# Customer Churn

#### Problem Statement:

Customer churn is when a company’s customers stop doing business with that company. Businesses are very keen on measuring churn because **keeping an existing customer is far less expensive than acquiring a new customer.** New business involves working leads through a sales funnel, using marketing and sales budgets to gain additional customers. Existing customers will often have a higher volume of service consumption and can generate additional customer referrals.

Customer retention can be achieved with good customer service and products. But the most effective way for a company to prevent attrition of customers is to truly know them. The vast volumes of data collected about customers can be used to build churn prediction models. Knowing who is most likely to defect means that a company can prioritise focused marketing efforts on that subset of their customer base.

Preventing customer churn is critically important to the telecommunications sector, as the barriers to entry for switching services are so low.

#### Objective:

Examine customer data from IBM Sample Data Sets with the aim of building and comparing several customer churn prediction models.

# Introduction – What is Customer Churn?

Customer churn refers to the rate at which customers discontinue their service, or simply speaking; it is the rate at which customers leave a company/brand and move to another company/brand. It is important to know how many of our customers are leaving because it can help us understand why they are leaving, what we need to do about it, and where we need to invest in attracting new customers into our fold.

# Why is Customer Churn analysis critical?

Customer churn analysis is essential because it helps us understand the reasons behind customer attrition and take necessary actions to reduce the customer churn rate.

It helps us identify our customers who are likely to leave and take preventive measures to retain them.

Customer churn analysis is helpful in improving customer retention, satisfaction, lifetime value, acquisition, and more.

# Customer Churn Rate Formula

The churn rate formula is: (Lost Customers ÷ Total Customers at the Start of Time Period) x 100.

The total customer base is the number of customers in our system at some point. For example, if we have 100 users and they all stayed with us for three months, that is a 100% retention rate. If someone leaves before their expected lifetime (or after it), their #'s are not included in this calculation.

# Data Analysis

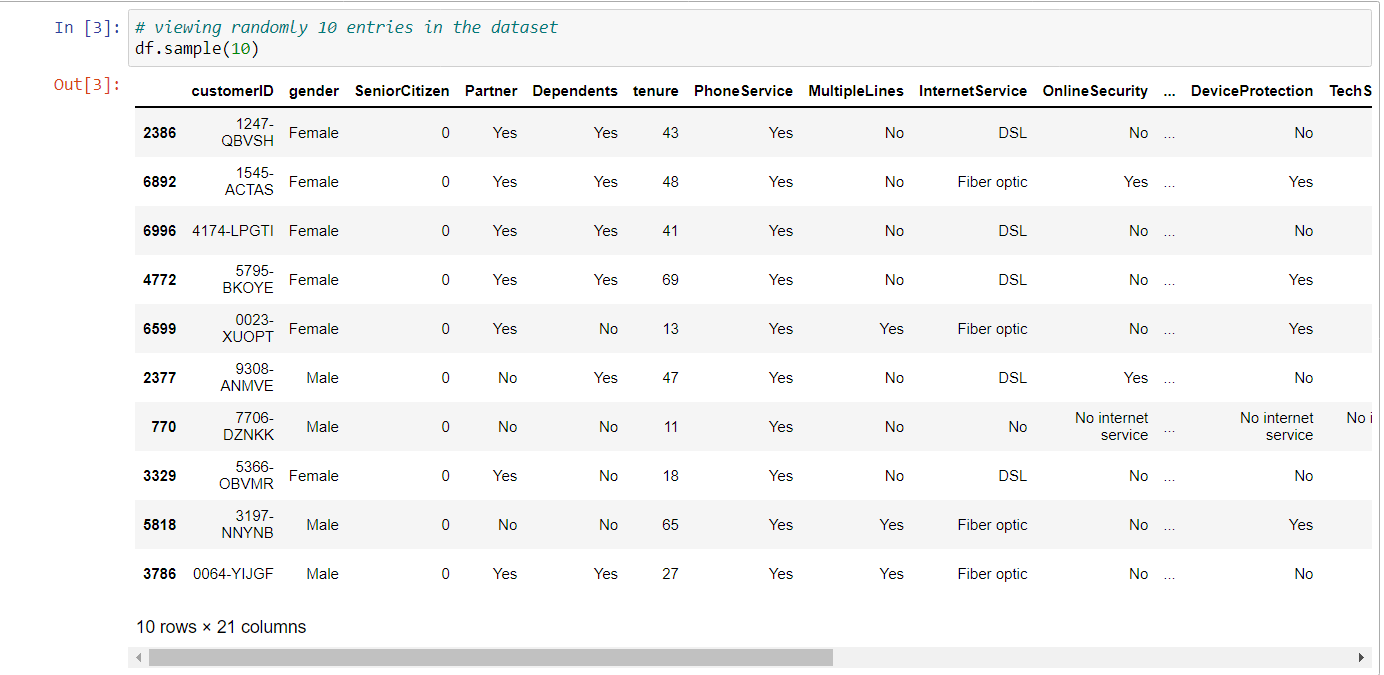
Data analysis is the process of deriving meaning from data. We can use it to understand what is going on in our customer base and how we can improve our business.

To do this, we need an understanding of the business model, so let us start there:

# importing necessary libraries for eda  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
import warnings  
warnings.filterwarnings('ignore')

# reading the csv file  
df = pd.read\_csv('Telecom\_customer\_churn.csv')

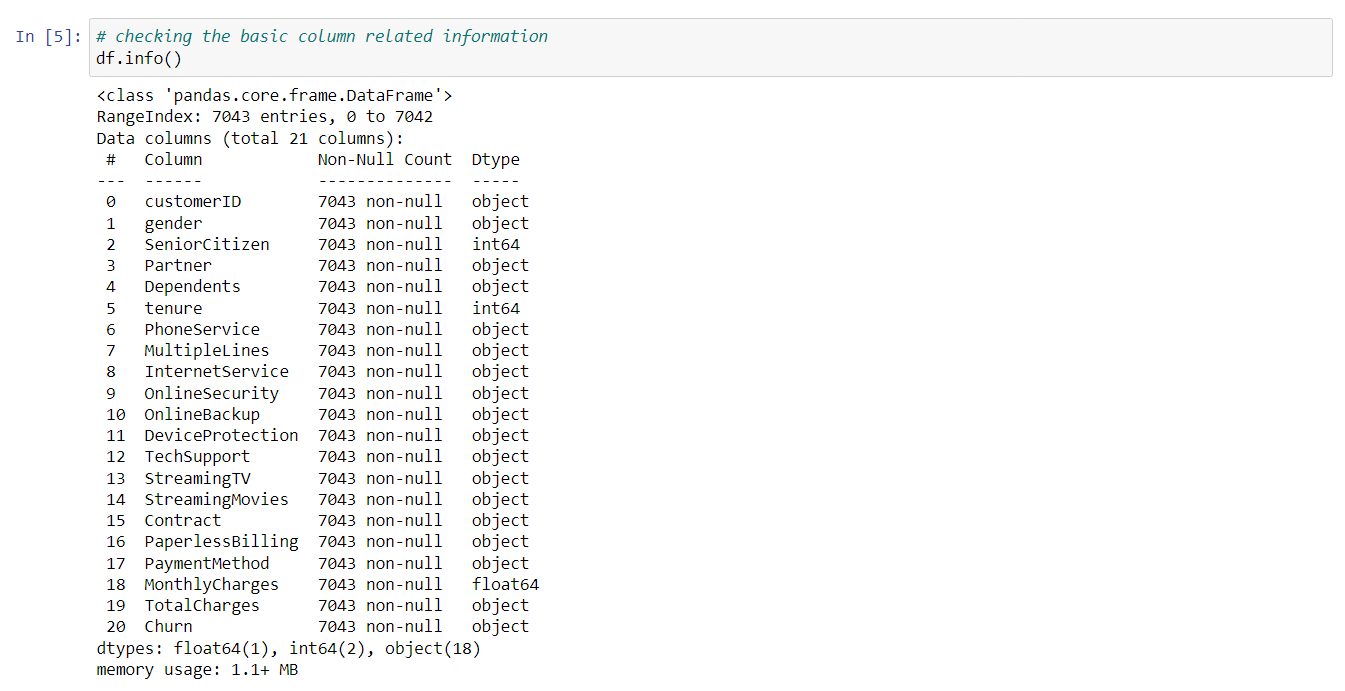
# viewing randomly 10 entries in the dataset  
df.sample(10)



# checking the shape of the dataset  
df.shape

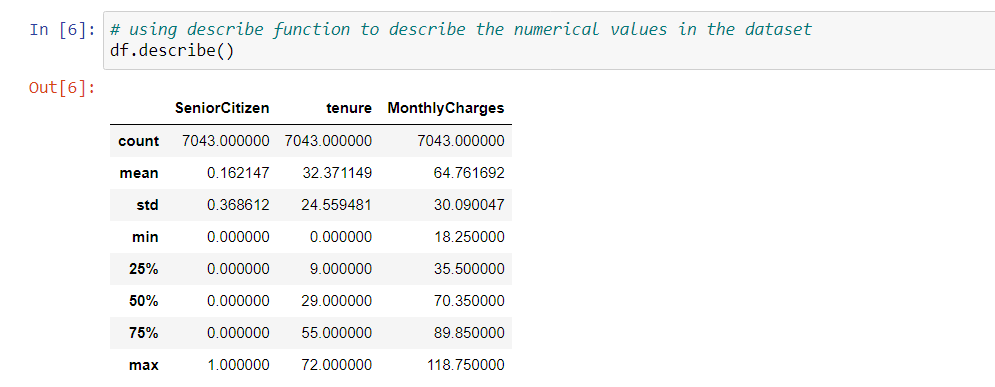
(7043, 21)

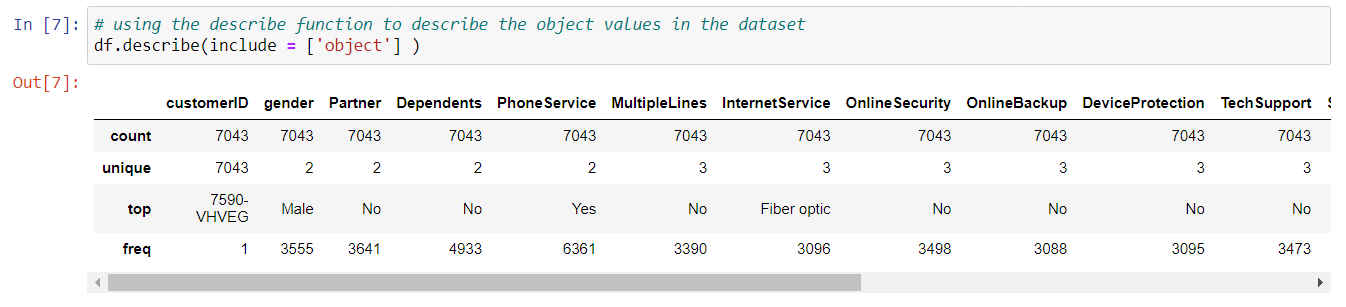
#### We have 7043 enteries in the dataset with 21 columns.

# checking the basic column related information  
df.info()

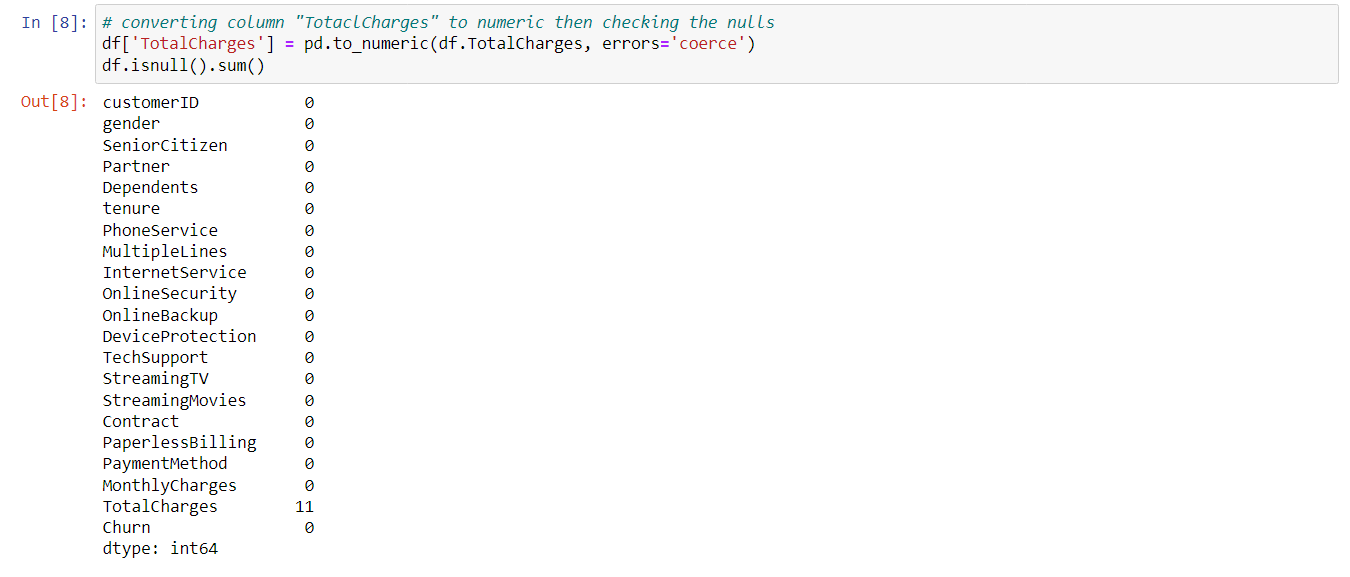
#### TotalCharges is being shown as object datatype but we know it should be float. Treating that below.

#### We can see that there seems to be no null values in our dataset. Let's verify this but after converting the data type for TotalCharges to numerical.

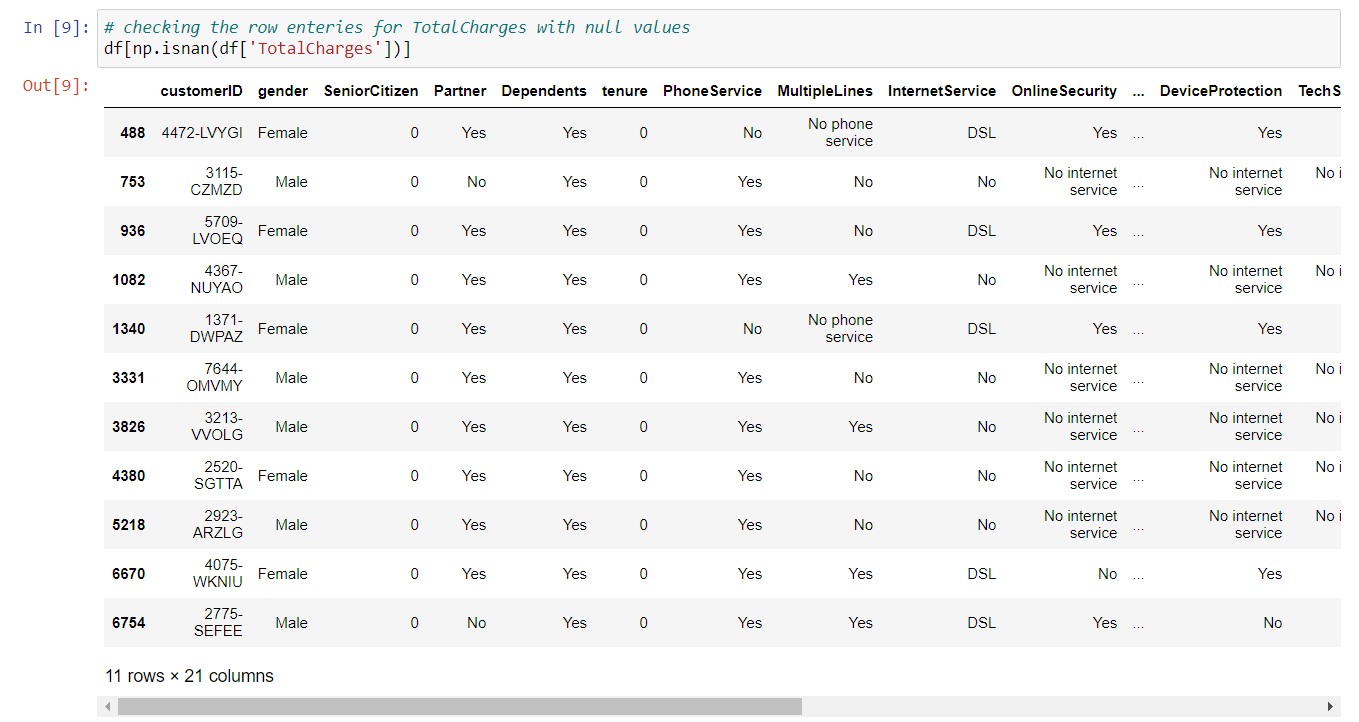
# using describe function to describe the numerical values in the dataset  
df.describe()

# using the describe function to describe the object values in the dataset  
df.describe(include = ['object'] ) 

#### From the above table we can see that column "TotalCharges" is shown as object but in actual it should be a numerical data. Also, we can see that there are empty enteries in it. So, we will need to first convert it into numerical data then need to treat null values appropriately.

# converting column "TotaclCharges" to numeric then checking the nulls  
df['TotalCharges'] = pd.to\_numeric(df.TotalCharges, errors='coerce')  
df.isnull().sum()Now, we can see that there are 11 null values in column "TotalCharges".

Now, let's check the other columns for these null values.

# checking the row enteries for TotalCharges with null values  
df[np.isnan(df['TotalCharges'])]

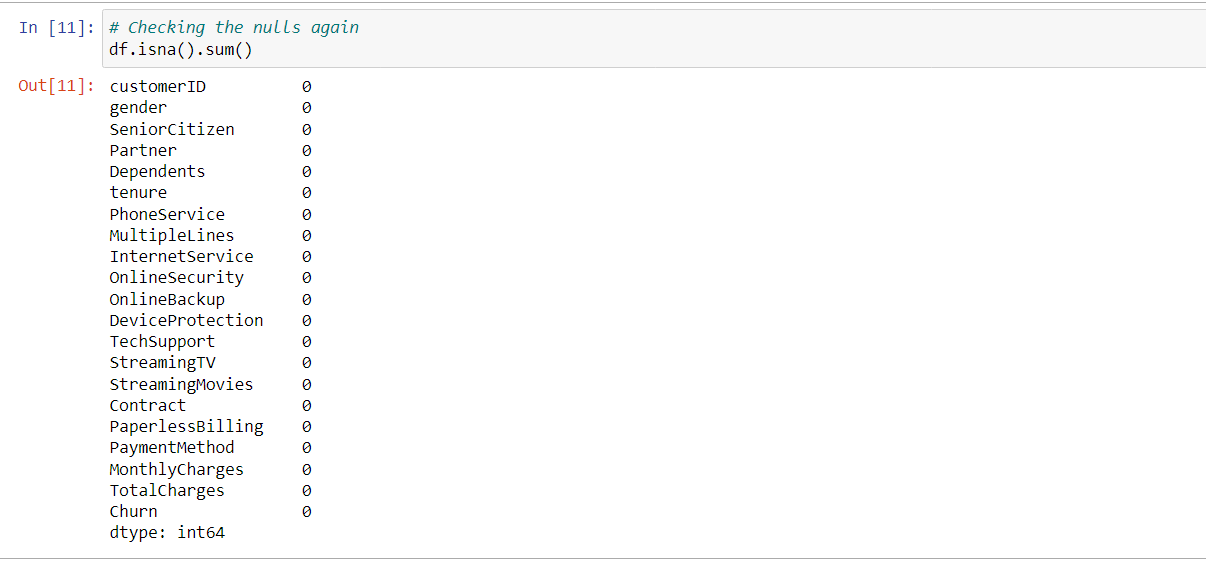
From the above data we can see that column "Tenure" has all 0 as entries and column "Contract" has (10 - Two year and 1 - One year) as enteries for the null values in TotalCharges.

For column "MonthlyCharges" we have different billing values.

Thus, we can infer that these are new customers who have been for less than a month avaialing services with the company.

So, we would need to fill missing values in TotalCharges based on MonthlyCharges as when their 1 month gets completed they will get TotalCharges = MonthlyCharges.

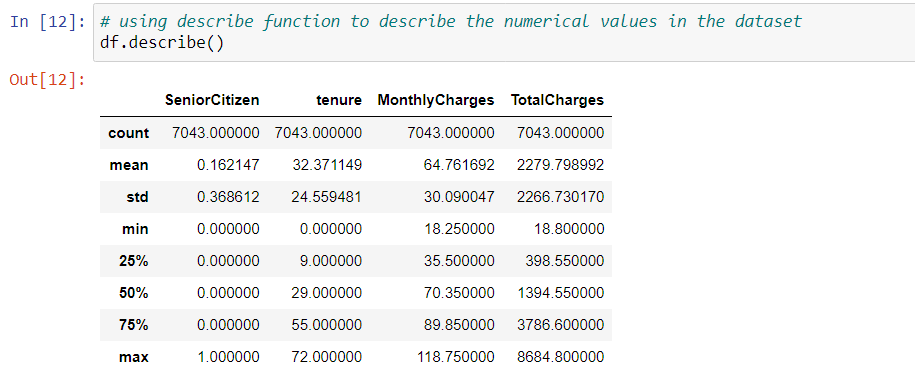
# filling the null values in TotalCharges witht MonthlyCharges as the tenure is less than 1.  
df['TotalCharges'] = df['TotalCharges'].fillna(df['MonthlyCharges'])

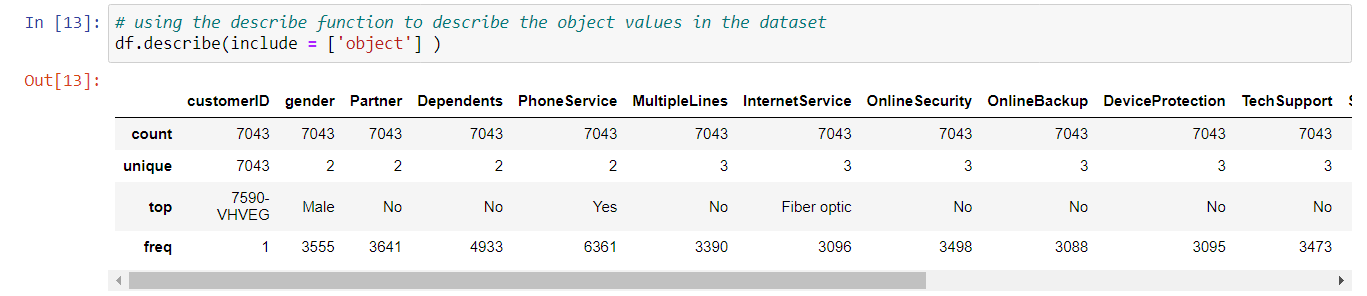
# Checking the nulls again  
df.isna().sum()

#### Now, we have no null values in the dataset.

#### So, let's see some trends in our dataset.

# using describe function to describe the numerical values in the dataset  
df.describe()



# using the describe function to describe the object values in the dataset  
df.describe(include = ['object'] ) 

# First let's drop customerID as it's not related to churn in any way.  
df.drop('customerID', axis = 1, inplace=True)

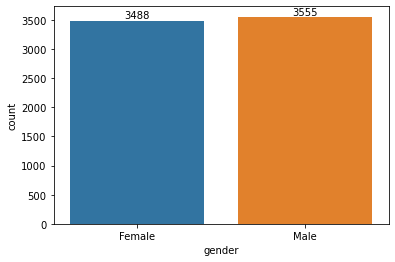
### Data Visualization for key insights

The goal here is to understand what is in our data set and how different variables are related to each other. For example, we might want to know how many customers are male or female, how old they are, and where they live (based on their IP address). This can be done using different visualization techniques such as charts, histograms, scatter plots, etc.

# importing pandasql for data manipulation  
from pandasql import sqldf

The visualization for different types of columns categorical and numerical is shown below. Firstly, showing for categorical data.  
Showing command lines for a single column. Similarly, we can visualize for other columns as well.

### Column - Gender

# plotting graph for gender count  
a = sns.countplot(x = 'gender', data = df)  
for c in a.containers:  
 labels = [f'{int((v.get\_height()))}' for v in c]  
 a.bar\_label(c, labels=labels, label\_type='edge')

#### The distribution of male and female categories is approx. equal. Male have higher count if we look exact values.

##### Now Let's see how is the churning within each category of the Gender.

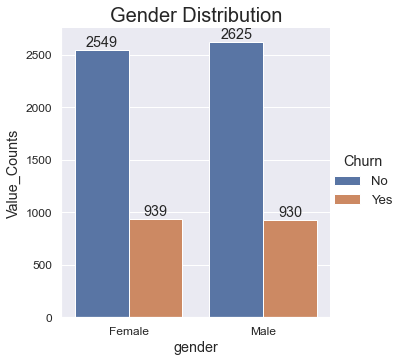
# Selecting the required columns for visualization  
df1 = df[['gender', 'Churn']]

# finding the value counts for each category that churned or not  
q = 'SELECT \*, COUNT(\*) AS Value\_Counts FROM df1 GROUP BY gender, churn';  
df\_temp = sqldf(q)

df\_temp

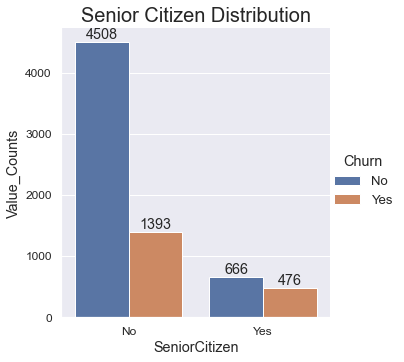
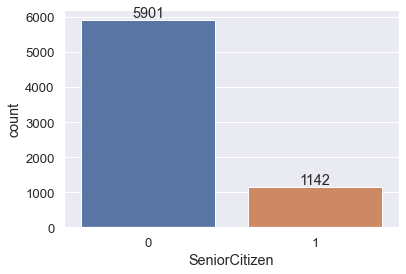
gender Churn Value\_Counts  
0 Female No 2549  
1 Female Yes 939  
2 Male No 2625  
3 Male Yes 930

# plotting the category-wise distribution for each category  
sns.set(font\_scale = 1.2)  
a = sns.catplot(x = 'gender', y = 'Value\_Counts', data = df\_temp, kind = 'bar', hue = 'Churn')  
plt.title('Gender Distribution', fontsize=20)  
plt.tick\_params(labelsize=12)  
for c in a.ax.containers:  
 labels = [f'{int((v.get\_height()))}' for v in c]  
 a.ax.bar\_label(c, labels=labels, label\_type='edge')  
plt.show()



#### From the above graph we can see that females have churned more than males if we look at exact values. While it is almost equal in both categories.

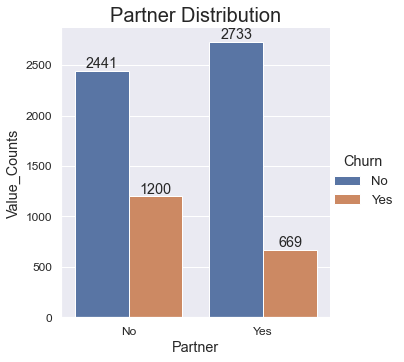
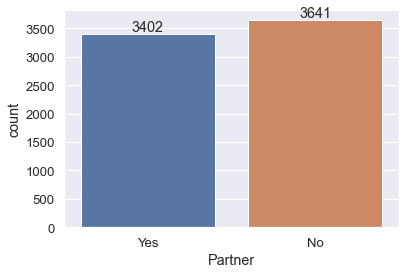
### Column - Senior Citizen



#### Client base seems to be highly comprising of Young Citizen.

#### Senior Citizens have churned more if we talk about proportions in each category - Senior or Young. And, according to exact values Young Citizens have churned more since their value count is very high.

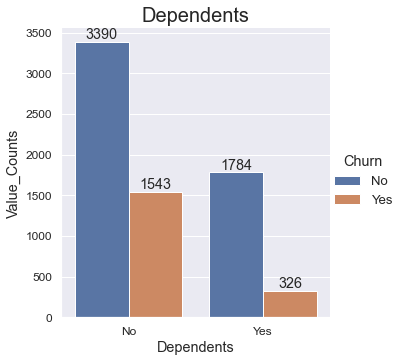
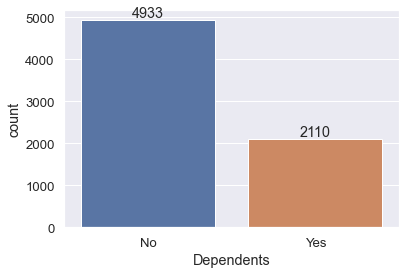
### Column - Partner



#### Customers who have partner are less than the customers who are without partners.

#### Customers who do not have partner have churned more than the customers who have partner.

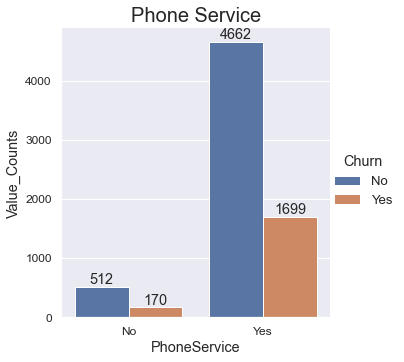
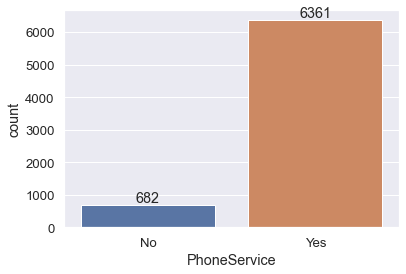
### Column - Dependents



#### Customer who are Independents comprises of almost twice of the number of dependent customers.

#### Customers who are independents have churned more than the customers who are dependents.

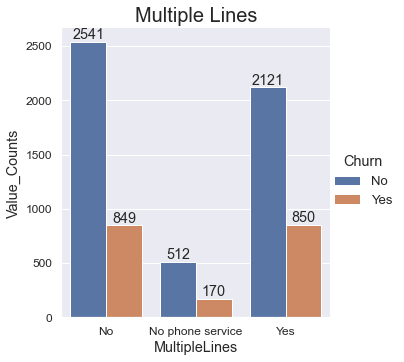
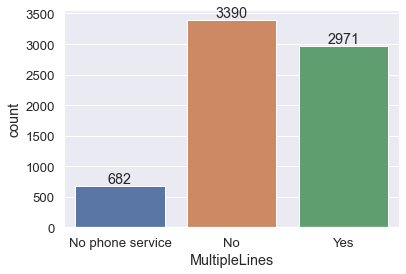
### Column - Phone Services



#### Almost 90% of customers have opted for Phone Services.

#### The customers who have opted Phone Services have churned more than the customers who have not opted Phone Services.

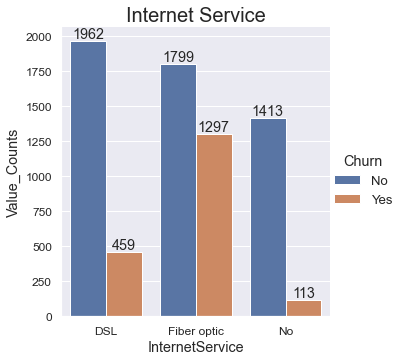
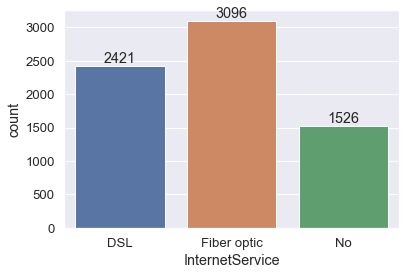
### Column - Multiple Lines



#### For Multiple Lines to be active, customer should have opted for Phone Services first. Among Customers who have opted Phone Services, mostly customers have not opted for Multiple Lines. But the customers who have opted for Multiple Lines are not so far behind.

#### Although Churning for customers with multiple lines seems to be equal for customers with or without multiple lines but proportionally we can see that customers who have Multi Lines have churned more.

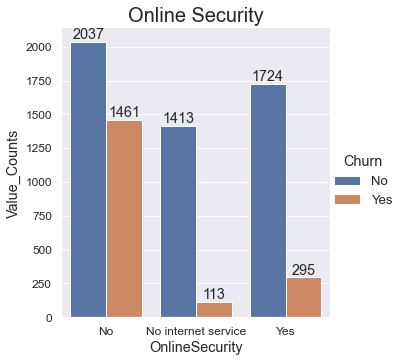
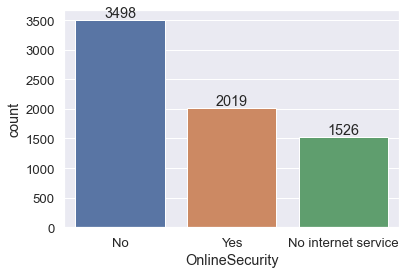
### Column - Internet Services



#### Most of the customers have opted for Fiber Optic Internet Services followed by DSL services.

#### Customers who opted for Fiber Optic Internet Service have churned most.

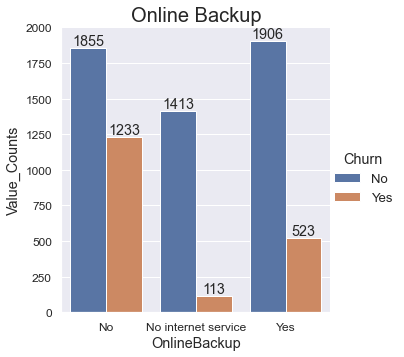
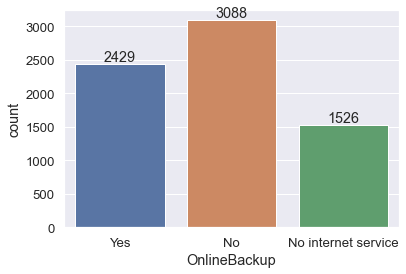
### Column - Online Security



#### Most of the customers do not have service access to OnlineSecurity who have opted for Internet Services.

#### Among the customers who have active Internet Services large customers have churned who did not have Online Security. Customers without Internet Services and without Online Security have churned the least.

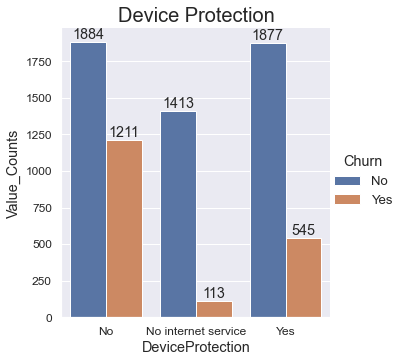
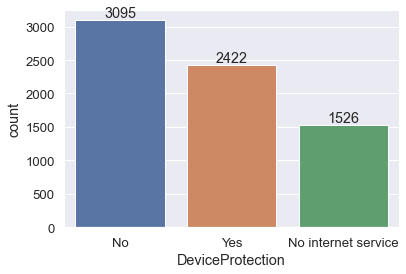
### Column - Online Backup



#### Among the customers who have opted for Internet Services, customers who do not have service access to Online Backup are greater in numbers compared to customers with Online Backup.

#### Among the customers who have active Internet Services significant amount of customers have churned who did not have Online Backup. Customers without Internet Services and without Online Backup have churned the least.

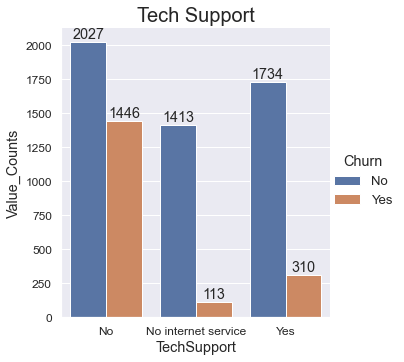
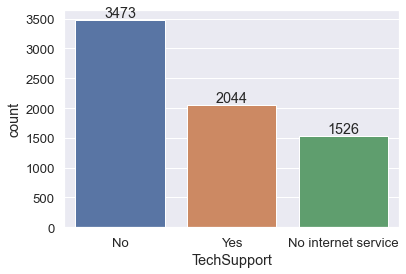
### Column - Device Protection



#### Most of the customers do not have service access to Device Protection who have opted for Internet Services.

#### Among the customers who have active Internet Services significant amount of customers have churned who did not have Device Protection. Customers without Internet Services and without Device Protection have churned the least.

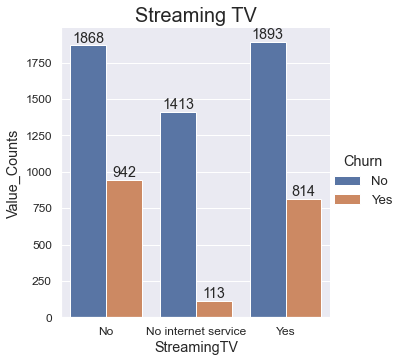
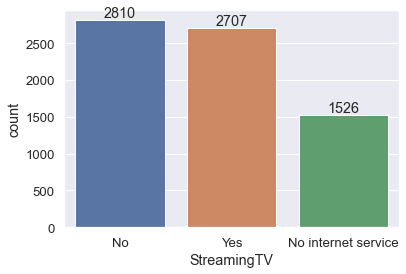
### Column - Tech Support



#### Most of the customers do not have service access to Tech Support who have opted for Internet Services.

#### Among the customers who have active Internet Services large customers have churned who did not have Tech Support.

### Column - StreamingTV



#### Customers are approx equally distributed for Streaming TV service among the customers who have opted for Internet Services. But if we consider actual number then we can say that the cutomers without Streaming TV services are more.

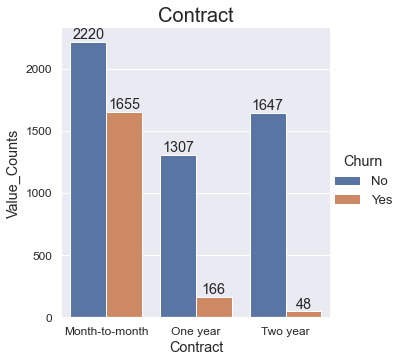
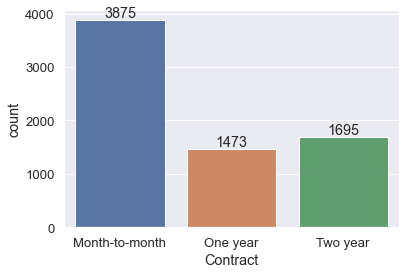
#### Among the customers who have active Internet Services, although the customers without Streaming TV services churned more but the difference for the churned customers with or without Streaming TV service is not so high. Customers without Internet Services and without Online Security have churned the least.

### Column - Streaming Movies

#### Customers are almost equally distributed for Streaming Movies service among the customers who have opted for Internet Services. But if we consider actual number then we can say that the cutomers without Streaming Movies service are more.

#### Among the customers who have active Internet Services, although the customers without Streaming Movies service churned more but the difference for the churned customers with and without Streaming TV service is not so high. Customers without Internet Services and without Online Security have churned the least.

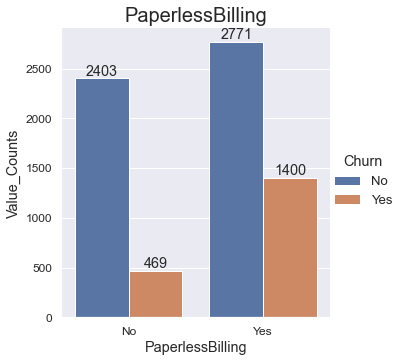
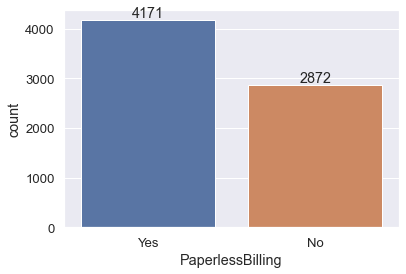
### Column - Contract



#### Most of the customers have opted for month-to-month Contract followed by Two Year contract with a large gap and then One Year contract with small gap.

#### Very high number of customers churned who had month-to-month contract. Least number of customers churned who have Two year contract.

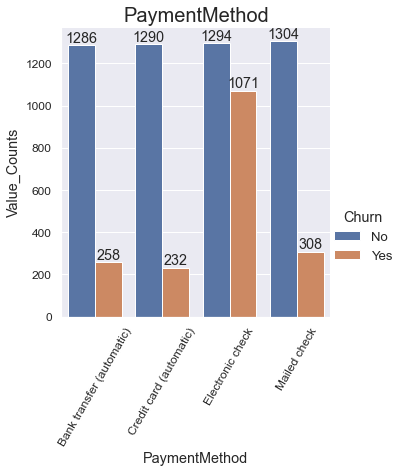
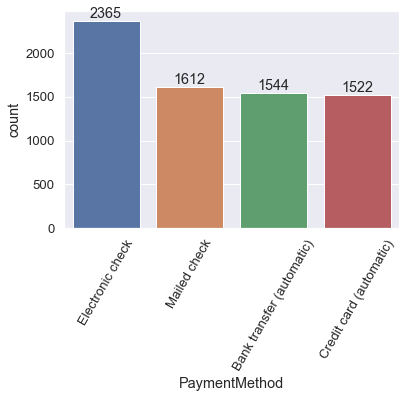
### Column - Paperless Billing



#### More number of customers have Paperless Billing.

#### Most number of customers have churned who had Paperless Billing.

### Column - Payment Method



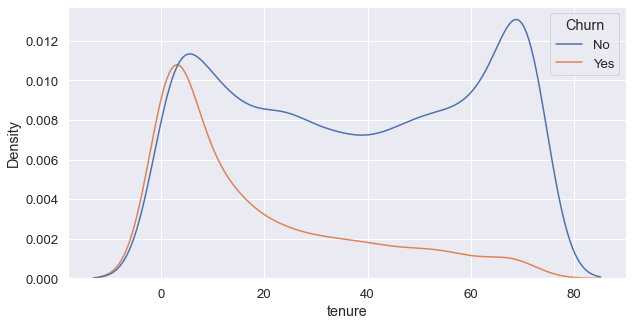
#### Most of the customers pay through Electronic Check followed by Mailed Check. Customers payment through Bank Transfer (automatic) and Credit Card (automatic) are almost equal in number.

#### Customer who make payment through Electronic Check churned most with a huge difference. Customers who pay by Credit Card (automatic) churned least.

Now, let’s see visualization for numerical columns.

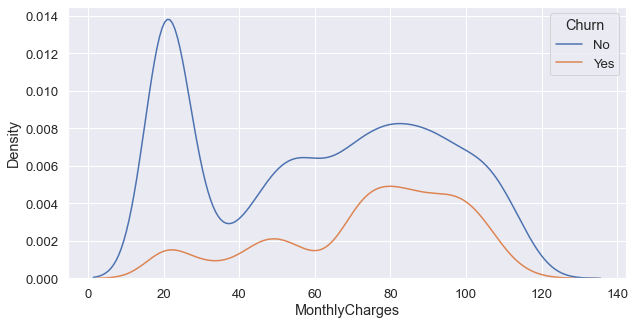
### Column - Tenure

# plotting graph for tenure based on churn - "Yes" or "No"  
plt.figure(figsize=(10, 5))  
sns.kdeplot(x = 'tenure', data = df, hue = 'Churn')  
plt.show()



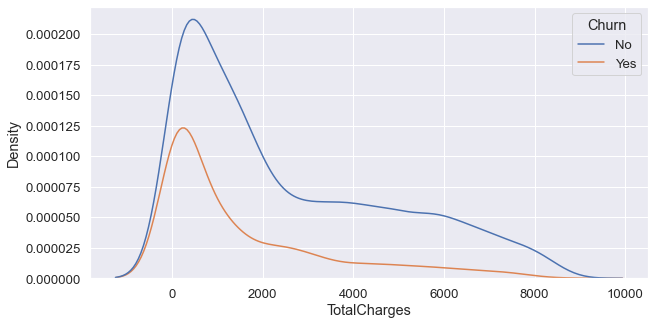
#### From the above graph we can conclude that customers are more likely to churn within first 5 years then their churing drops significantly at a tenure of 20 years.

### Column - Monthly Charges



#### From the above graph we can conclude that customers have churned most if their monthly charges are on a higher side. This monthly charges range in which customers are likely to churn lies typically from 70 to 105.

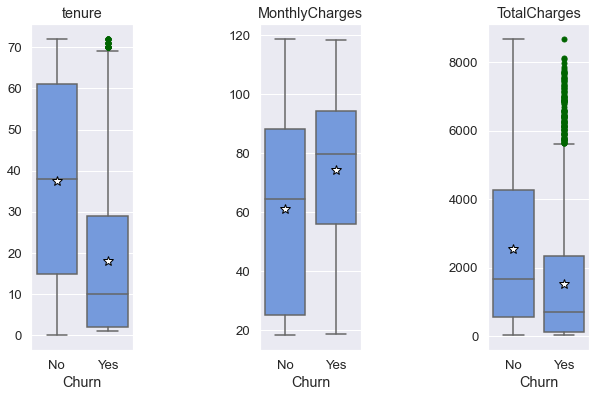
### Column - Total Charges



#### The customers with Total Charges lower than 1500 have churned the most. Churning reduces as the Total Charges for a customer increases.

##### Let's confirm these findings through boxplot as well.

# creating a temporary dataframe to plot boxplot  
tem\_df = df[['tenure', 'MonthlyCharges', 'TotalCharges', 'Churn']]

# Determine the number of columns  
num\_inputs = len(list(tem\_df.columns)) - 1  
  
# Create a figure with 1 row, and num\_inputs long  
fig, axs = plt.subplots(1, num\_inputs, figsize=(10,6))  
  
# Use a for loop to loop over each column in the dataframe and   
# create a separate box plot  
for i, (ax, curve) in enumerate(zip(axs.flat, list(tem\_df.columns))):  
 sns.boxplot(x = tem\_df['Churn'], y=tem\_df[curve], ax=ax, color='cornflowerblue', showmeans=True,   
 meanprops={"marker":"\*",  
 "markerfacecolor":"white",   
 "markeredgecolor":"black",  
 "markersize":"10"},  
 flierprops={'marker':'o',   
 'markerfacecolor':'darkgreen',  
 'markeredgecolor':'darkgreen'})  
   
 ax.set\_title(list(tem\_df.columns)[i])  
 ax.set\_ylabel('')  
   
plt.subplots\_adjust(hspace=0.15, wspace=1.25)  
plt.show()  


#### From the boxplot we can clearly see that Customers churned when the MonthlyCharges were high, when the tenure was short. TotalCharges is dependent on the tenure and monthlyCharges. So, we can say that for a longer tenure the TotalCharges would also be higher.

# EDA Concluding Remark

* Singles and independents tend to churn more.
* The customer with fiber optics internet churn more that may be due to many reasons like connectivity issues, price plans, service or support.
* Among the customers who have internet services, the customers who did not have extra services like Online Security, Online Backup, Device Protection, Tech Support have churned more.
* Customers having Internet with Streaming services like Streaming TV, Streaming Movies do not show much difference of churned customers.
* Customers with month-to-month contract have high chances of churning.
* Customers with short tenure have high chance of churning.
* For a high monthly billing, customers tend to churn more.

# Pre-Processing Pipeline

The next step in our data analysis is pre-processing, which involves cleaning and transforming the raw data so that it is ready for further analysis and prediction.

Data cleaning includes removing non-essential information like zeros and missing values, as well as reducing duplicate records (for example, if you have two customers with exactly the same order history).

Data transformation can be done using different techniques, such as:

scaling (for example, converting your data from Fahrenheit to Celsius),

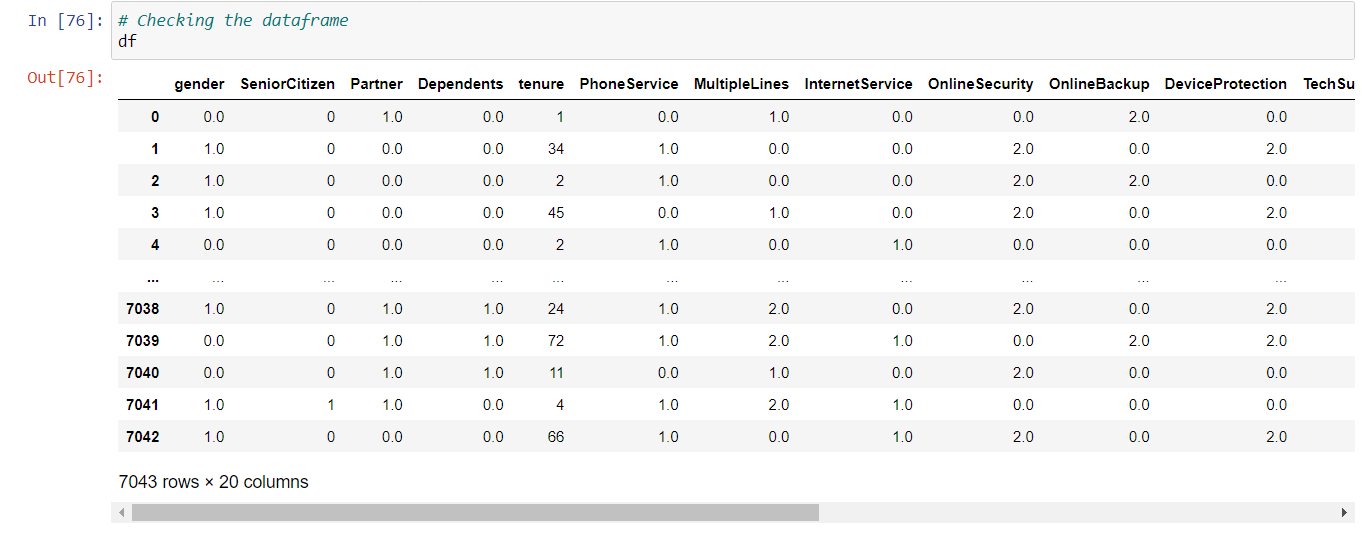
normalization (for example, converting a categorical variable with many levels into one with only two levels), and binning (which involves grouping continuous variables into discrete bins).

## Data Preprocessing

# Separating categorical columns from numerical  
num\_col = ['tenure', 'MonthlyCharges', 'TotalCharges']  
cat\_col = list(set(df.columns) - set(num\_col))

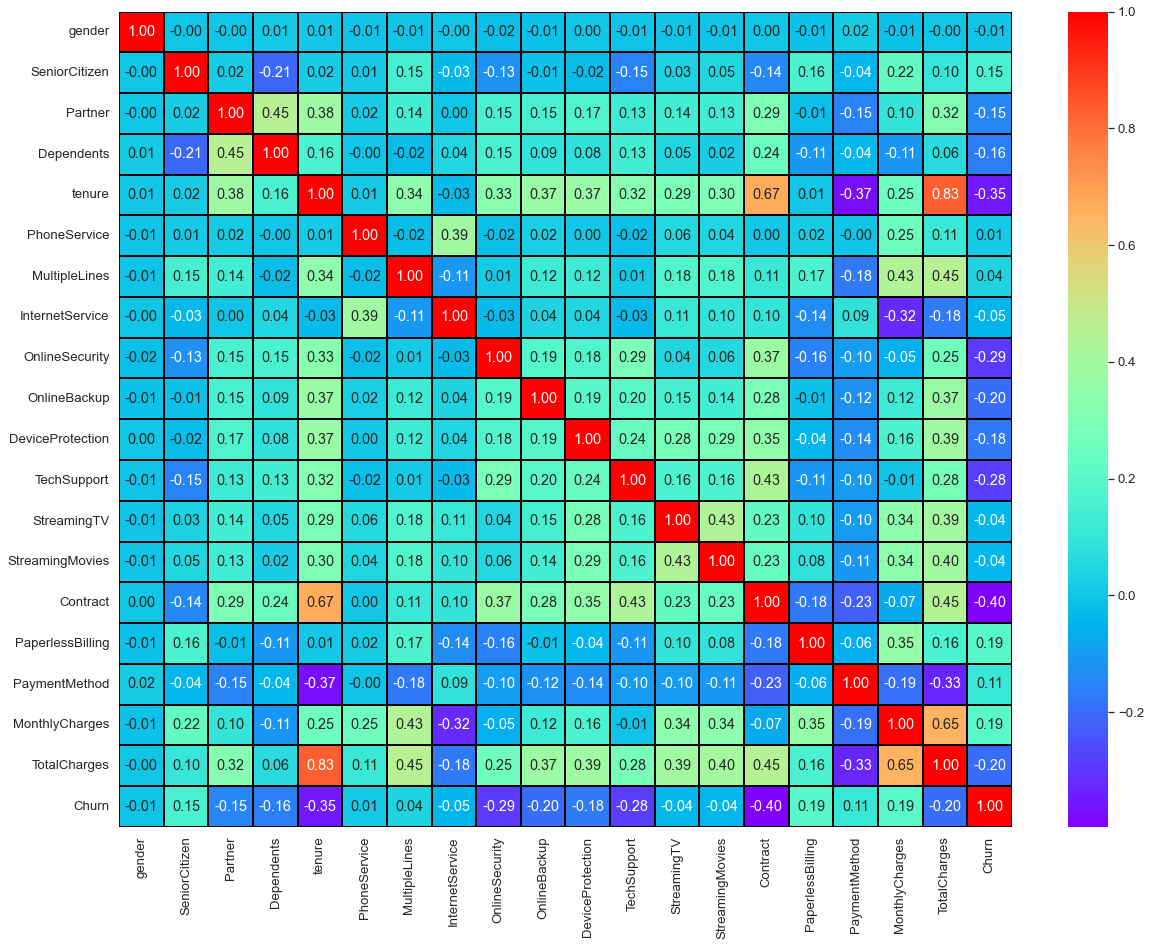
# importing the encoder  
from sklearn.preprocessing import OrdinalEncoder  
enc = OrdinalEncoder()

# encoding the categorical columns  
for i in df.columns:  
 if df[i].dtypes == 'object':  
 df[i]= enc.fit\_transform(df[i].values.reshape(-1,1))

# Checking the dataframe  
df 

### Correlation Heatmap

# plotting the correlation heatmap  
plt.figure(figsize=(20,15))  
sns.heatmap(df.corr(), annot = True, linewidths = 0.1, linecolor = 'black', fmt = '0.2f', cmap = 'rainbow')  
plt.show()



* **TotalCharges has very high correlation with tenure and high correlation with MonthlyCharges.**
* **Contract has high correlation with tenure.**

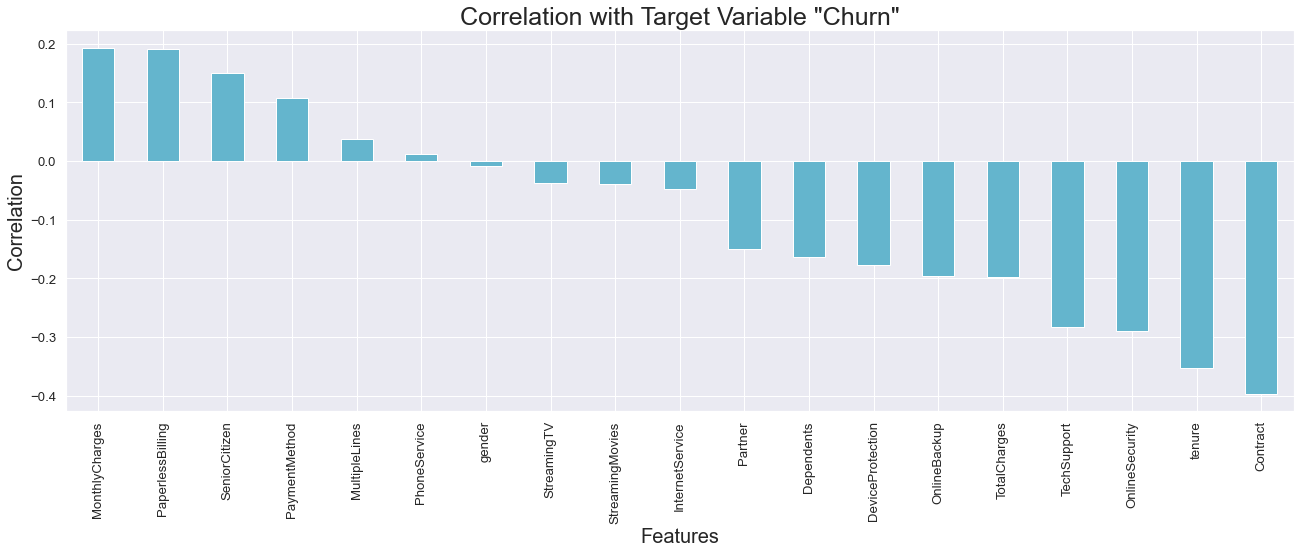
### Correlation with the target variable

# finding correlation with the target variable "Churn" and excluding "Churn" column in result  
df.corr()['Churn'].sort\_values()[:-1]

Contract -0.396713  
tenure -0.352229  
OnlineSecurity -0.289309  
TechSupport -0.282492  
TotalCharges -0.198347  
OnlineBackup -0.195525  
DeviceProtection -0.178134  
Dependents -0.164221  
Partner -0.150448  
InternetService -0.047291  
StreamingMovies -0.038492  
StreamingTV -0.036581  
gender -0.008612  
PhoneService 0.011942  
MultipleLines 0.038037  
PaymentMethod 0.107062  
SeniorCitizen 0.150889  
PaperlessBilling 0.191825  
MonthlyCharges 0.193356  
Name: Churn, dtype: float64

#### Visualizing the negative and positive related columns.

# visualizing the correlations using bar plot  
plt.figure(figsize=(22,7))  
df.corr()['Churn'].sort\_values(ascending=False).drop(['Churn']).plot(kind='bar', color = 'c')  
plt.xlabel('Features', fontsize = 20)  
plt.ylabel('Correlation', fontsize = 20)  
plt.title('Correlation with Target Variable "Churn"',fontsize=25)  
plt.show()



### Checking for the Skewness

# checking the skewness in the data  
skewness = pd.DataFrame(df.skew(),columns=['skewness'])

#### Keeping the range of (0.5, -0.5) as optimal range for skeness range.

# Selecting the columns outside skewness range.  
skewness[(skewness['skewness'] > 0.5) | (skewness['skewness'] < -0.5)]

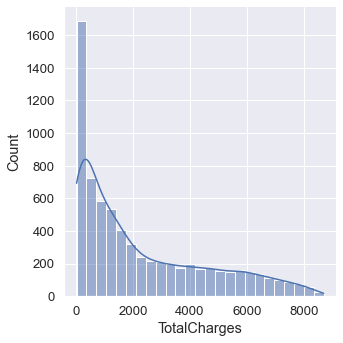
skewness  
SeniorCitizen 1.833633  
Dependents 0.875199  
PhoneService -2.727153  
Contract 0.630959  
TotalCharges 0.963316  
Churn 1.063031

#### Since the columns "Senior Citizen, Dependents, Phone Service, Contract" are Categorical Columns. Therefore, no need for removing skewness.

#### Column "Churn" is a target variable so for it also skewness removal not required.

#### Column "Total Charges" is the only column for which we would remove skewness.

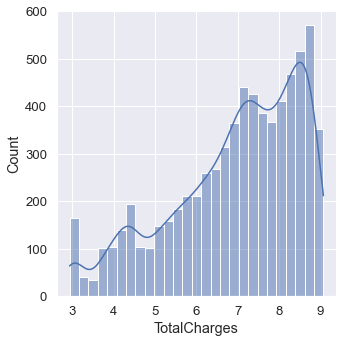
# checking the plot  
sns.displot(df['TotalCharges'], kde = True)



### Checking Log Transformation

# removing the skewness using log transformation  
df1['TotalCharges'] = np.log(df['TotalCharges'])

# checking the skewness in plot  
sns.displot(df1['TotalCharges'], kde = True)



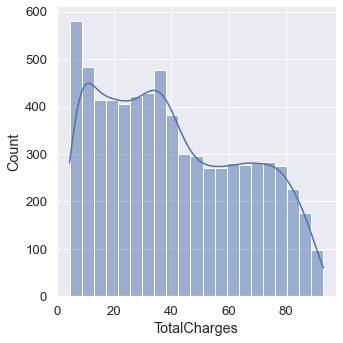
# checking the skewness  
skewness = pd.DataFrame(df1.skew(),columns=['skewness'])  
  
skewness.loc["TotalCharges"]

skewness -0.752664  
Name: TotalCharges, dtype: float64

#### From the graph and skew value we can see this did not work. So, let's try Square root transformation.

# removing the skewness using square root transformation  
df1['TotalCharges'] = np.sqrt(df['TotalCharges'])

# checking the skewness in plot  
sns.displot(df1['TotalCharges'], kde = True)



# checking the skewness  
skewness = pd.DataFrame(df1.skew(),columns=['skewness'])  
  
skewness.loc["TotalCharges"]

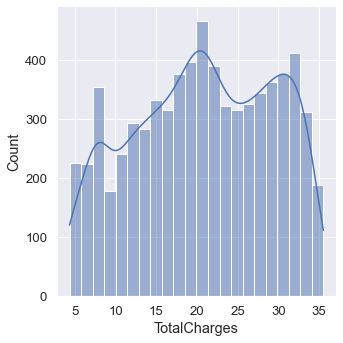
skewness 0.309874  
Name: TotalCharges, dtype: float64

#### From the graph and skew value, we can see that skewness is within permissible range. We try Box Cox transformation as well to see how it affects the data.

# importing necessary library  
from scipy import stats

# removing the skewness using Box-Cox transformation  
df1['TotalCharges'] = stats.boxcox(df['TotalCharges'])[0]

# checking the skewness in plot  
sns.displot(df1['TotalCharges'], kde = True)



# checking the skewness  
skewness = pd.DataFrame(df1.skew(),columns=['skewness'])  
  
skewness.loc["TotalCharges"]

skewness -0.146201  
Name: TotalCharges, dtype: float64

#### Box-Cox transformation has completely removed the skewness comparing it to the original skew value = 0.963316. Now, it is at -0.146201

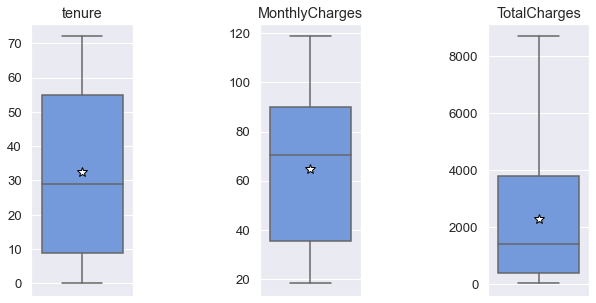
#### Since the square root transforms data within skew range. Therefore, going with square root transformation.

#### If the model score is not fair enough, then would try with Box-Cox transformation.

### Outliers Check

# creating a temporary dataframe with necessary columns  
tem\_df = df[['tenure', 'MonthlyCharges', 'TotalCharges']]

# Determine the number of columns to be plotted  
num\_inputs = len(list(tem\_df.columns))  
  
# Create a figure with 1 row, and num\_inputs long  
fig, axs = plt.subplots(1, num\_inputs, figsize=(10,5))  
  
# Use a for loop to loop over each column in the dataframe and   
# create a separate box plot  
for i, (ax, curve) in enumerate(zip(axs.flat, list(tem\_df.columns))):  
 sns.boxplot(y=tem\_df[curve], ax=ax, color='cornflowerblue', showmeans=True,   
 meanprops={"marker":"\*",  
 "markerfacecolor":"white",   
 "markeredgecolor":"black",  
 "markersize":"10"},  
 flierprops={'marker':'o',   
 'markerfacecolor':'darkgreen',  
 'markeredgecolor':'darkgreen'})  
   
 ax.set\_title(list(tem\_df.columns)[i])  
 ax.set\_ylabel('')  
   
plt.subplots\_adjust(hspace=0.15, wspace=1.25)  
plt.show()



#### From the above box plot we conclude that there seems to be no outliers in the dataset.

# Building Machine Learning Models

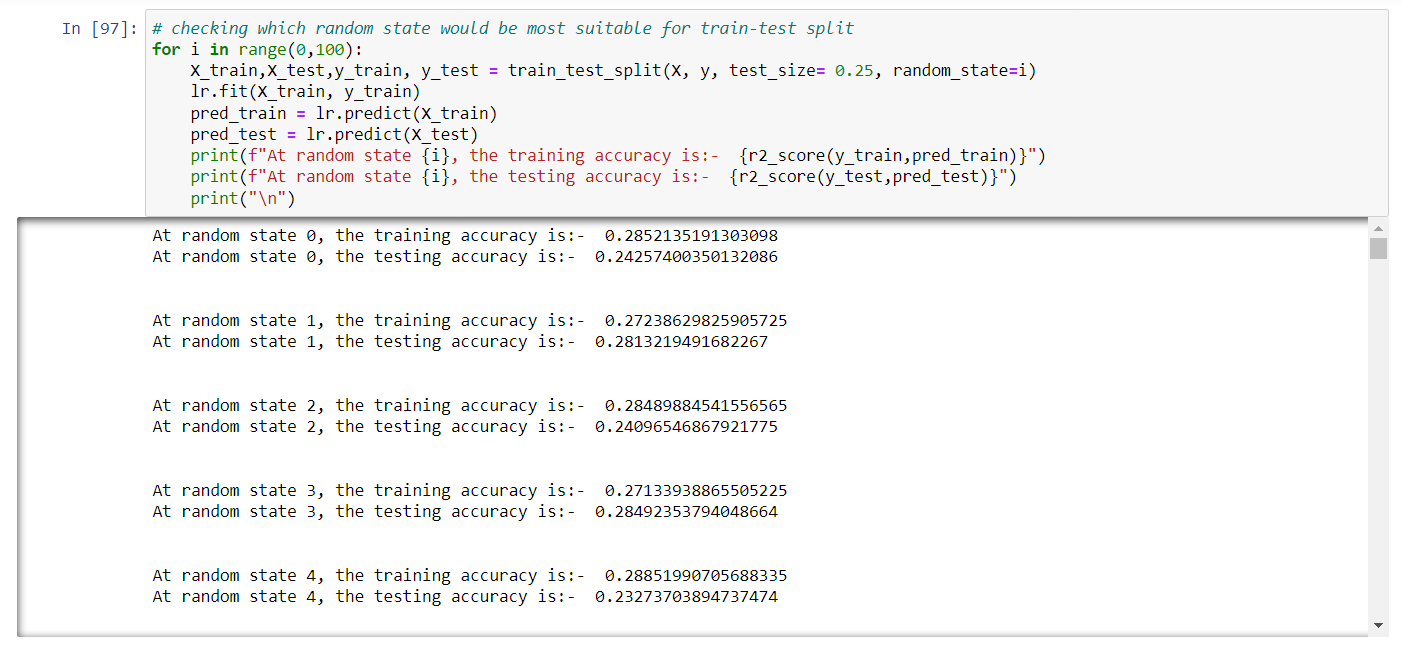
In this section, we will cover how to build machine learning models, as well as test and tune them. We will also discuss how to deploy our model in production.

### Separating the columns into features and target:

# importing necessary libraries  
from sklearn.preprocessing import MinMaxScaler  
from sklearn.linear\_model import LinearRegression  
lr = LinearRegression()  
import sklearn.metrics as metrics  
from sklearn.metrics import r2\_score, accuracy\_score, classification\_report, roc\_auc\_score, roc\_curve, RocCurveDisplay  
from sklearn.model\_selection import train\_test\_split  
from sklearn.model\_selection import cross\_val\_score  
from sklearn.metrics import confusion\_matrix

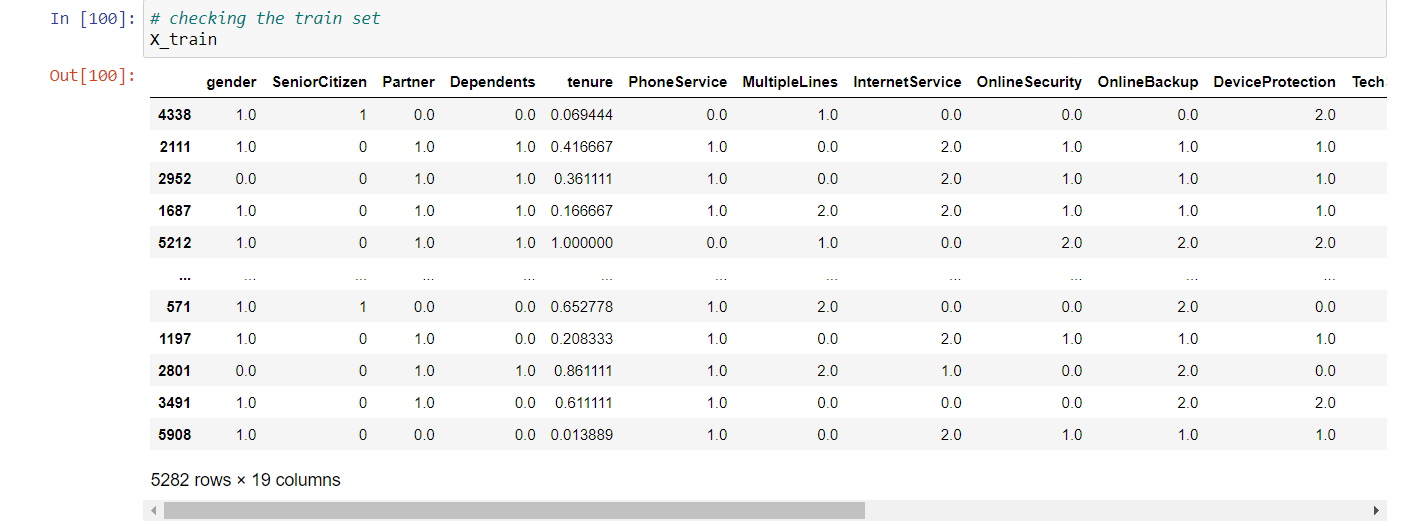
# Let's split our dataset into independent (X) and dependent (y) variables  
X = df.drop('Churn', axis = 1)  
y = df.Churn

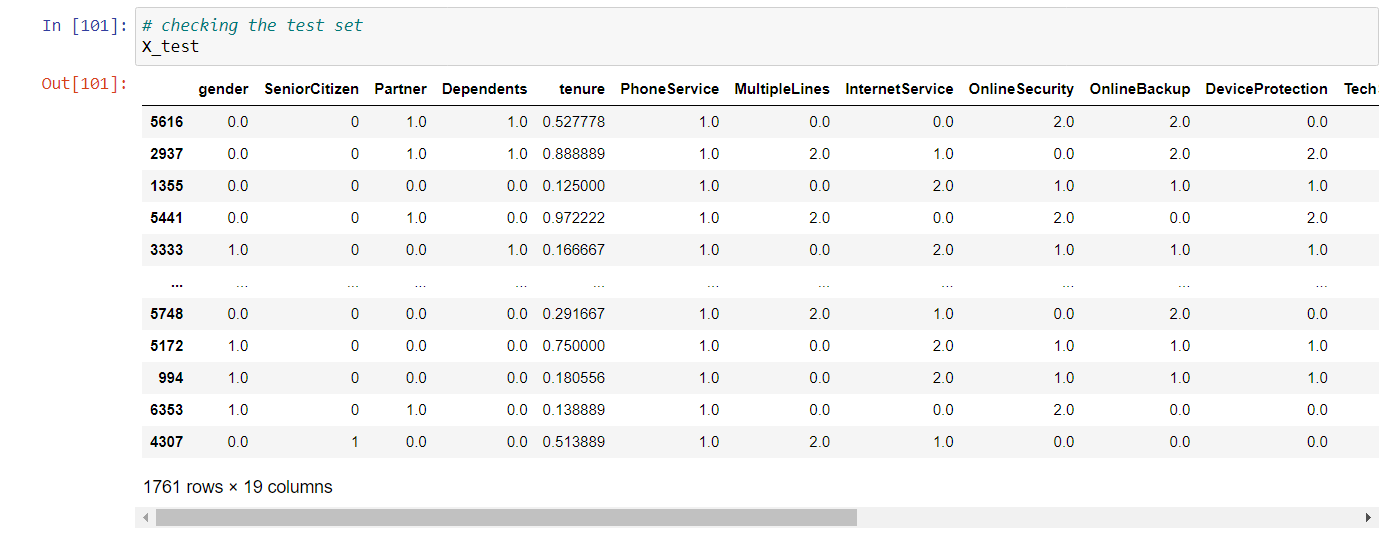
# checking which random state would be most suitable for train-test split  
for i in range(0,100):  
 X\_train,X\_test,y\_train, y\_test = train\_test\_split(X, y, test\_size= 0.25, random\_state=i)  
 lr.fit(X\_train, y\_train)  
 pred\_train = lr.predict(X\_train)  
 pred\_test = lr.predict(X\_test)  
 print(f"At random state {i}, the training accuracy is:- {r2\_score(y\_train,pred\_train)}")  
 print(f"At random state {i}, the testing accuracy is:- {r2\_score(y\_test,pred\_test)}")  
 print("\n")



# we use random state 13 since the train and test accuracy are the closest.  
X\_train,X\_test,y\_train, y\_test = train\_test\_split(X, y, test\_size= 0.25, random\_state=44)

# Let's standardize tenure, MonthlyCharges and TotalCharges  
num\_col = ['tenure', 'MonthlyCharges', 'TotalCharges']  
scaler = MinMaxScaler()  
  
X\_train[num\_col] = scaler.fit\_transform(X\_train[num\_col])  
X\_test[num\_col] = scaler.transform(X\_test[num\_col])

# checking the train set  
X\_train 

# checking the test set  
X\_test 

# Dealing with imbalanced data   
y\_train.value\_counts()

0.0 3876  
1.0 1406  
Name: Churn, dtype: int64

# importing necessary library for oversampling  
from imblearn.over\_sampling import SMOTE

# oversampling using SMOTE  
X\_train, y\_train = SMOTE().fit\_resample(X\_train, y\_train)

# checking the data is balanced now  
y\_train.value\_counts()

0.0 3876  
1.0 3876  
Name: Churn, dtype: int64

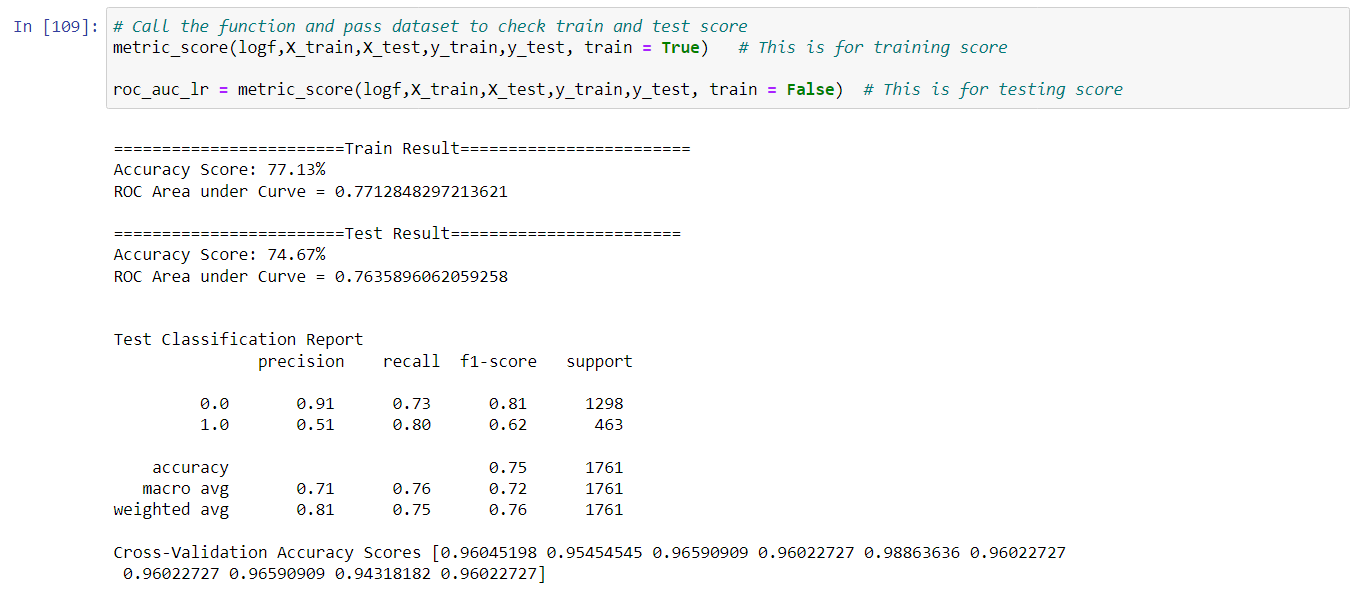
Now, both the classes have balanced.

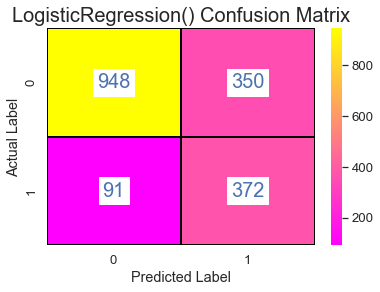
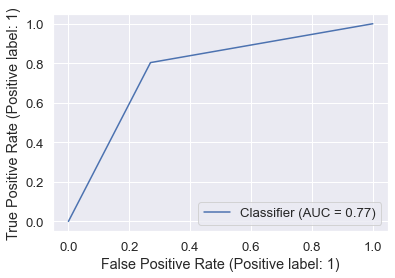
# Write one function and call as many as times to check different performance parameters for different models  
def metric\_score(clf, X\_train,X\_test,y\_train,y\_test, train = True):  
   
 if train:  
 y\_pred = clf.predict(X\_train)  
 roc\_auc = roc\_auc\_score(y\_train, y\_pred)  
 print("\n========================Train Result========================")  
 print(f"Accuracy Score: {accuracy\_score(y\_train, y\_pred) \* 100:.2f}%")  
 print("ROC Area under Curve = {}".format(roc\_auc))  
   
 elif train == False:  
 pred = clf.predict(X\_test)  
 roc\_auc = roc\_auc\_score(y\_test, pred)  
 print("\n========================Test Result========================")  
 print(f"Accuracy Score: {accuracy\_score(y\_test, pred) \* 100:.2f}%")  
 print("ROC Area under Curve = {}".format(roc\_auc))  
 print ('\n\nTest Classification Report\n', classification\_report(y\_test, pred, digits = 2))  
 scores = cross\_val\_score(clf, X\_test, pred, cv=10)  
 print('Cross-Validation Accuracy Scores', scores)  
 # plotting ROC\_AUC curve  
 RocCurveDisplay.from\_predictions(y\_test, pred)  
   
 # plotting confusion matrix for the test set  
 confusion\_matrix\_ = confusion\_matrix(y\_test, pred)  
 plt.figure(figsize=(6,4))  
 ax = plt.subplot()  
 sns.heatmap(confusion\_matrix\_, annot=True, fmt='g', ax = ax, cmap = 'spring', linewidths = 0.1, linecolor = 'black', annot\_kws={'fontsize': 20, 'color':'b', 'backgroundcolor': 'w'})  
 plt.title('{} Confusion Matrix'.format(clf), fontsize=20)  
 ax.set\_xlabel('Predicted Label')  
 ax.set\_ylabel('Actual Label')  
 plt.show()  
 return roc\_auc

### 1. Logistic Regression

# Running Logistic Regression  
from sklearn.linear\_model import LogisticRegression  
logf = LogisticRegression()  
logf.fit(X\_train, y\_train)

LogisticRegression()

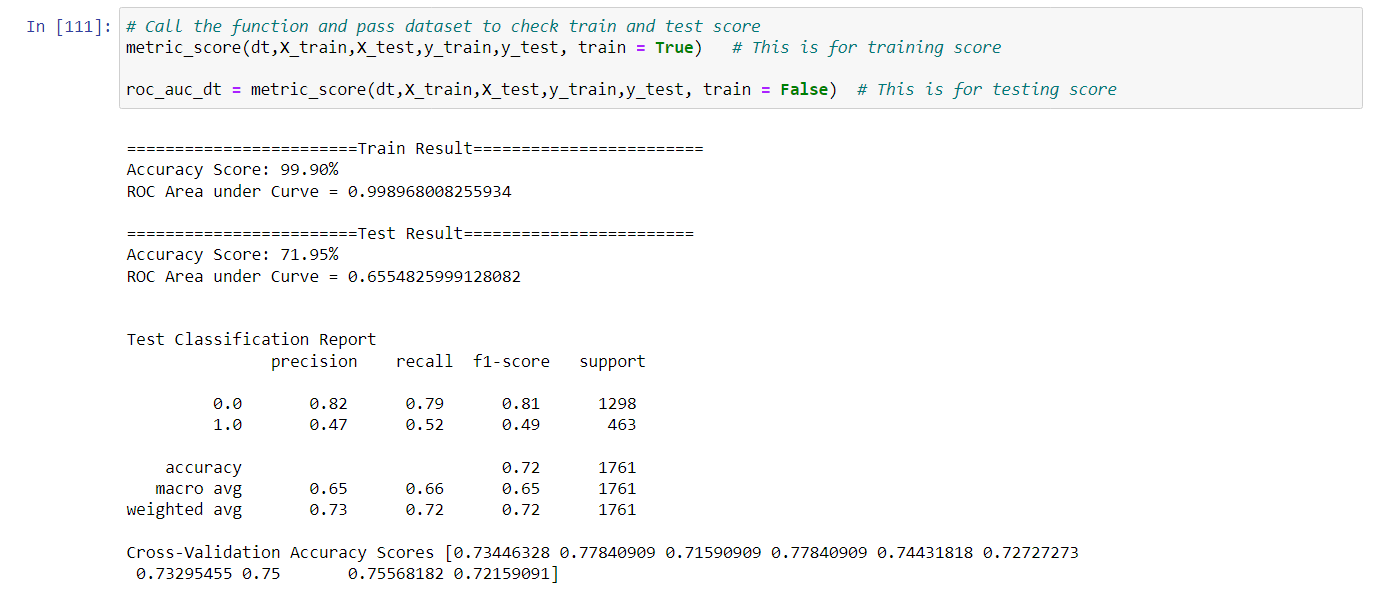
# Call the function and pass dataset to check train and test score  
metric\_score(logf,X\_train,X\_test,y\_train,y\_test, train = True) # This is for training score  
  
roc\_auc\_lr = metric\_score(logf,X\_train,X\_test,y\_train,y\_test, train = False) # This is for testing score  


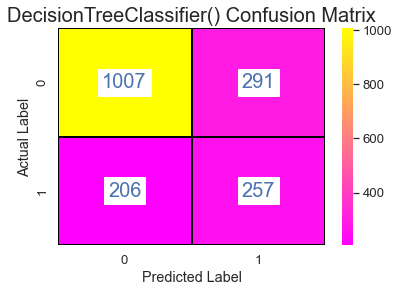


### 2. Decision Tree

# Running Decision Tree  
from sklearn.tree import DecisionTreeClassifier  
  
dt = DecisionTreeClassifier()  
dt.fit(X\_train, y\_train)

DecisionTreeClassifier()

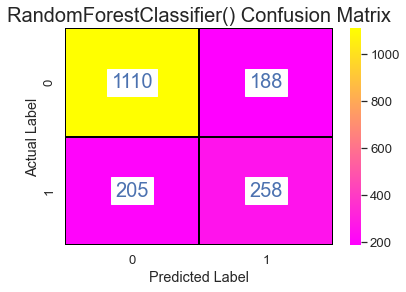
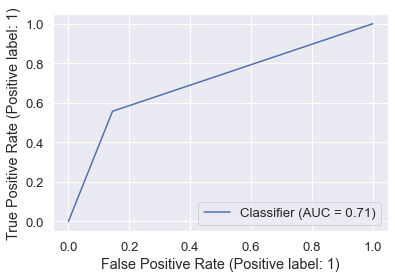
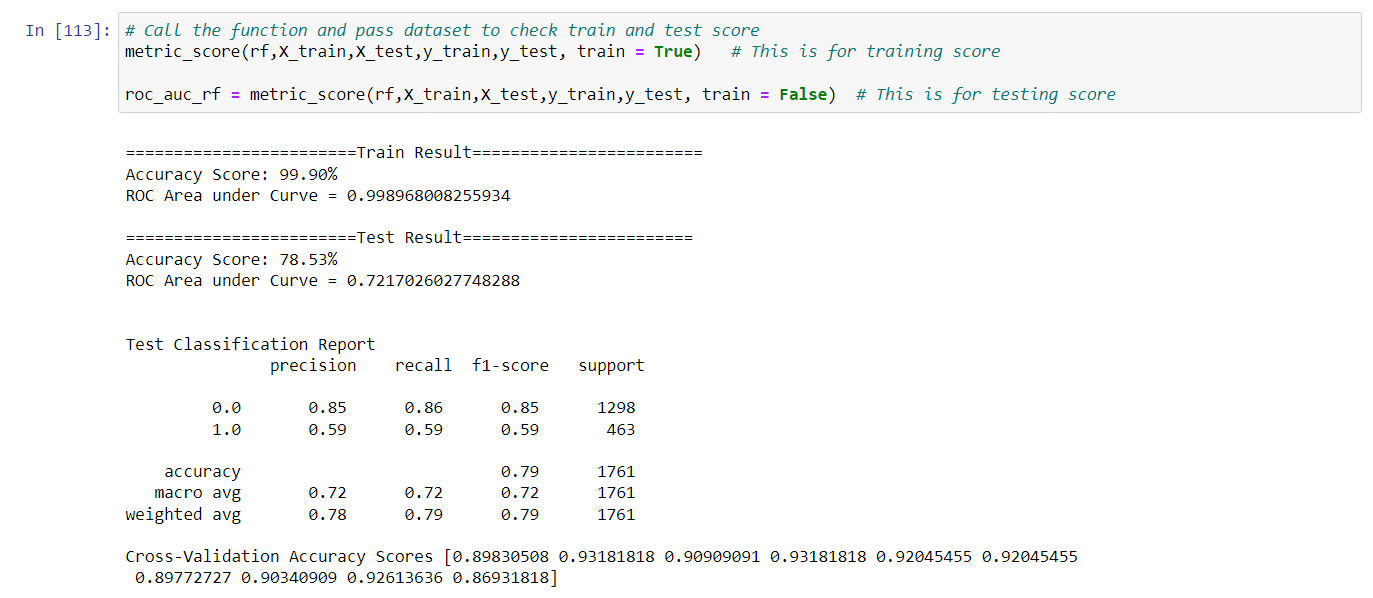
# Call the function and pass dataset to check train and test score  
metric\_score(dt,X\_train,X\_test,y\_train,y\_test, train = True) # This is for training score  
  
roc\_auc\_dt = metric\_score(dt,X\_train,X\_test,y\_train,y\_test, train = False) # This is for testing score 



### 3. Random Forest

# Running Random Forest  
from sklearn.ensemble import RandomForestClassifier  
rf = RandomForestClassifier()  
rf.fit(X\_train, y\_train)

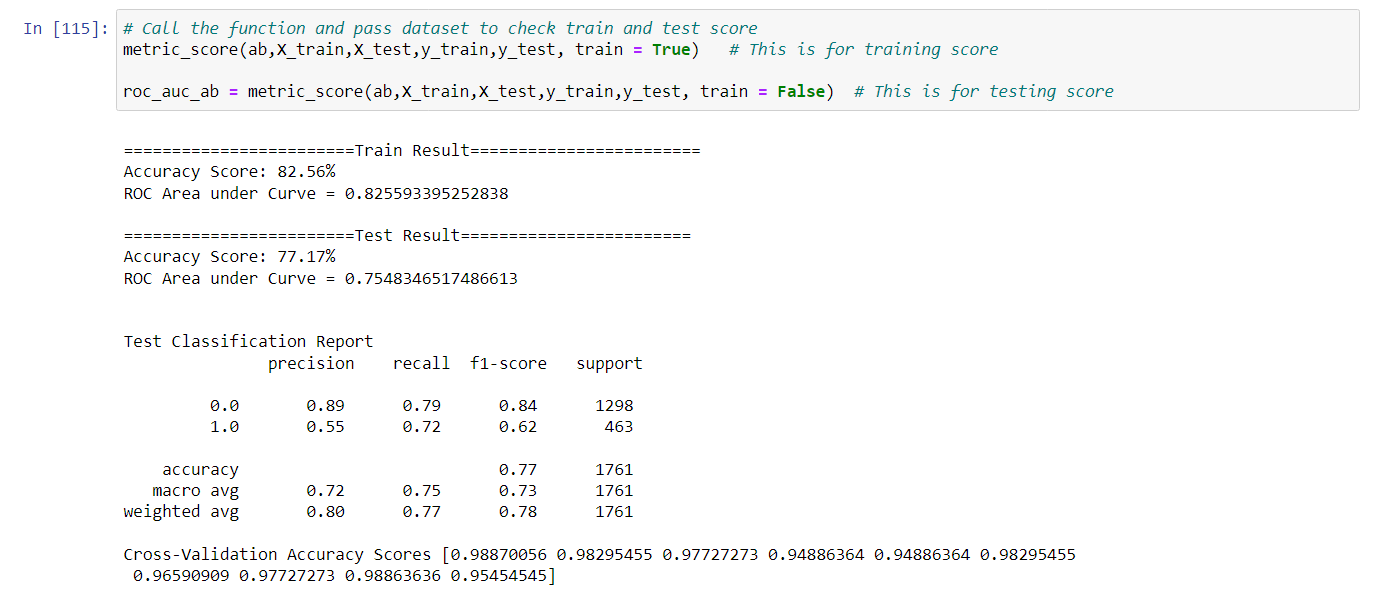
RandomForestClassifier()

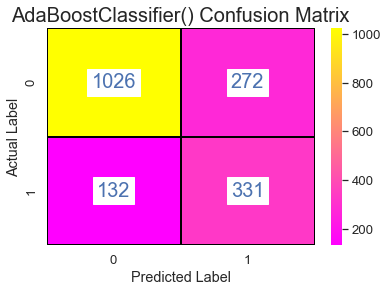
# Call the function and pass dataset to check train and test score  
metric\_score(rf,X\_train,X\_test,y\_train,y\_test, train = True) # This is for training score  
  
roc\_auc\_rf = metric\_score(rf,X\_train,X\_test,y\_train,y\_test, train = False) # This is for testing score 

### 4. Ada Boost

#Running Ada Boost  
from sklearn.ensemble import AdaBoostClassifier  
ab = AdaBoostClassifier()  
ab.fit(X\_train, y\_train)

AdaBoostClassifier()

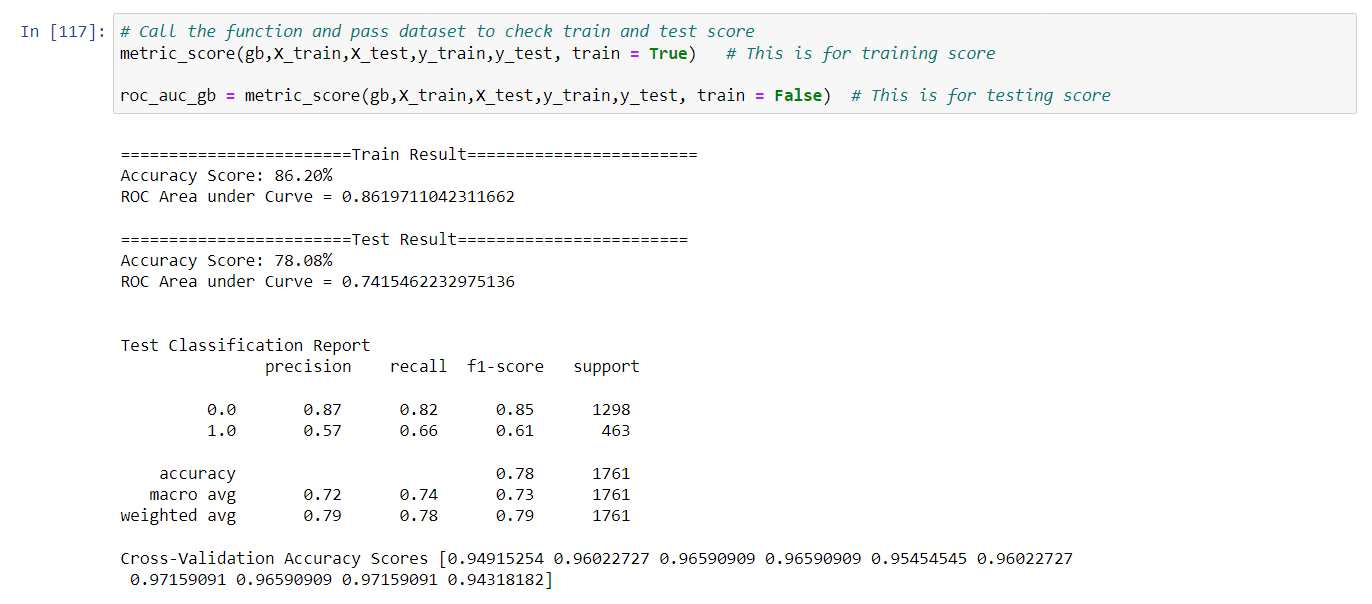
# Call the function and pass dataset to check train and test score  
metric\_score(ab,X\_train,X\_test,y\_train,y\_test, train = True) # This is for training score  
  
roc\_auc\_ab = metric\_score(ab,X\_train,X\_test,y\_train,y\_test, train = False) # This is for testing score 

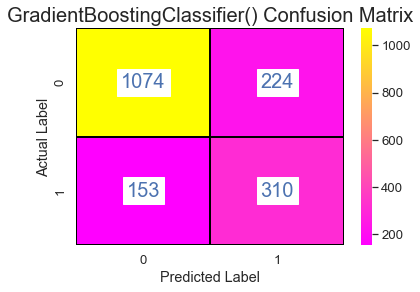


### 5. Gradient Boost

# Running Gradient Boosting  
from sklearn.ensemble import GradientBoostingClassifier  
gb = GradientBoostingClassifier()  
gb.fit(X\_train, y\_train)

GradientBoostingClassifier()

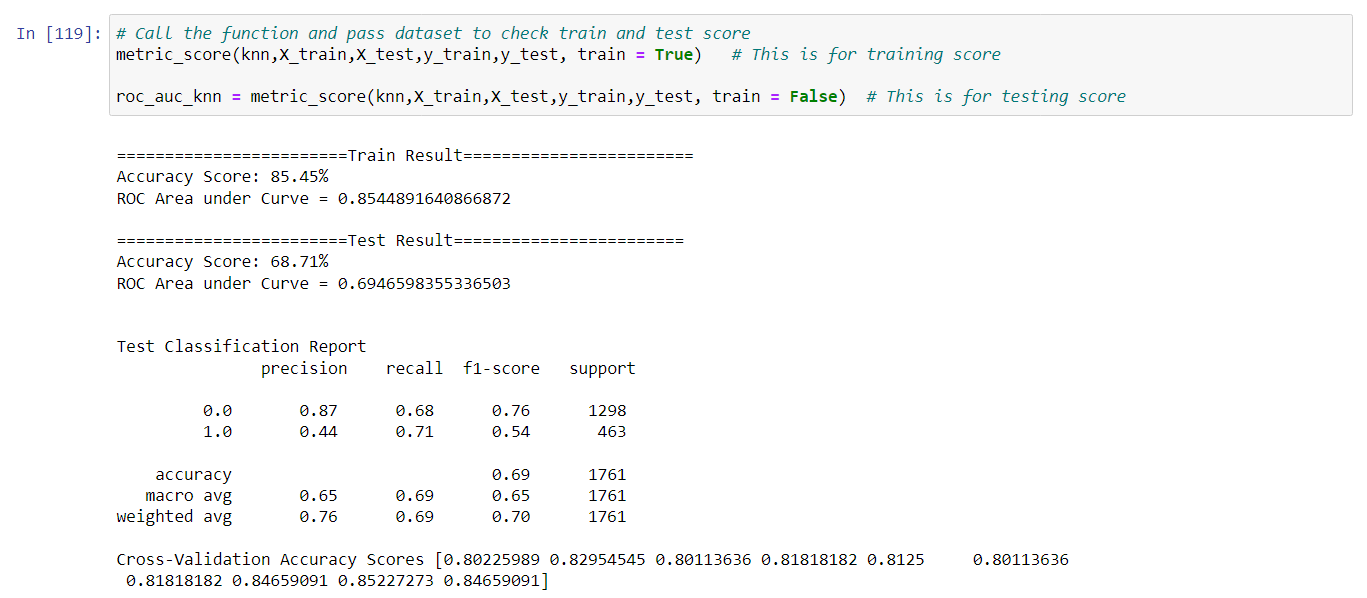
# Call the function and pass dataset to check train and test score  
metric\_score(gb,X\_train,X\_test,y\_train,y\_test, train = True) # This is for training score  
  
roc\_auc\_gb = metric\_score(gb,X\_train,X\_test,y\_train,y\_test, train = False) # This is for testing score 

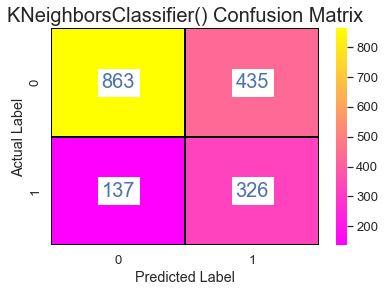


### 6. K-Nearest Neighbors

# Running K-Nearest Neighbors  
from sklearn.neighbors import KNeighborsClassifier  
knn = KNeighborsClassifier()  
knn.fit(X\_train, y\_train)

KNeighborsClassifier()

# Call the function and pass dataset to check train and test score  
metric\_score(knn,X\_train,X\_test,y\_train,y\_test, train = True) # This is for training score  
  
roc\_auc\_knn = metric\_score(knn,X\_train,X\_test,y\_train,y\_test, train = False) # This is for testing score 

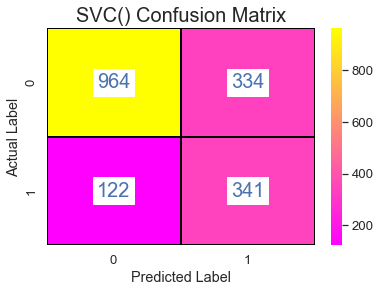
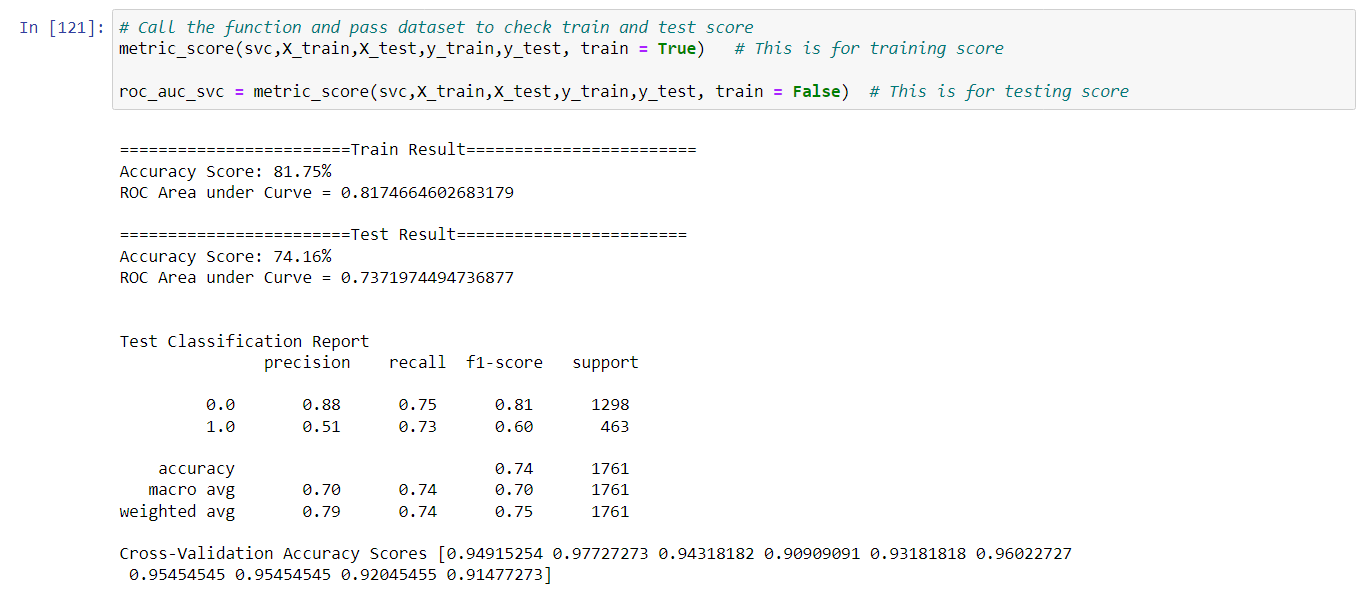


### 7. Support Vector Machine

# Running Support Machine  
from sklearn.svm import SVC  
svc = SVC()  
svc.fit(X\_train, y\_train)

SVC()

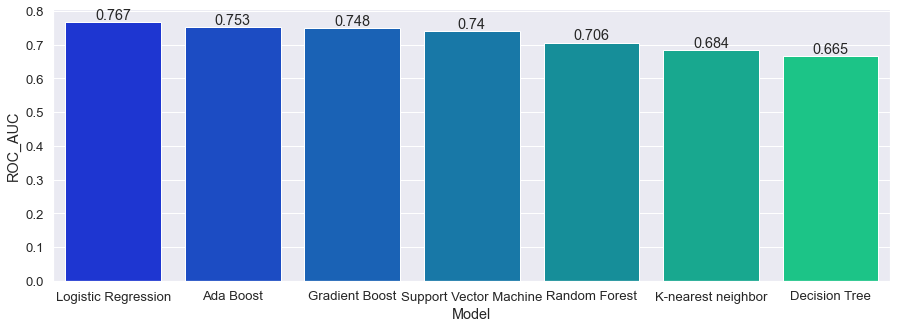
# Call the function and pass dataset to check train and test score  
metric\_score(svc,X\_train,X\_test,y\_train,y\_test, train = True) # This is for training score  
  
roc\_auc\_svc = metric\_score(svc,X\_train,X\_test,y\_train,y\_test, train = False) # This is for testing score



### Let's compare ROC AUC to choose best model

roc\_auc\_scores = [roc\_auc\_lr, roc\_auc\_dt, roc\_auc\_rf, roc\_auc\_ab, roc\_auc\_gb, roc\_auc\_knn, roc\_auc\_svc]  
model\_data = {'Model': ['Logistic Regression','Decision Tree','Random Forest','Ada Boost','Gradient Boost','K-nearest neighbor', 'Support Vector Machine'],  
 'ROC\_AUC': roc\_auc\_scores}  
data = pd.DataFrame(model\_data)

plt.figure(figsize=(15,5))  
ax.set\_title('Model Comparison: Area under ROC and Cohens Kappa', fontsize=13)  
color = 'tab:blue'  
ax.set\_xlabel('Model', fontsize=18)  
ax.set\_ylabel('ROC\_AUC', fontsize=18)  
ax = sns.barplot(x='Model', y='ROC\_AUC', data = data, palette='winter', order=data.sort\_values('ROC\_AUC',ascending = False).Model)  
ax.tick\_params(axis='y')  
for c in ax.containers:  
 labels = [f'{round(float((v.get\_height())),3)}' for v in c]  
 ax.bar\_label(c, labels=labels, label\_type='edge')  
plt.show()



**Since the Logistic Regression, Ada Boost and Gradient Boost are the top 3 models based on AUC score without parameters tuning. Therefore, I am choosing them to work on them.**

## Hyperparameter Tuning

from sklearn.model\_selection import RepeatedStratifiedKFold, GridSearchCV

### 1. Logistic Regression

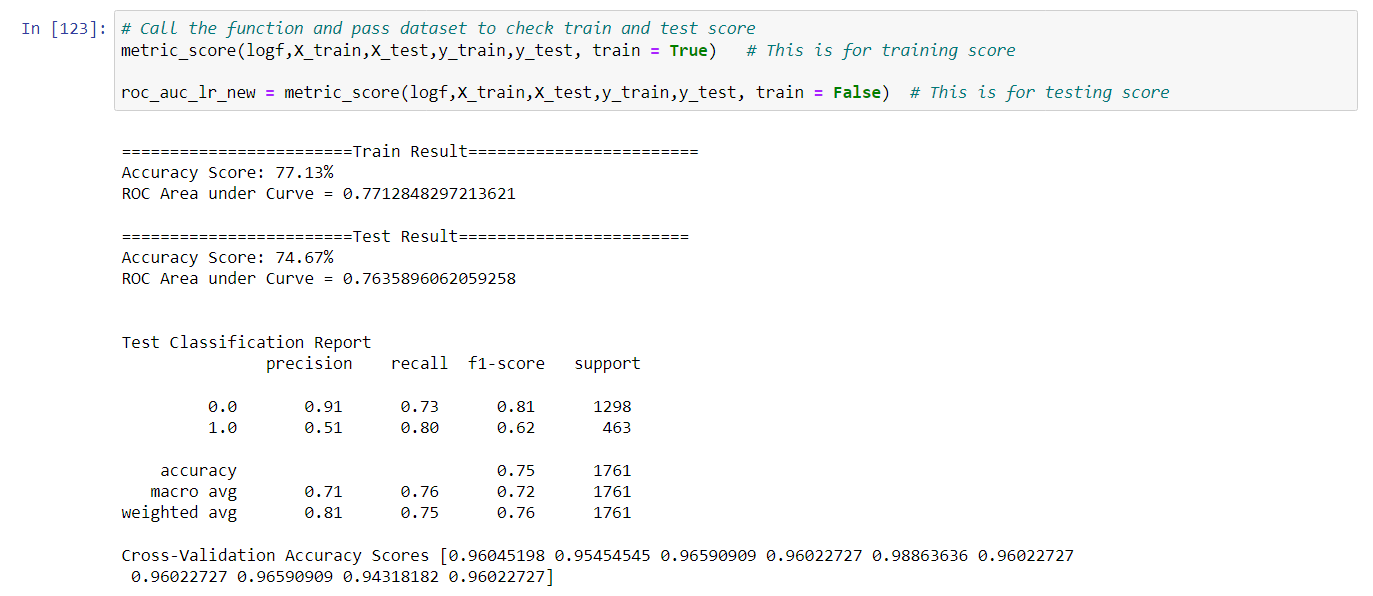
logf = LogisticRegression()  
solvers = ['newton-cg', 'lbfgs', 'liblinear']  
penalty = ['l2']  
# define grid search  
grid = dict(solver=solvers,penalty=penalty)  
cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)  
grid\_search = GridSearchCV(estimator=logf, param\_grid=grid, n\_jobs=-1, cv=cv, scoring='roc\_auc',error\_score=0)  
grid\_result = grid\_search.fit(X, y)  
# summarize results  
print("Best: %f using %s" % (grid\_result.best\_score\_, grid\_result.best\_params\_))  
means = grid\_result.cv\_results\_['mean\_test\_score']  
stds = grid\_result.cv\_results\_['std\_test\_score']  
params = grid\_result.cv\_results\_['params']  
for mean, stdev, param in zip(means, stds, params):  
 print("%f (%f) with: %r" % (mean, stdev, param))

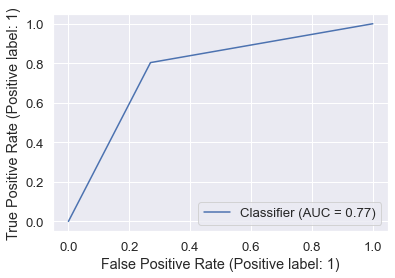
Best: 0.844242 using {'penalty': 'l2', 'solver': 'newton-cg'}  
0.844242 (0.014027) with: {'penalty': 'l2', 'solver': 'newton-cg'}  
0.841007 (0.015086) with: {'penalty': 'l2', 'solver': 'lbfgs'}  
0.842835 (0.014717) with: {'penalty': 'l2', 'solver': 'liblinear'}

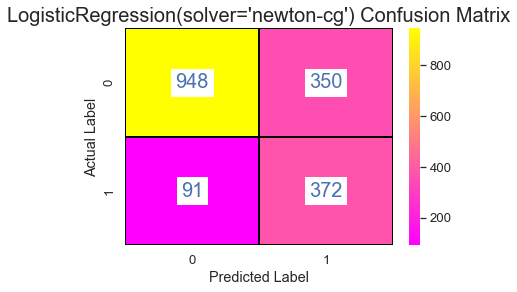
# Running Logistic Regression with the best parameters  
logf = LogisticRegression(penalty= 'l2', solver= 'newton-cg')  
logf.fit(X\_train, y\_train)

LogisticRegression(solver='newton-cg')

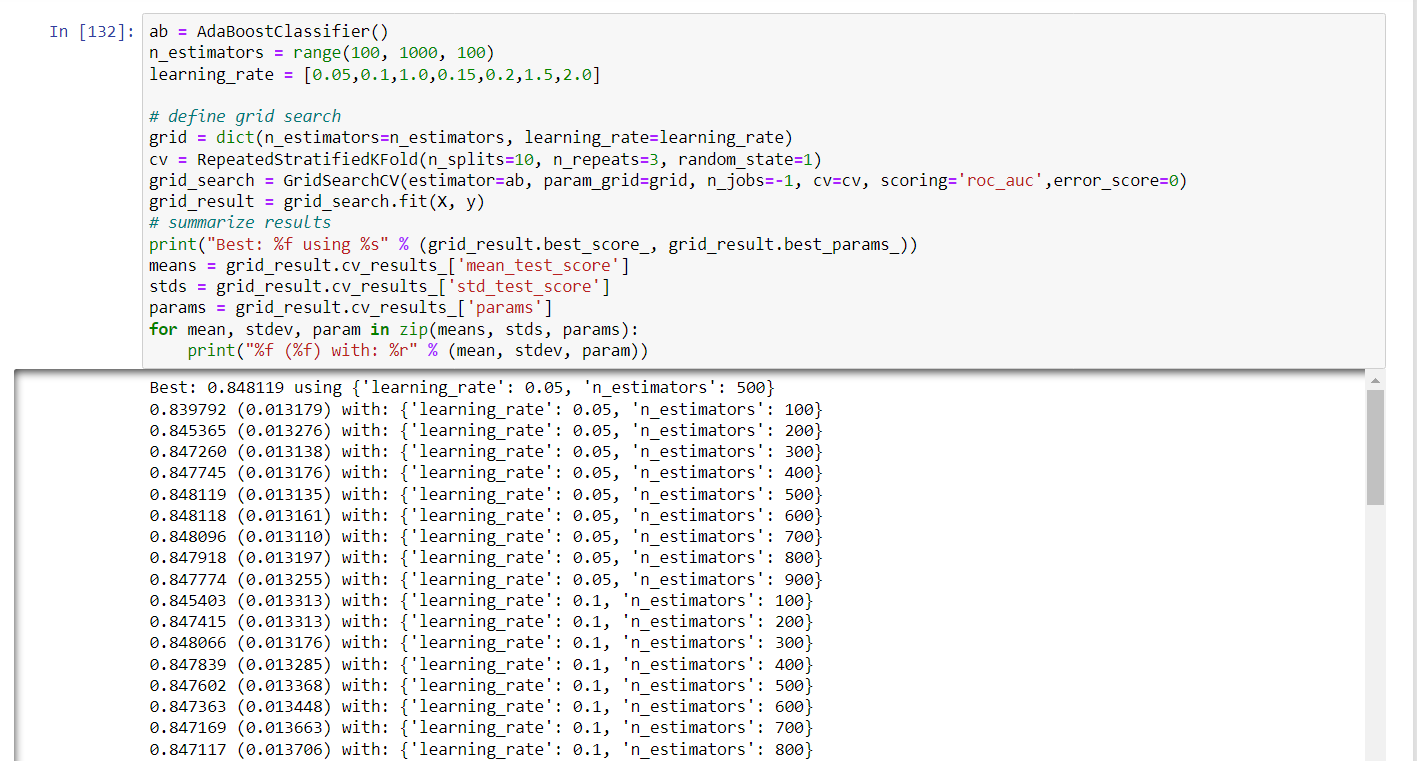
# Call the function and pass dataset to check train and test score  
metric\_score(logf,X\_train,X\_test,y\_train,y\_test, train = True) # This is for training score  
  
roc\_auc\_lr\_new = metric\_score(logf,X\_train,X\_test,y\_train,y\_test, train = False) # This is for testing score







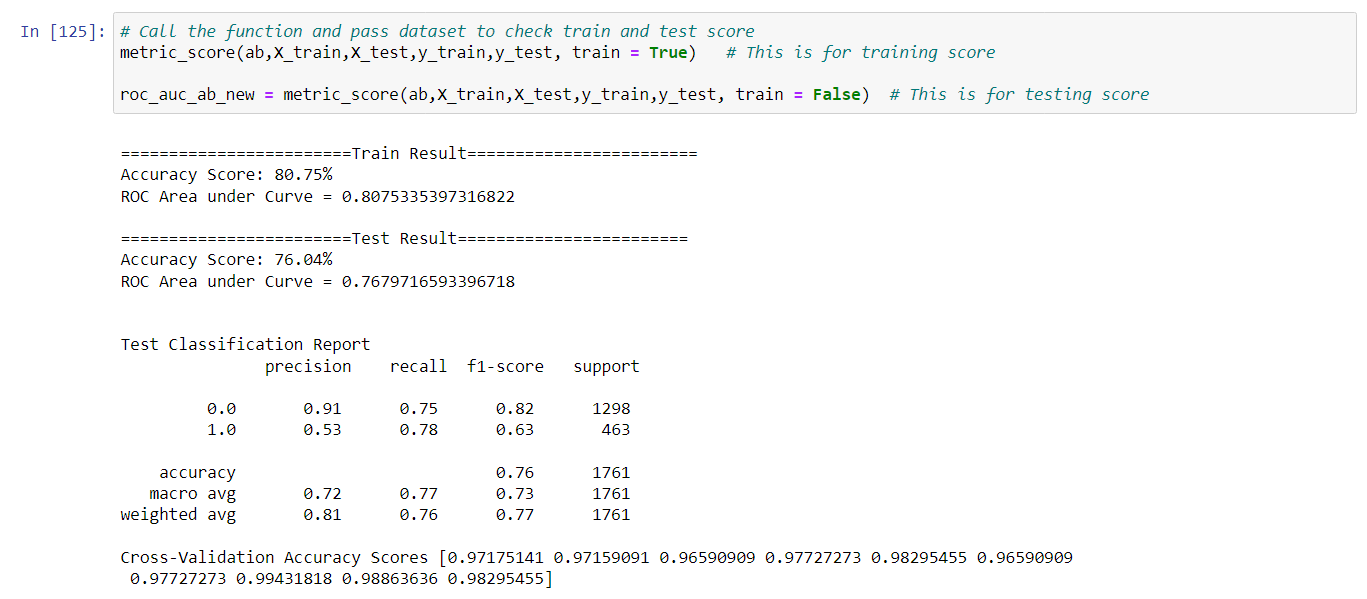
### 2. Ada Boost

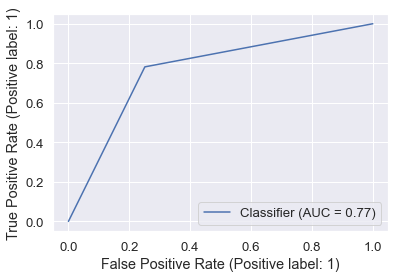
ab = AdaBoostClassifier()  
n\_estimators = range(100, 1000, 100)  
learning\_rate = [0.05,0.1,1.0,0.15,0.2,1.5,2.0]  
  
# define grid search  
grid = dict(n\_estimators=n\_estimators, learning\_rate=learning\_rate)  
cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)  
grid\_search = GridSearchCV(estimator=ab, param\_grid=grid, n\_jobs=-1, cv=cv, scoring='roc\_auc',error\_score=0)  
grid\_result = grid\_search.fit(X, y)  
# summarize results  
print("Best: %f using %s" % (grid\_result.best\_score\_, grid\_result.best\_params\_))  
means = grid\_result.cv\_results\_['mean\_test\_score']  
stds = grid\_result.cv\_results\_['std\_test\_score']  
params = grid\_result.cv\_results\_['params']  
for mean, stdev, param in zip(means, stds, params):  
 print("%f (%f) with: %r" % (mean, stdev, param)) 

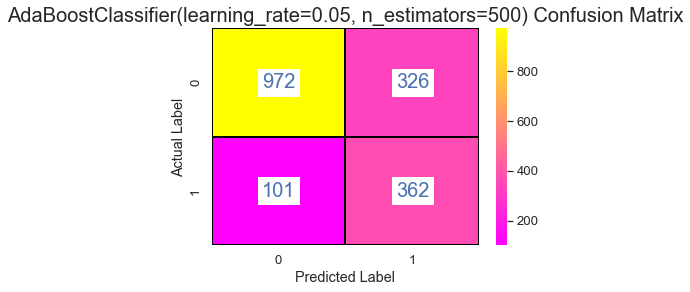
# Running Ada Boost with tuned parameters  
ab = AdaBoostClassifier(learning\_rate= 0.05, n\_estimators= 500)  
ab.fit(X\_train, y\_train)

AdaBoostClassifier(learning\_rate=0.05, n\_estimators=500)

# Call the function and pass dataset to check train and test score  
metric\_score(ab,X\_train,X\_test,y\_train,y\_test, train = True) # This is for training score  
  
roc\_auc\_ab\_new = metric\_score(ab,X\_train,X\_test,y\_train,y\_test, train = False) # This is for testing score

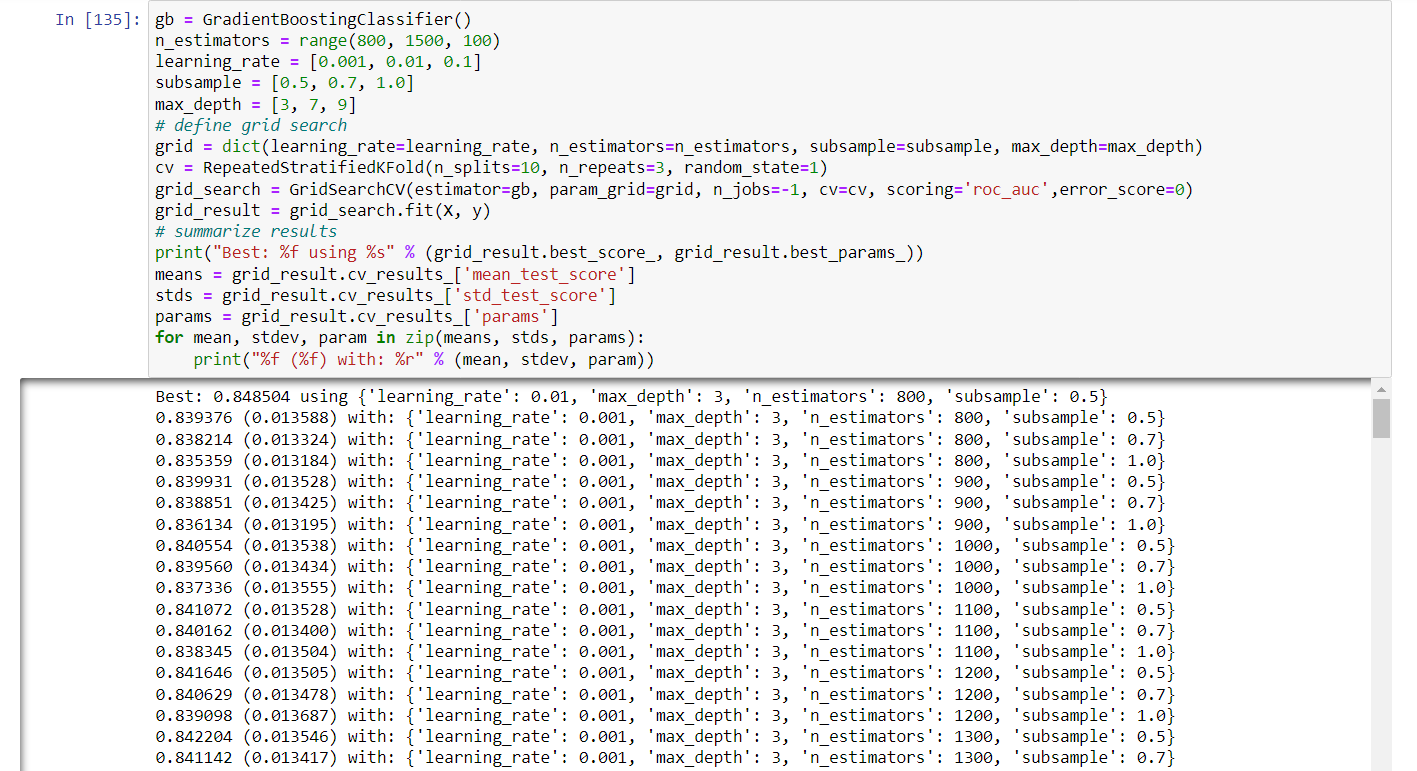






### 3. Gradient Boost

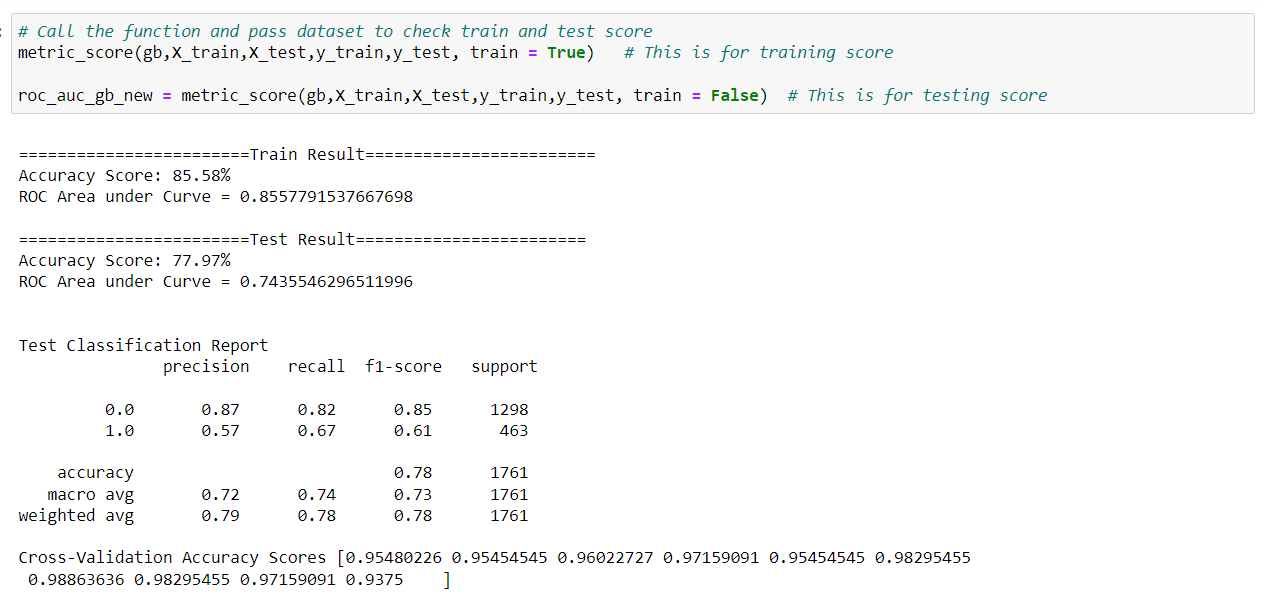
gb = GradientBoostingClassifier()  
n\_estimators = range(800, 1500, 100)  
learning\_rate = [0.001, 0.01, 0.1]  
subsample = [0.5, 0.7, 1.0]  
max\_depth = [3, 7, 9]  
# define grid search  
grid = dict(learning\_rate=learning\_rate, n\_estimators=n\_estimators, subsample=subsample, max\_depth=max\_depth)  
cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)  
grid\_search = GridSearchCV(estimator=gb, param\_grid=grid, n\_jobs=-1, cv=cv, scoring='roc\_auc',error\_score=0)  
grid\_result = grid\_search.fit(X, y)  
# summarize results  
print("Best: %f using %s" % (grid\_result.best\_score\_, grid\_result.best\_params\_))  
means = grid\_result.cv\_results\_['mean\_test\_score']  
stds = grid\_result.cv\_results\_['std\_test\_score']  
params = grid\_result.cv\_results\_['params']  
for mean, stdev, param in zip(means, stds, params):  
 print("%f (%f) with: %r" % (mean, stdev, param))



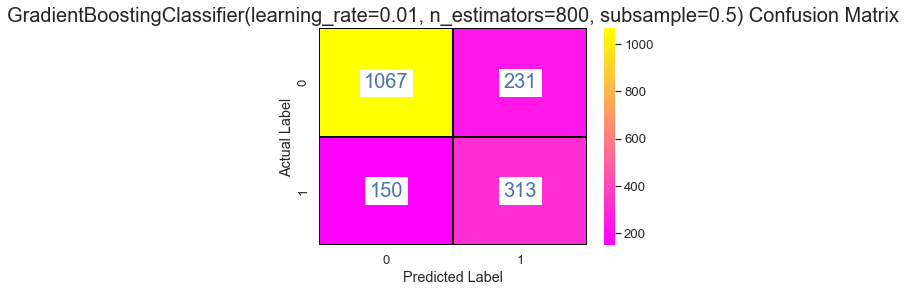
# Running Gradient Boosting  
gb = GradientBoostingClassifier(learning\_rate= 0.01, max\_depth= 3, n\_estimators= 800, subsample= 0.5)  
gb.fit(X\_train, y\_train)

GradientBoostingClassifier(learning\_rate=0.01, n\_estimators=800, subsample=0.5)

# Call the function and pass dataset to check train and test score  
metric\_score(gb,X\_train,X\_test,y\_train,y\_test, train = True) # This is for training score  
  
roc\_auc\_gb\_new = metric\_score(gb,X\_train,X\_test,y\_train,y\_test, train = False) # This is for testing score



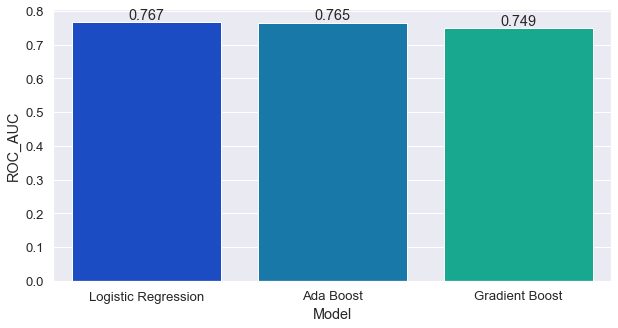




### Let's compare ROC AUC for tuned models to choose best model

roc\_auc\_scores = [roc\_auc\_lr\_new, roc\_auc\_ab\_new, roc\_auc\_gb\_new]  
model\_data = {'Model': ['Logistic Regression', 'Ada Boost','Gradient Boost'],  
 'ROC\_AUC': roc\_auc\_scores}  
data\_new = pd.DataFrame(model\_data)

plt.figure(figsize=(10,5))  
ax.set\_title('Model Comparison: Area under ROC and Cohens Kappa', fontsize=13)  
color = 'tab:blue'  
ax.set\_xlabel('Model', fontsize=18)  
ax.set\_ylabel('ROC\_AUC', fontsize=18)  
ax = sns.barplot(x='Model', y='ROC\_AUC', data = data\_new, palette='winter', order=data.sort\_values('ROC\_AUC',ascending = False).Model)  
ax.tick\_params(axis='y')  
for c in ax.containers:  
 labels = [f'{round(float((v.get\_height())),3)}' for v in c]  
 ax.bar\_label(c, labels=labels, label\_type='edge')  
plt.show()



#### Since the scores did not improve much therefore, using the model without any hyper tuning.

From the above model, Logistic Regression seems to be performing well based on accuracy and recall.

# saving the model  
import pickle  
filename = 'telecom\_churn.pkl'  
pickle.dump(rf, open(filename, 'wb'))

# Ways to reduce the customer churn rate

Customer retention is a key factor in improving the customer churn rate. To reduce the number of customers who leave our business, we can use predictive analytics to identify those at risk of churn and use machine learning (ML) to automate their retention programs.

Data science offers other ways to improve customer satisfaction and loyalty by using data from different sources, such as social media platforms or surveys. The goal is not just to understand what makes unhappy customers unhappy—but how they can be helped through better service delivery or product features that improve their experience with the company's products or services.

# Conclusion

Customer churn is the number of customers that leave our business which may be calculated for different time intervals, for example, within a year. Most companies have a customer churn rate they strive to keep below 10%, but it can be higher or lower depending on the industry we are in and how well we service our customers. In our dataset, the customer churn rate is 26.29% which is very high.

Based on our Machine Learning Model we can predict almost 80% of customers who are actually going to churn. Although, our model is incorrectly predicting for 26.96% of customers who did not churn.