PROJECT: Customer Churn Analysis

In [42]:

```
#Importing the libreris like pandas, numpy for selecing the data and convering the data tc
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
# importing the matplotlib and seaborn for visualizing the data present in the dataset df
import sklearn
import matplotlib.pyplot as plt
import seaborn as sns
# importing the logisticregressor for training and predicting the output
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

In [43]:

```
#Creating the variable df and loading the dataset to the variable df
df = pd.read_csv('churndata')
df
```

Out[43]:

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

7043 rows × 21 columns

In [44]:

```
# checking the dimention of the df
print('Shape:\n', df.shape, '\n')
```

```
Shape:
 (7043, 21)
```

Loading [MathJax]/extensions/Safe.js

df dataset as 70243 rows and 21 columns

In [45]:

```
# checking for the columns name of the df dataset
print('Columns:\n', df.columns.values, '\n')
```

```
Columns:
 ['customerID' 'gender' 'SeniorCitizen' 'Partner' 'Dependents' 'tenure'
 'PhoneService' 'MultipleLines' 'InternetService' 'OnlineSecurity'
 'OnlineBackup' 'DeviceProtection' 'TechSupport' 'StreamingTV'
 'StreamingMovies' 'Contract' 'PaperlessBilling' 'PaymentMethod'
 'MonthlyCharges' 'TotalCharges' 'Churn']
```

In [46]:

```
# Check columns for missing values:
print('Missing Values:\n', df.isna().sum(),)
```

```
Missing Values:
 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
```

df dataset is not having any null values

In [47]:

```
# Summary Statistics:
df.describe()
```

Out[47]:

	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

Loading [MathJax]/extensions/Safe.js

```
r churn count:
```

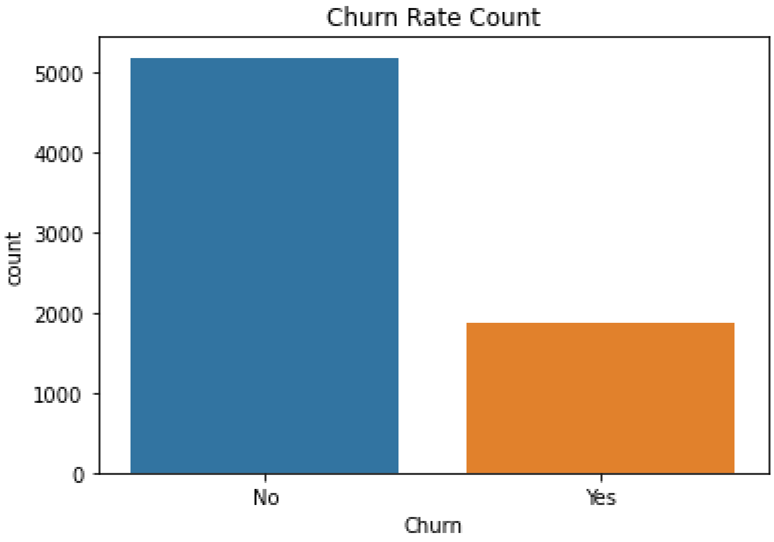
```
print('\n--Churn Value Counts--\n', df['Churn'].value_counts())
```

```
--Churn Value Counts--
 No     5174
Yes    1869
Name: Churn, dtype: int64
```

In [49]:

```
# Create a bar graph of our count:
sns.countplot(data=df, x='Churn').set(title="Churn Rate Count")

# Define yes/no conditions:
NO = df['Churn'] == 'No'
YES = df['Churn'] == 'Yes'

num_retained = df[NO].shape[0]
num_churned = df[YES].shape[0]

# Percentage of customer that have stayed vs. those who've left:
retain_rate = num_retained/(num_churned + num_retained) * 100
churn_rate = num_churned/(num_churned + num_retained) * 100

print(round(retain_rate, 3), "% of customers stayed.")
print(round(churn_rate, 3), "% of customers left.")
```

```
73.463 % of customers stayed.
26.537 % of customers left.
```



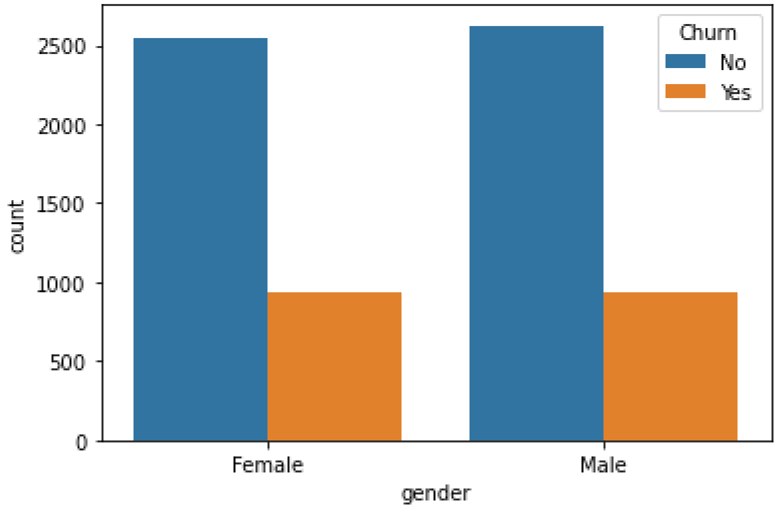
In [50]:

```
# Create bar graph based on gender:
sns.countplot(data=df, x='gender', hue='Churn')
```

Out[50]:

```
<AxesSubplot:xlabel='gender', ylabel='count'>
```

Loading [MathJax]/extensions/Safe.js

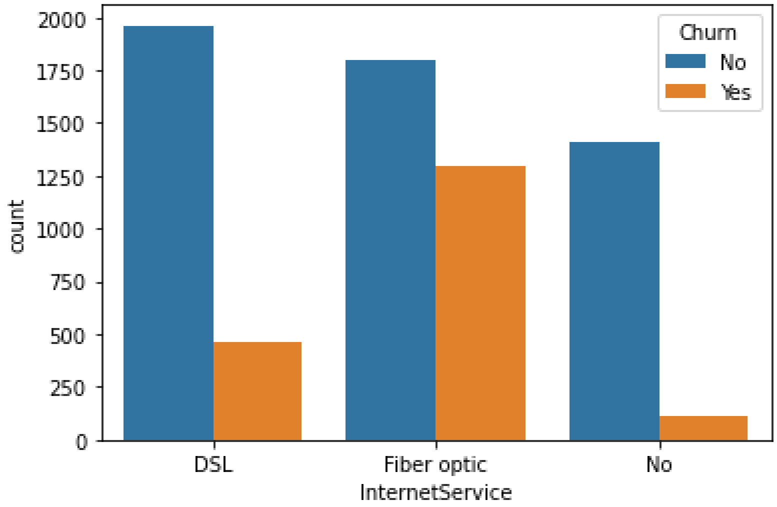


In [51]:

```
# Create bar graph based on customer's internet service:
sns.countplot(data=df, x='InternetService', hue='Churn')
```

Out[51]:

```
<AxesSubplot:xlabel='InternetService', ylabel='count'>
```



In [52]:

```
# Drop the id column:
clean_df = df.drop('customerID', axis=1)

# Convert non-numerical values to numerical:
for col in clean_df.columns:
    if (clean_df[col].dtype == np.number):
      continue
    clean_df[col] = LabelEncoder().fit_transform(clean_df[col])

# See the data types of our columns:
print(clean_df.dtypes)
```

```
gender                      int32
SeniorCitizen               int64
Partner                     int32
Dependents                  int32
tenure                      int64
PhoneService                int32
MultipleLines               int32
InternetService             int32
OnlineSecurity              int32
OnlineBackup                int32
DeviceProtection            int32
                            int32
```

Loading [MathJax]/extensions/Safe.js

```
StreamingTV           int32
StreamingMovies       int32
Contract              int32
PaperlessBilling      int32
PaymentMethod         int32
MonthlyCharges      float64
TotalCharges          int32
Churn                 int32
dtype: object
```

In [53]:

```
# Scale the data set:
x = clean_df.drop('Churn', axis=1)
y = clean_df['Churn']
x = StandardScaler().fit_transform(x)

# Split the data 80 training/20 testing:
#xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.2, random_state=7)
```

In [54]:

```
# Use the logisitic regression algorithm:
model = LogisticRegression()

# Train our model / fit our data:
model.fit(xtrain, ytrain)
```

Out[54]:

```
LogisticRegression()
```

In [55]:

```
# Create predictions from the test data:
predictions = model.predict(xtest)

# See preview of predictions:
print(predictions)
```

```
[0 1 1 ... 0 0 1]
```

In [56]:

```
# Create a report of the accuracy of our classifications:
report = classification_report(ytest, predictions)

print(report)
```

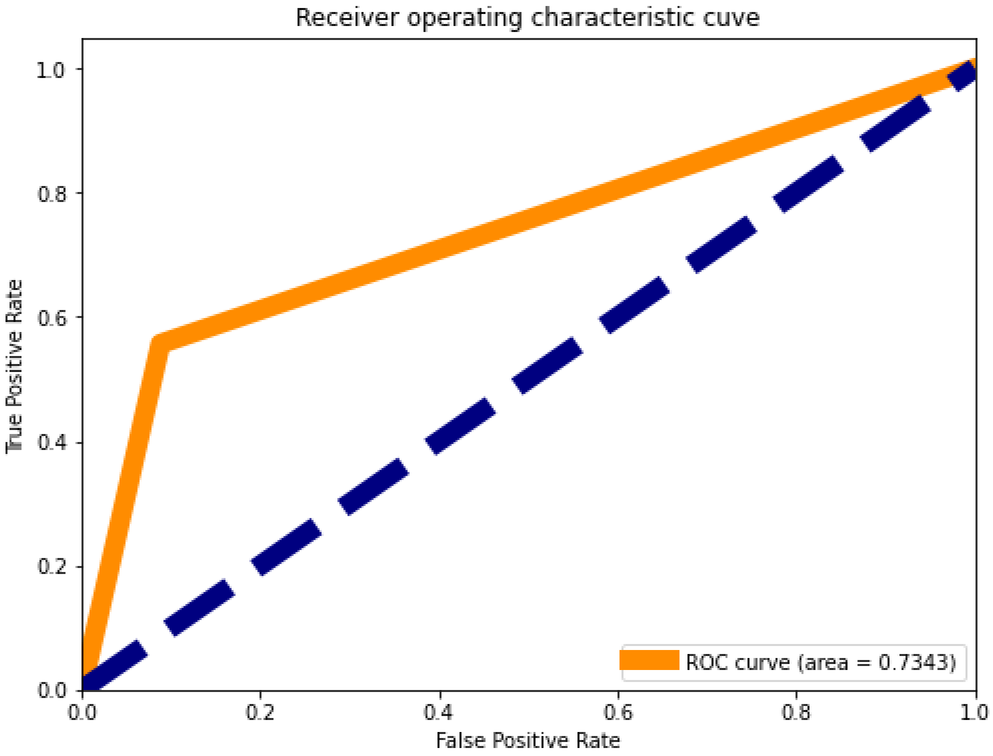
```
              precision    recall  f1-score   support

           0       0.84      0.91      0.88      1021
           1       0.71      0.56      0.62       388

    accuracy                           0.81      1409
   macro avg       0.77      0.73      0.75      1409
weighted avg       0.81      0.81      0.81      1409
```

In [57]:

```
#aoc-roc curve
from sklearn.metrics import roc_curve,auc
fpr, tpr, thresholds = roc_curve(ytest, predictions)
roc_auc=auc(fpr,tpr)


plt.figure(figsize=(8, 6))
plt.plot( fpr, tpr,color='darkorange',lw=10,label='ROC curve (area = %0.4f)' % roc_auc)
plt.plot([0, 1], [0, 1],color='navy',lw=10,linestyle='--')
plt.xlim([0.0, 1.0])
                 , 1.05])
```

Loading [MathJax]/extensions/Safe.js

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic cuve')
plt.legend(loc="lower right")
plt.show()
```



1) area under the curve is 73.43percent

2) last step of the project,we know the better performance algorith which is LogisticRegression(), then save the model by usig pickel