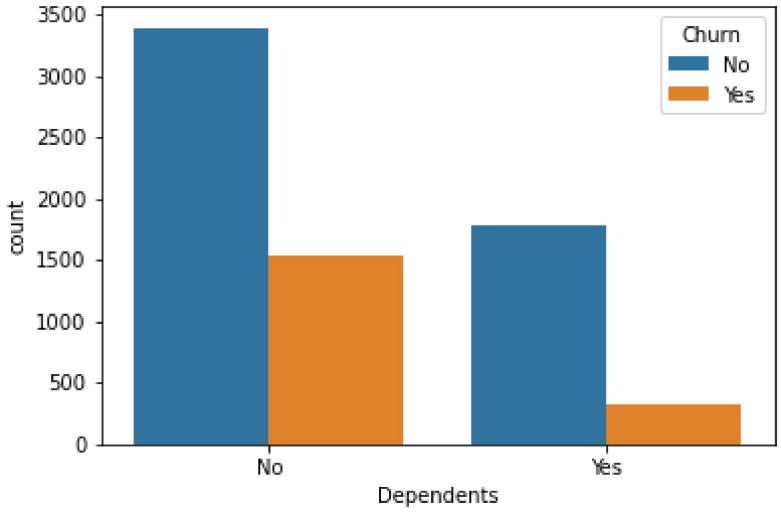
In [38]:

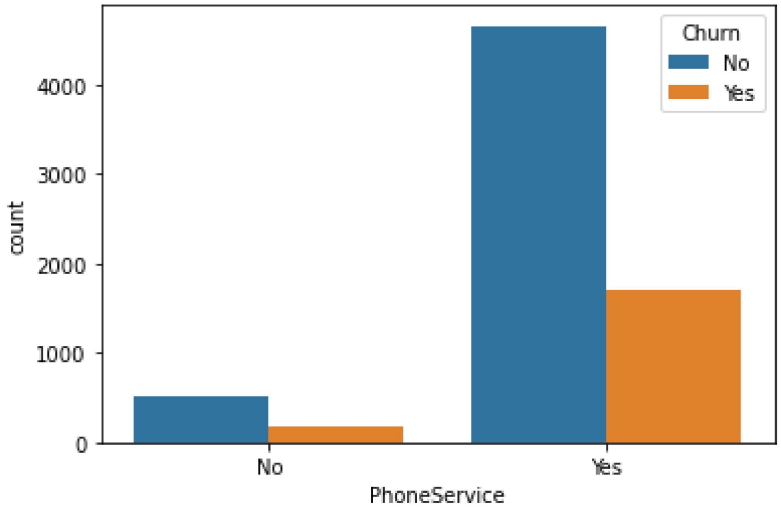
```
#1. Plot distibution of individual predictors by churn
#Univariate Analysis
for i, predictor in enumerate(telco_data.drop(columns=['Churn', 'TotalCharges', 'MonthlyCharges'])):
    plt.figure(i)
    sns.countplot(data=telco_data, x=predictor, hue='Churn')
```

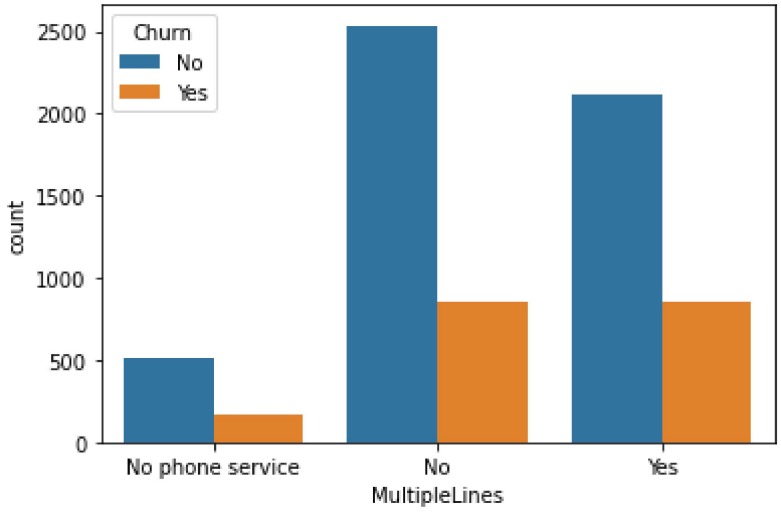

Churn
No
Yes
2500
2000
1500
1000
500
0
count
Female
Male
gender

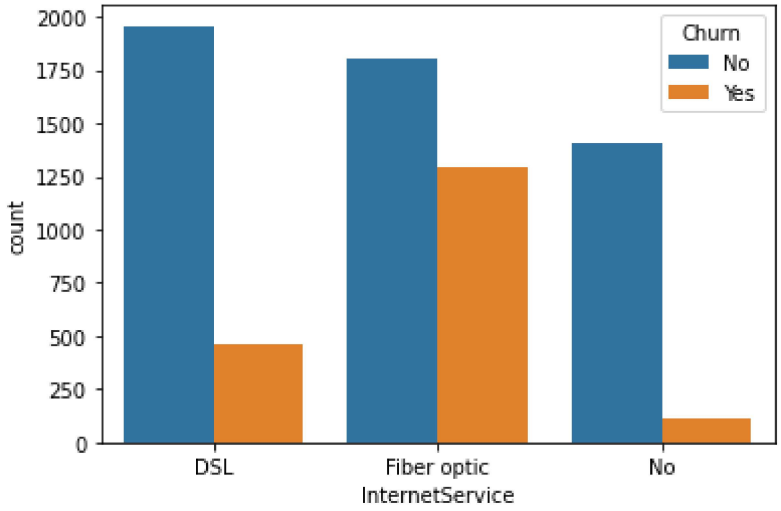

Churn
No
Yes
4000
3000
2000
1000
0
count
0
1
SeniorCitizen

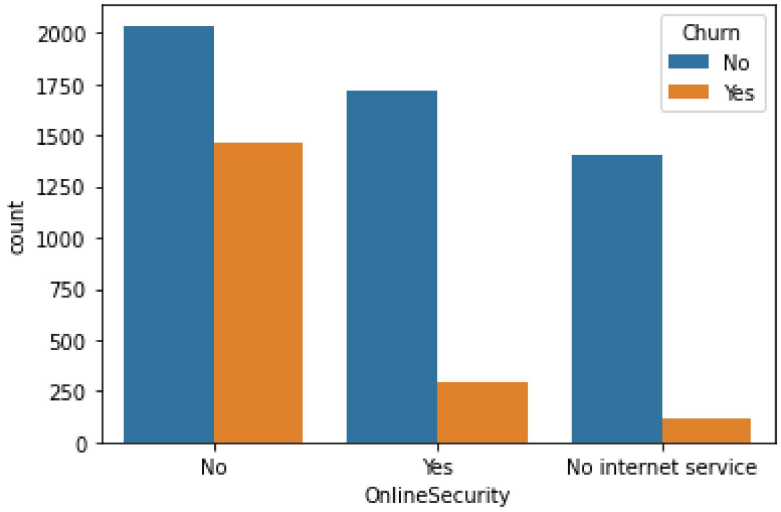

Churn
No
Yes
count
2500
2000
1500
1000
500
0
Yes
No
Partner

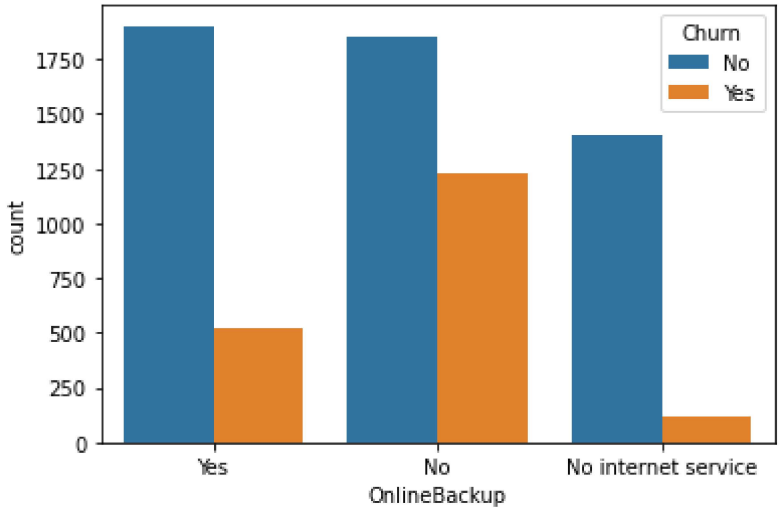

Churn
No
Yes
count
3500
3000
2500
2000
1500
1000
500
0
No
Yes
Dependents

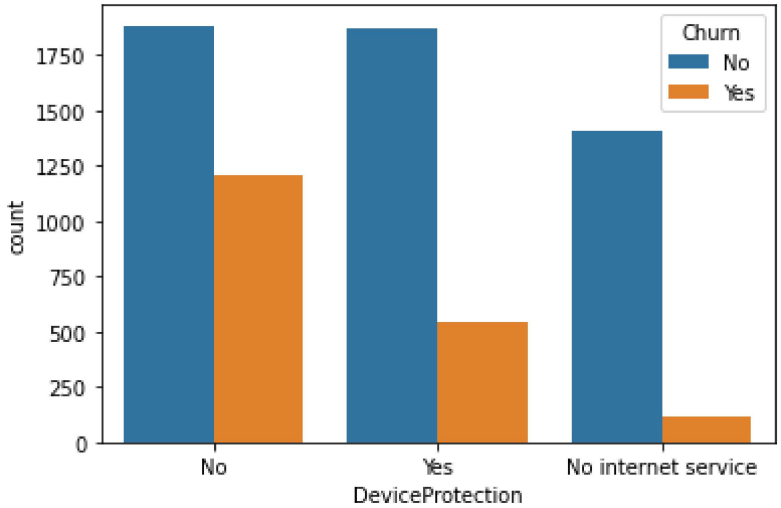

Churn
No
Yes
4000
3000
2000
1000
0
count
No
Yes
PhoneService

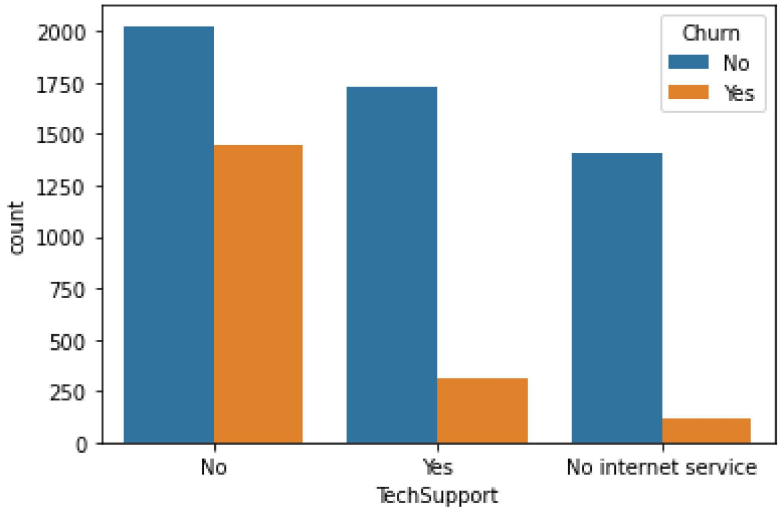

Churn
No
Yes
2500
2000
1500
1000
500
0
count
No phone service
No
Yes
MultipleLines

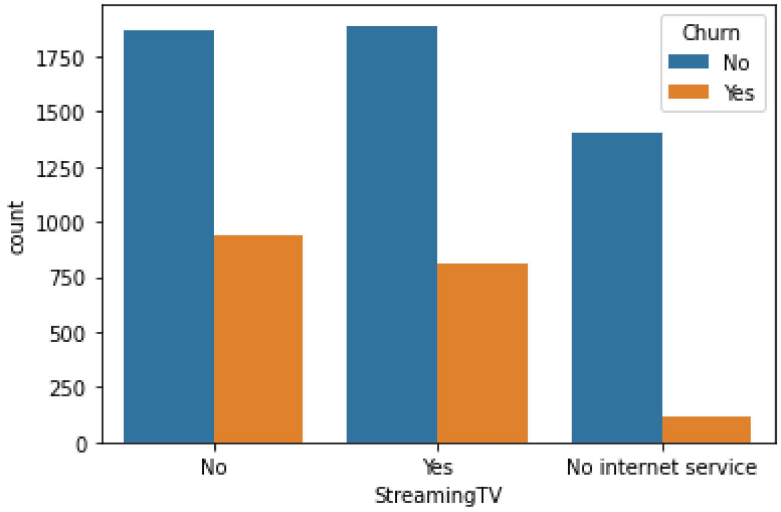

Churn
No
Yes
count
2000
1750
1500
1250
1000
750
500
250
0
DSL
Fiber optic
No
InternetService

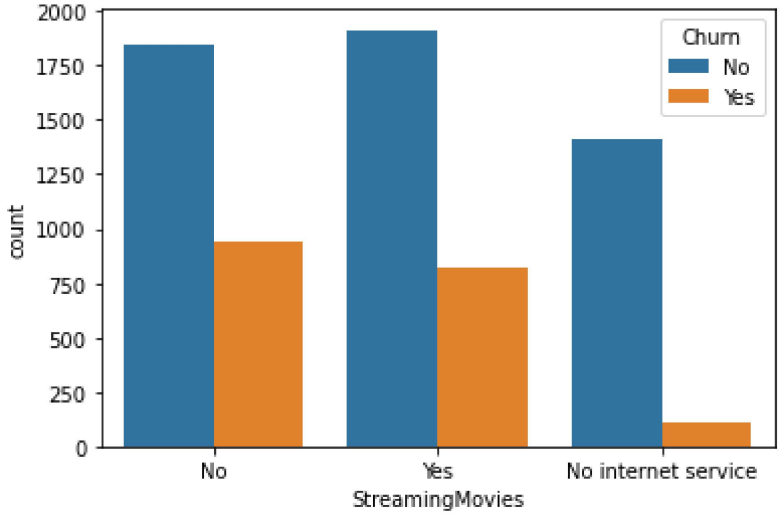

Churn
No
Yes
count
2000
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
OnlineSecurity

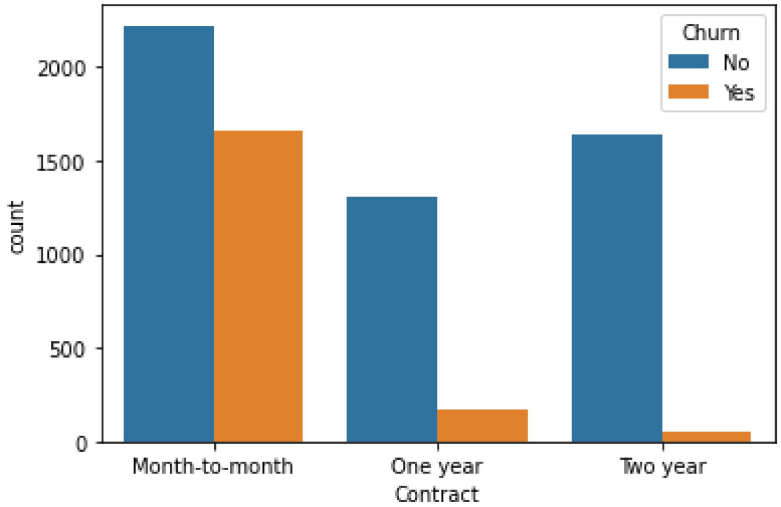

Churn
No
Yes
count
1750
1500
1250
1000
750
500
250
0
Yes
No
No internet service
OnlineBackup

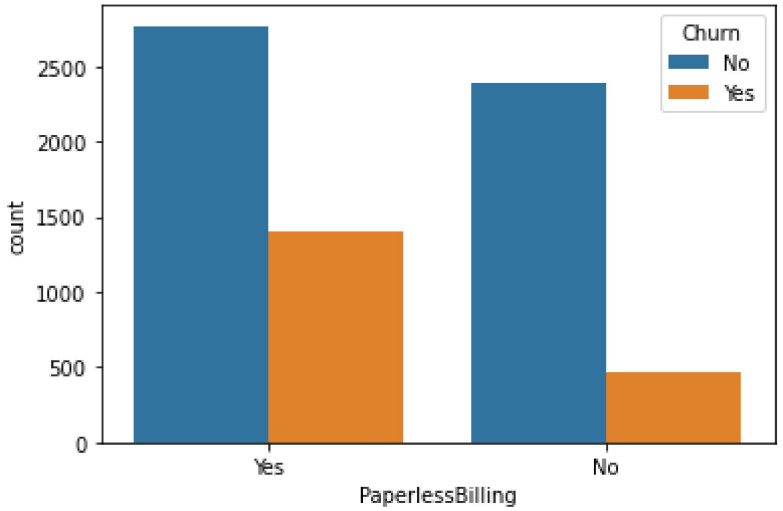

Churn
No
Yes
count
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
DeviceProtection

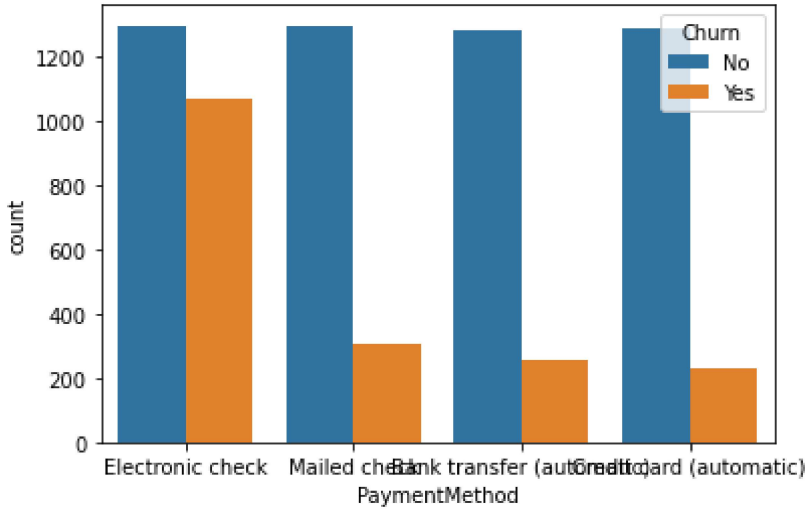

Churn
No
Yes
count
2000
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
TechSupport

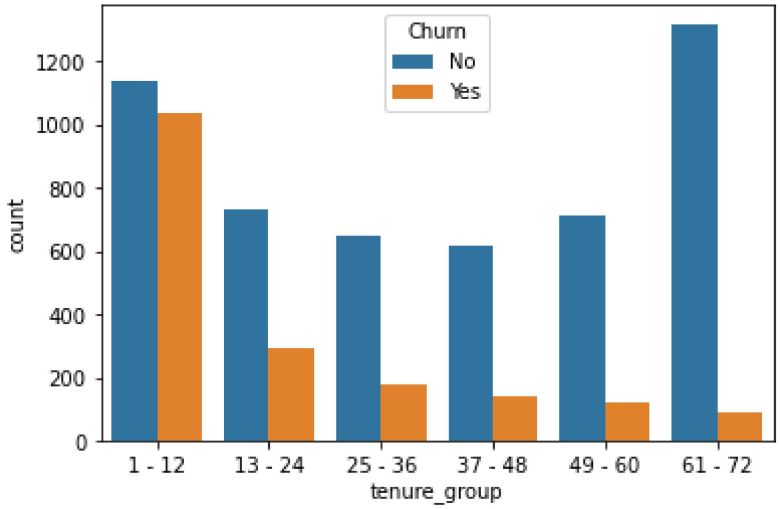

Churn
No
Yes
count
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
StreamingTV


Churn
No
Yes
count
2000
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
StreamingMovies


Churn
No
Yes
count
2000
1500
1000
500
0
Month-to-month
One year
Two year
Contract


Churn
No
Yes
count
2500
2000
1500
1000
500
0
Yes
No
PaperlessBilling


Churn
No
Yes
count
1200
1000
800
600
400
200
0
Electronic check
Mailed check
Bank transfer (automatic)
Credit card (automatic)
PaymentMethod



In [39]:
```
#2. Convert the target variable 'Churn' in a binary numeric variable i.e. Yes=1 ; No = 0
telco_data['Churn'] = np.where(telco_data.Churn == 'Yes',1,0)
telco_data.head()
```

Out[39]:

	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSuppo
0	Female	0	Yes	No	No	No phone service	DSL	No	Yes	No	N
1	Male	0	No	No	Yes	No	DSL	Yes	No	Yes	N
2	Male	0	No	No	Yes	No	DSL	Yes	Yes	No	N
3	Male	0	No	No	No	No phone service	DSL	Yes	No	Yes	Y
4	Female	0	No	No	Yes	No	Fiber optic	No	No	No	N

In [40]:
```
#3. Convert all the categorical variables into dummy variables
telco_data_dummies = pd.get_dummies(telco_data)
```

```
telco_data_dummies.head()
```

Out[40]:

	SeniorCitizen	MonthlyCharges	TotalCharges	Churn	gender_Female	gender_Male	Partner_No	Partner_Yes	Dependents_No	Dependents_Yes	...	Pay t
0	0	29.85	29.85	0	1	0	0	1	1	0	...	
1	0	56.95	1889.50	0	0	1	1	0	1	0	...	
2	0	53.85	108.15	1	0	1	1	0	1	0	...	
3	0	42.30	1840.75	0	0	1	1	0	1	0	...	
4	0	70.70	151.65	1	1	0	1	0	1	0	...	

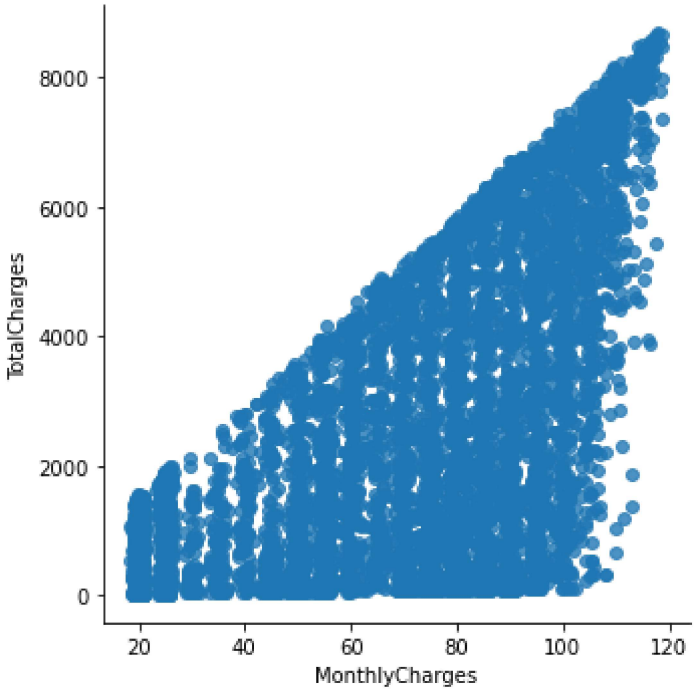
5 rows × 51 columns

In [41]:

```
#4. Relationship between Monthly Charges and Total Charges
sns.lmplot(data=telco_data_dummies, x='MonthlyCharges', y='TotalCharges', fit_reg=False)
```

Out[41]:

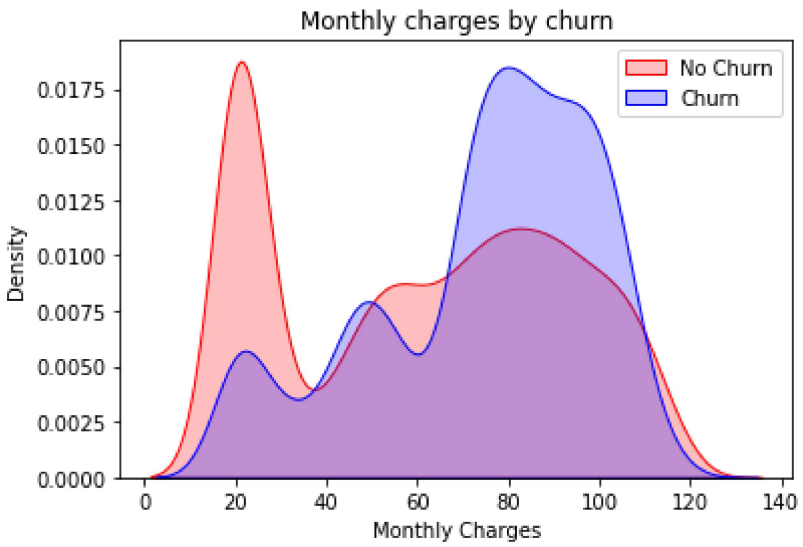
```
<seaborn.axisgrid.FacetGrid at 0x1d91dd1a400>
```



In [42]:
```
#Total Charges increase as Monthly Charges increase - as expected.
```

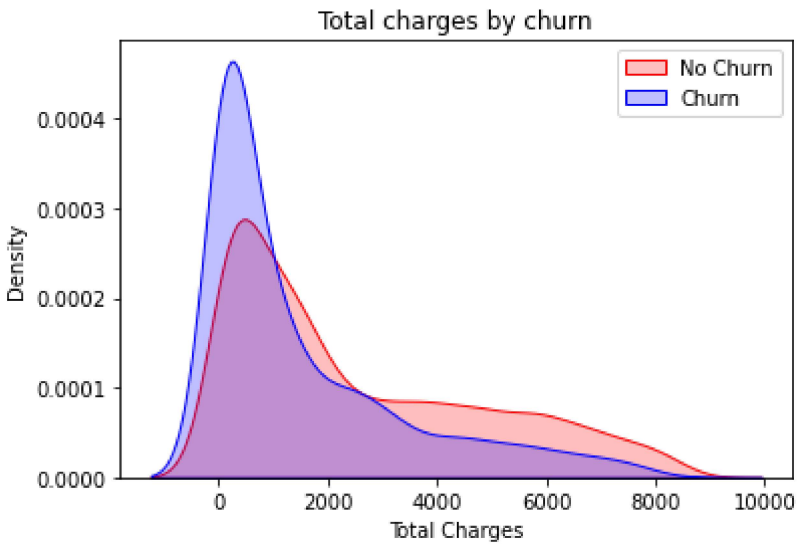
In [43]:
```
#5. Churn by Monthly Charges and Total Charges
Mth = sns.kdeplot(telco_data_dummies.MonthlyCharges[(telco_data_dummies["Churn"] == 0) ],
                 color="Red", shade = True)
Mth = sns.kdeplot(telco_data_dummies.MonthlyCharges[(telco_data_dummies["Churn"] == 1) ],
                 ax =Mth, color="Blue", shade= True)
Mth.legend(["No Churn","Churn"],loc='upper right')
Mth.set_ylabel('Density')
Mth.set_xlabel('Monthly Charges')
Mth.set_title('Monthly charges by churn')
```

Out[43]:
```
Text(0.5, 1.0, 'Monthly charges by churn')
```



In [44]:
```
Tot = sns.kdeplot(telco_data_dummies.TotalCharges[(telco_data_dummies["Churn"] == 0) ],
                 color="Red", shade = True)
Tot = sns.kdeplot(telco_data_dummies.TotalCharges[(telco_data_dummies["Churn"] == 1) ],
                 ax =Tot, color="Blue", shade= True)
Tot.legend(["No Churn","Churn"],loc='upper right')
Tot.set_ylabel('Density')
Tot.set_xlabel('Total Charges')
Tot.set_title('Total charges by churn')
```

Out[44]:
```
Text(0.5, 1.0, 'Total charges by churn')
```



In [45]:
```
#Surprising insight as higher Churn at lower Total Charges

#However if we combine the insights of 3 parameters i.e. Tenure, Monthly Charges & Total Charges then the picture is bit clear :-
```

In [47]:
```
#CONCLUSION

#Electronic check medium are the highest churners
#Contract Type - Monthly customers are more likely to churn because of no contract terms, as they are free to go customers.
#No Online security, No Tech Support category are high churners
#Non senior Citizens are high churners
#Note: There could be many more such insights, so take this as an assignment and try to get more insights :)
```

In [52]:
```
telco_data_dummies.to_csv('tel_churn.csv')
```

Model Building

In [48]:
```
import pandas as pd
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import recall_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

```
from sklearn.tree import DecisionTreeClassifier
from imblearn.combine import SMOTEENN
```

In [53]:
```
df=pd.read_csv("tel_churn.csv")
df.head()
```

Out[53]:

	Unnamed: 0	SeniorCitizen	MonthlyCharges	TotalCharges	Churn	gender_Female	gender_Male	Partner_No	Partner_Yes	Dependents_No	...	Payment transfe
0	0	0	29.85	29.85	0	1	0	0	1	1	...	
1	1	0	56.95	1889.50	0	0	1	1	0	1	...	
2	2	0	53.85	108.15	1	0	1	1	0	1	...	
3	3	0	42.30	1840.75	0	0	1	1	0	1	...	
4	4	0	70.70	151.65	1	1	0	1	0	1	...	

5 rows × 52 columns

In [54]:
```
df=df.drop('Unnamed: 0',axis=1)
```

In [55]:
```
x=df.drop('Churn',axis=1)
x
```

Out[55]:

	SeniorCitizen	MonthlyCharges	TotalCharges	gender_Female	gender_Male	Partner_No	Partner_Yes	Dependents_No	Dependents_Yes	PhoneServi
0	0	29.85	29.85	1	0	0	1	1	0	
1	0	56.95	1889.50	0	1	1	0	1	0	
2	0	53.85	108.15	0	1	1	0	1	0	
3	0	42.30	1840.75	0	1	1	0	1	0	
4	0	70.70	151.65	1	0	1	0	1	0	
...	...	...	...	...	...	...	...	...	...	
7027	0	84.80	1990.50	0	1	0	1	0	1	
7028	0	103.20	7362.90	1	0	0	1	0	1	
7029	0	29.60	346.45	1	0	0	1	0	1	
7030	1	74.40	306.60	0	1	0	1	1	0	
7031	0	105.65	6844.50	0	1	1	0	1	0	

7032 rows × 50 columns

In [56]:

```
y=df['Churn']
y
```

Out[56]:

```
0       0
1       0
2       1
3       0
4       1
       ..
7027    0
7028    0
7029    0
7030    1
7031    0
Name: Churn, Length: 7032, dtype: int64
```

In [58]:
```
#Train Test Split

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
```

In [59]:
```
#Decision Tree Classifier
model_dt=DecisionTreeClassifier(criterion = "gini",random_state = 100,max_depth=6, min_samples_leaf=8)
model_dt.fit(x_train,y_train)
```

Out[59]:
```
DecisionTreeClassifier(max_depth=6, min_samples_leaf=8, random_state=100)
```

In [60]:
```
y_pred=model_dt.predict(x_test)
y_pred
```

Out[60]:
```
array([0, 0, 0, ..., 0, 1, 0], dtype=int64)
```

In [62]:
```
model_dt.score(x_test,y_test)
```

Out[62]:
```
0.7739872068230277
```

In [63]:
```
print(classification_report(y_test, y_pred, labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.81      0.90      0.85      1016
           1       0.63      0.45      0.52       391

    accuracy                           0.77      1407
   macro avg       0.72      0.67      0.69      1407
weighted avg       0.76      0.77      0.76      1407
```

In [64]:
```
#As you can see that the accuracy is quite low, and as it's an imbalanced dataset, we shouldn't consider Accuracy as our metrics
#Hence, we need to check recall, precision & f1 score for the minority class, and it's quite evident that the precision, recall &
#Hence, moving ahead to call SMOTEENN (UpSampling + ENN)
```

In [68]:
```
sm = SMOTEENN()
X_resampled, y_resampled = sm.fit_resample(x,y)
```

In [69]:
```
xr_train,xr_test,yr_train,yr_test=train_test_split(X_resampled, y_resampled,test_size=0.2)
```

In [70]:
```
model_dt_smote=DecisionTreeClassifier(criterion = "gini",random_state = 100,max_depth=6, min_samples_leaf=8)
```

In [71]:
```
model_dt_smote.fit(xr_train,yr_train)
yr_predict = model_dt_smote.predict(xr_test)
model_score_r = model_dt_smote.score(xr_test, yr_test)
print(model_score_r)
print(metrics.classification_report(yr_test, yr_predict))
```

```
0.9490500863557858
              precision    recall  f1-score   support

           0       0.95      0.94      0.94       521
           1       0.95      0.96      0.95       637

    accuracy                           0.95      1158
   macro avg       0.95      0.95      0.95      1158
weighted avg       0.95      0.95      0.95      1158
```

In [72]:
```
print(metrics.confusion_matrix(yr_test, yr_predict))
```

```
[[490  31]
 [ 28 609]]
```

In [73]:
```
#Now we can see quite better results, i.e. Accuracy: 92 %, and a very good recall, precision & f1 score for minority class.
```

Random Forest Classifier

In [74]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [75]:
```
model_rf=RandomForestClassifier(n_estimators=100, criterion='gini', random_state = 100,max_depth=6, min_samples_leaf=8)
```

In [76]:
```
model_rf.fit(x_train,y_train)
```

Out[76]:
```
RandomForestClassifier(max_depth=6, min_samples_leaf=8, random_state=100)
```

In [77]:
```
y_pred=model_rf.predict(x_test)
```

In [78]:
```
model_rf.score(x_test,y_test)
```

Out[78]: 0.7974413646055437

In [79]:
```
print(classification_report(y_test, y_pred, labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.82      0.93      0.87      1016
           1       0.71      0.45      0.55       391

    accuracy                           0.80      1407
   macro avg       0.76      0.69      0.71      1407
weighted avg       0.79      0.80      0.78      1407
```

In [81]:
```
sm = SMOTEENN()
X_resampled1, y_resampled1 = sm.fit_resample(x,y)
```

In [82]:
```
xr_train1,xr_test1,yr_train1,yr_test1=train_test_split(X_resampled1, y_resampled1,test_size=0.2)
```

In [83]:
```
model_rf_smote=RandomForestClassifier(n_estimators=100, criterion='gini', random_state = 100,max_depth=6, min_samples_leaf=8)
```

In [84]:
```
model_rf_smote.fit(xr_train1,yr_train1)
```

Out[84]:
```
RandomForestClassifier(max_depth=6, min_samples_leaf=8, random_state=100)
```

In [85]:
```
yr_predict1 = model_rf_smote.predict(xr_test1)
```

In [87]:
```
model_score_r1 = model_rf_smote.score(xr_test1, yr_test1)
```

In [88]:
```
print(model_score_r1)
print(metrics.classification_report(yr_test1, yr_predict1))
```

```
0.9413763806287171
              precision    recall  f1-score   support

           0       0.95      0.92      0.93       520
           1       0.94      0.96      0.95       657

    accuracy                           0.94      1177
   macro avg       0.94      0.94      0.94      1177
weighted avg       0.94      0.94      0.94      1177
```

In [89]:
```
print(metrics.confusion_matrix(yr_test1, yr_predict1))
```

```
[[478  42]
 [ 27 630]]
```

In [90]:
```
#With RF Classifier, also we are able to get quite good results, infact better than Decision Tree.
#We can now further go ahead and create multiple classifiers to see how the model performance is, but that's not covered here, so
```

In [91]:
```
#Pickling the model
```

In [92]:
```
import pickle
```

In [93]:
```
filename = 'model.sav'
```

In [94]:
```
pickle.dump(model_rf_smote, open(filename, 'wb'))
```

In [95]:
```
load_model = pickle.load(open(filename, 'rb'))
```

In [96]:
```
model_score_r1 = load_model.score(xr_test1, yr_test1)
```

In [97]:
```
model_score_r1
```

Out[97]:
```
0.9413763806287171
```

In [98]:
```
#Our final model i.e. RF Classifier with SMOTEENN, is now ready and dumped in model.sav, which we will use and prepare API's so t
```