**PROJECT TITLE**

Predicting Customer Churn in a Telecommunications Company Using Machine Learning Techniques.

**DATASET**

The dataset used in this project is the [Telco Customer Churn dataset]() from Kaggle. It contains information on customers of a telecommunications company, including demographic information, services used, and whether the customer churned. There are 7043 rows and 21 columns in the dataset.

Dataset Description

| Feature Name | Description | |
|---|---|---|
| customerID | Contains customer ID | categorical |
| gender | whether the customer female or male | categorical |
| SeniorCitizen | Whether the customer is a senior citizen or not (1, 0) | numeric, int |
| Partner | Whether the customer has a partner or not (Yes, No) | categorical |
| Dependents | Whether the customer has dependents or not (Yes, No) | categorical |
| tenure | Number of months the customer has stayed with the company | numeric, int |
| PhoneService | Whether the customer has a phone service or not (Yes, No) | categorical |
| MultipleLines | Whether the customer has multiple lines r not (Yes, No, No phone service) | categorical |
| InternetService | Customer's internet service provider (DSL, Fiber optic, No) | categorical |
| OnlineSecurity | Whether the customer has online security or not (Yes, No, No internet service) | categorical |
| OnlineBackup | Whether the customer has online backup or not (Yes, No, No internet service) | categorical |
| DeviceProtection | Whether the customer has device protection or not (Yes, No, No internet service) | categorical |
| TechSupport | Whether the customer has tech support or not (Yes, No, No internet service) | categorical |
| streamingTV | Whether the customer has streaming TV or not (Yes, No, No internet service) | categorical |
| streamingMovies | Whether the customer has streaming movies or not (Yes, No, No internet service) | categorical |
| Contract | The contract term of the customer (Month-to-month, One year, Two year) | categorical |
| PaperlessBilling | Whether the customer has paperless billing or not (Yes, No) | categorical |
| PaymentMethod | The customer's payment method (Electronic check, Mailed check, Bank transfer, Credit card) | categorical |
| MonthlyCharges | The amount charged to the customer monthly | numeric , int |
| TotalCharges | The total amount charged to the customer | object |
| Churn | Whether the customer churned or not (Yes or No) | categorical |

**Step 1: Frame of the Problem**

This project aims to solve the problem of customer churn in the telecommunications industry. Customer churn refers to the loss of customers who cancel their subscriptions or stop using a product or service, which can be costly for companies, leading to revenue loss and affecting customer loyalty and brand reputation. The project uses machine learning algorithms to predict whether customers will likely churn based on available features such as demographics, services, and account information. The project aims to help telecommunications companies reduce customer churn rates and improve customer retention strategies by building a predictive model. The project also includes feature importance analysis and model interpretation, providing insights into the key drivers of customer churn, which telecommunications companies can use to improve their customer retention strategies. Ultimately, the project demonstrates the potential of machine learning techniques for predicting customer churn and provides a roadmap for developing similar models in other industries.

Objectives:

- Build a machine learning model to predict customer churn in the telecommunications industry.
- Select relevant features and evaluate and compare the performance of different machine learning algorithms (Logistic Regression, Random Forest, XGBoost,…) using multiple evaluation metrics.
- Identify the key drivers of customer churn using feature importance analysis and model interpretation.
- Provide insights and recommendations to help telecommunications companies reduce customer churn rates and improve customer retention strategies.

Solution:

- The project follows a typical machine learning pipeline, starting with data cleaning and preprocessing, followed by exploratory data analysis and feature engineering.
- Some different machine learning algorithms are trained and evaluated using multiple evaluation metrics, with the XGBoost algorithm identified as the best-performing algorithm for predicting customer churn in this dataset.
- Feature importance analysis and model interpretation provide insights into the key drivers of customer churn, which telecommunications companies can use to improve their customer retention strategies.

**Step 2: Get the data.**

Import Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.figure_factory as ff
import seaborn as sns
import sklearn
import xgboost as xgb
import imblearn
```

Loading the dataset into a pandas dataframe

```
df = pd.read_csv('Telco-Customer-Churn.csv')
```

```
df.head()
```

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

5 rows × 21 columns

**Step 3: Explore the data to gain insights.**

Data Exploration

```
print('Dataset shape: ',df.shape)
```

```
Dataset shape:  (7043, 21)
```

```
df.describe()
```

| | 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 |

```
df.info()
```

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

This gives a comprehensive display of the dataset's statistical properties. Not only does it unveil the number of rows and columns, but it also exposes the intricate details of the data's distribution. The statistical summary encompasses many essential metrics, including the count, mean, standard deviation, and minimum and maximum validity as percentiles. In addition to this, the dataset summary paints a vivid picture of the data's composition, revealing the number of non-null values and the data types for each column. These valuable insights serve as the bedrock for informed decision-making and optimal utilization of the dataset.

Data Cleaning

Converting TotalCharges to numeric.

```
df['TotalCharges']=pd.to_numeric(df['TotalCharges'], errors='coerce')
df.TotalCharges.dtypes
```

```
missing_data = df.isnull().sum(axis=0).reset_index()  #checking missing value
missing_data.columns=['variable','missing values']
missing_data['filling factor %']=(df.shape[0]-missing_data['missing values'])/df.shape[0]*100
missing_data.sort_values('filling factor %').reset_index(drop = True)
```

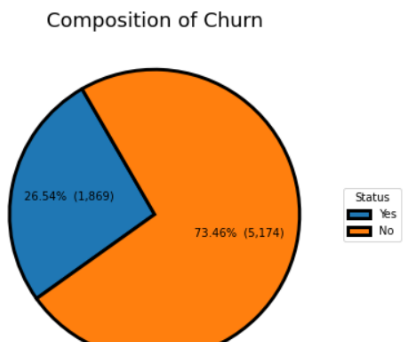
| | variable | missing values | filling factor % |
|---|---|---|---|
| 0 | TotalCharges | 11 | 99.843817 |
| 1 | customerID | 0 | 100.000000 |
| 2 | MonthlyCharges | 0 | 100.000000 |
| 3 | PaymentMethod | 0 | 100.000000 |
| 4 | PaperlessBilling | 0 | 100.000000 |
| 5 | Contract | 0 | 100.000000 |
| 6 | StreamingMovies | 0 | 100.000000 |
| 7 | StreamingTV | 0 | 100.000000 |
| 8 | TechSupport | 0 | 100.000000 |
| 9 | DeviceProtection | 0 | 100.000000 |
| 10 | OnlineBackup | 0 | 100.000000 |
| 11 | InternetService | 0 | 100.000000 |
| 12 | MultipleLines | 0 | 100.000000 |
| 13 | PhoneService | 0 | 100.000000 |
| 14 | tenure | 0 | 100.000000 |
| 15 | Dependents | 0 | 100.000000 |
| 16 | Partner | 0 | 100.000000 |
| 17 | SeniorCitizen | 0 | 100.000000 |
| 18 | gender | 0 | 100.000000 |
| 19 | OnlineSecurity | 0 | 100.000000 |
| 20 | Churn | 0 | 100.000000 |

This describes a procedure for calculating the percentage of missing values in a panda DataFrame and displaying the resulting values in a sorted DataFrame. Initially, a new DataFrame, "missing data," is constructed by summing the number of missing values for each variable in the original DataFrame. Subsequently, the filling factor percentage is computed for each variable. The resulting sorted DataFrame presents the variables with the highest percentage of missing values at the top and those with the lowest percentage of missing values at the bottom. This methodology is a valuable tool for identifying variables that necessitate attention during data analysis.

## Data Visualization

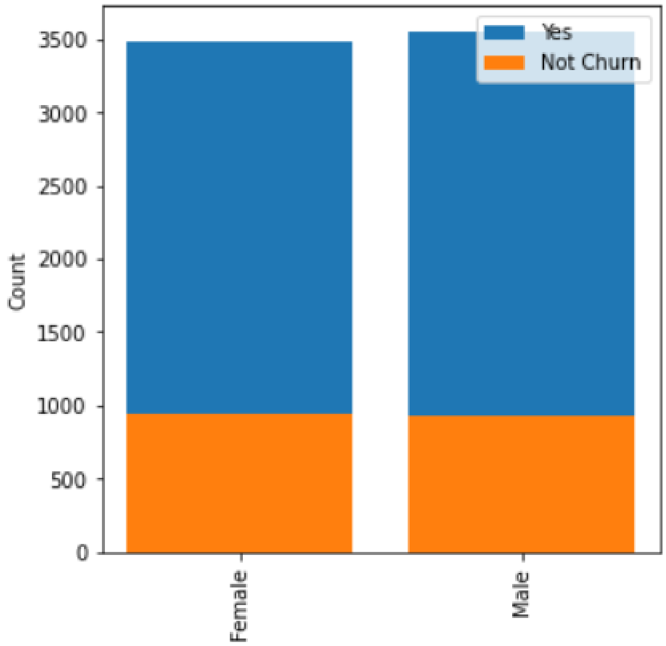
```
value_count=df.Churn.value_counts()

fig,ax=plt.subplots(figsize=(10, 6),subplot_kw=dict(aspect="equal"))
plt.title('Composition of Churn',fontsize=18)

data=[value_count[1],value_count[0]]

plt.pie(data,explode = (0,0),
        textprops=dict(size= 10, color= "black"),
        autopct=lambda p : '{:.2f}%  ({:,.0f})'.format(p,p * sum(data)/100),startangle = 120,wedgeprops=dict( edgecolor = "black'


labels='Yes','No'
plt.legend(labels,title='Status',loc='center',bbox_to_anchor=(1,0,0.2,1))
```

```
<matplotlib.legend.Legend at 0x165dc59e940>
```




The churn rate in the dataset is approximately 26.5%. This means that around 26.5% of customers in the dataset have churned, while the remaining 73.5% have not. This information shown on the bar chart can be used to assess the severity of the churn problem for the telecommunications company.

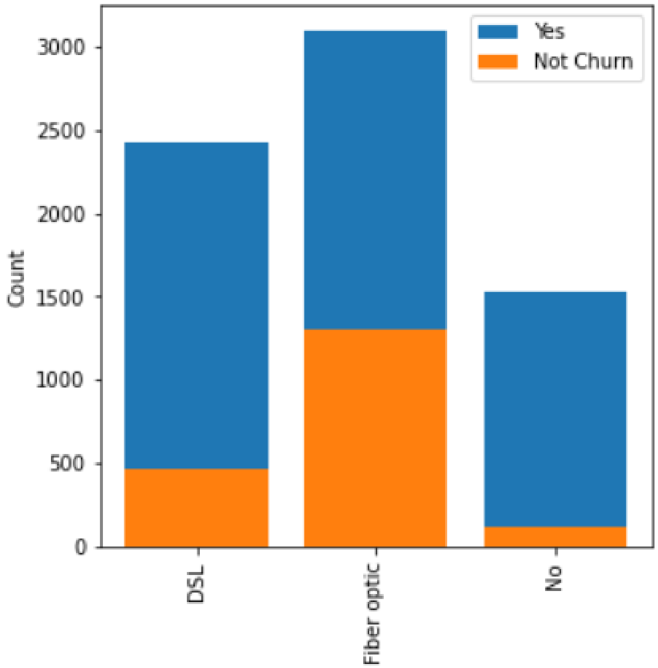
```
barplots(data,'gender','Churn_Yes')
```




```
   gender  Churn_Yes  total       Avg
0  Female        939   3488  0.269209
1    Male        930   3555  0.261603
```

Visualizing the relationship between gender (Male and Female) and churn. When we looked at the churn rate by gender, we found that female customers had a slightly higher churn rate (26.0%) than male customers (23.2%).The bar chart shows 939 female customers churn out 3488 and 930 male customers out of 3555.

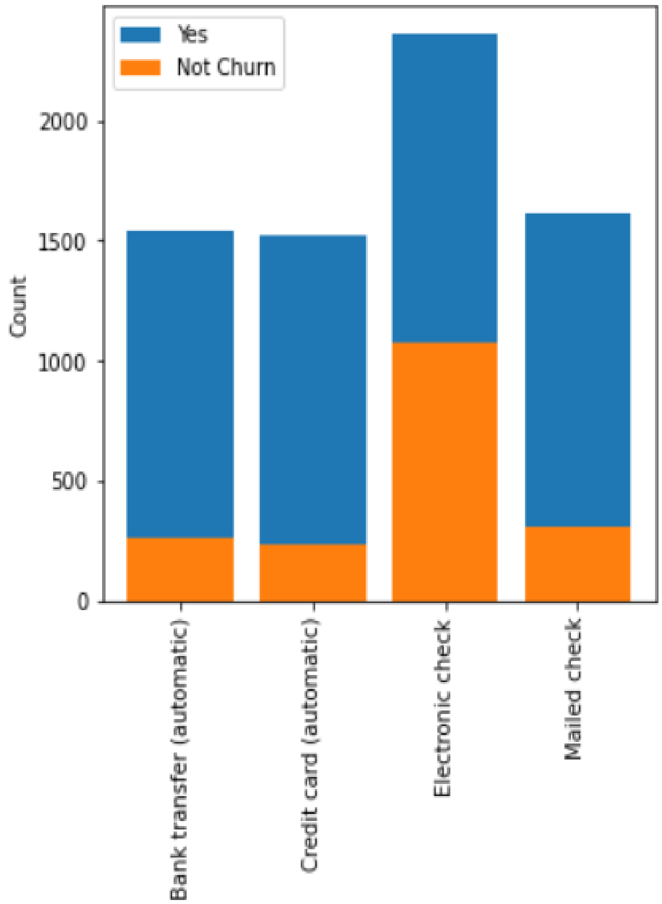
```
barplots(data,'InternetService','Churn_Yes')
```




```
  InternetService  Churn_Yes  total       Avg
0             DSL        459   2421  0.189591
1     Fiber optic       1297   3096  0.418928
2              No        113   1526  0.074050
```

This bar chart shows the relationship between customers' internet service and churn. 459 customers churned out of 2421 and used DSL internet service. 1297 customers churned out of 3096 with fiber optic internet service, and 113 customers churned out of 1526 without internet service.

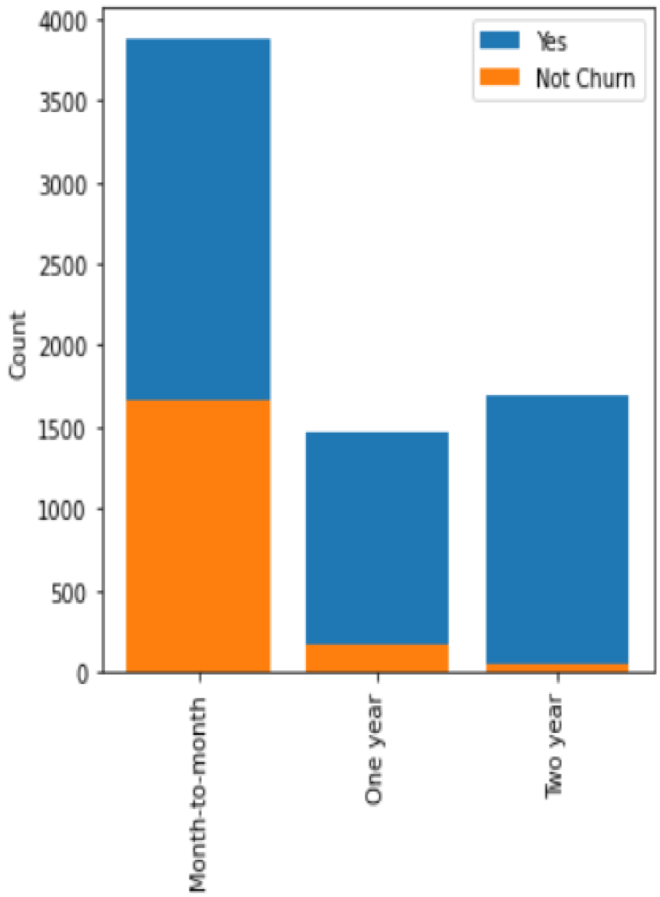
```
barplots(data,'PaymentMethod','Churn_Yes')
```




| | PaymentMethod | Churn_Yes | total | Avg |
|---|---|---|---|---|
| 0 | Bank transfer (automatic) | 258 | 1544 | 0.167098 |
| 1 | Credit card (automatic) | 232 | 1522 | 0.152431 |
| 2 | Electronic check | 1071 | 2365 | 0.452854 |
| 3 | Mailed check | 308 | 1612 | 0.191067 |

This bar chart shows how payment methods influenced customer churn. There are four different payment methods (bank transfer, credit card, electronic check, and mailed check). 256 customers churned out of 1544 using bank transfer as the payment method. 232 customers churned out of 1522 as credit card as payment method. 1071 customers churned out of 2365 electronic checks as the payment method, and 308 customers churned out of 1612 using mailed checks as the payment method.

```
barplots(data,'Contract','Churn_Yes')
```

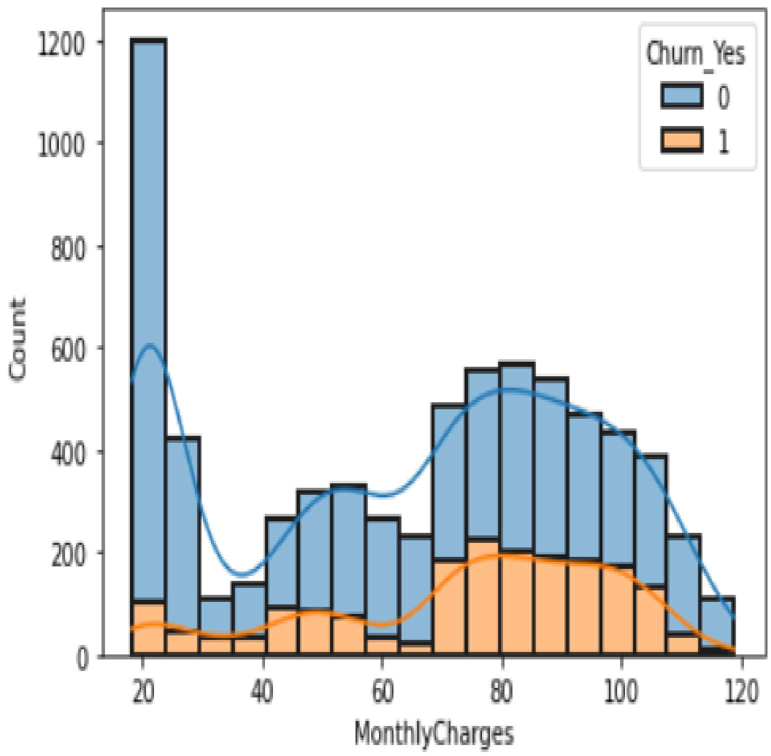



| | Contract | Churn_Yes | total | Avg |
|---|---|---|---|---|
| 0 | Month-to-month | 1655 | 3875 | 0.427097 |
| 1 | One year | 166 | 1473 | 0.112695 |
| 2 | Two year | 48 | 1695 | 0.028319 |

This graph shows the number of customer churn based on contract (Month-to-Month, One year, and Two years). 1655 customers churned out of 3875 month-to-month contracts. 166 customers churned out of1473 one-year contracts, and 46 customers churned out of 1695 two years contracts.

```
(df,x='MonthlyCharges',hue='Churn_Yes',kde=True,multiple='stack',element='bars'
```

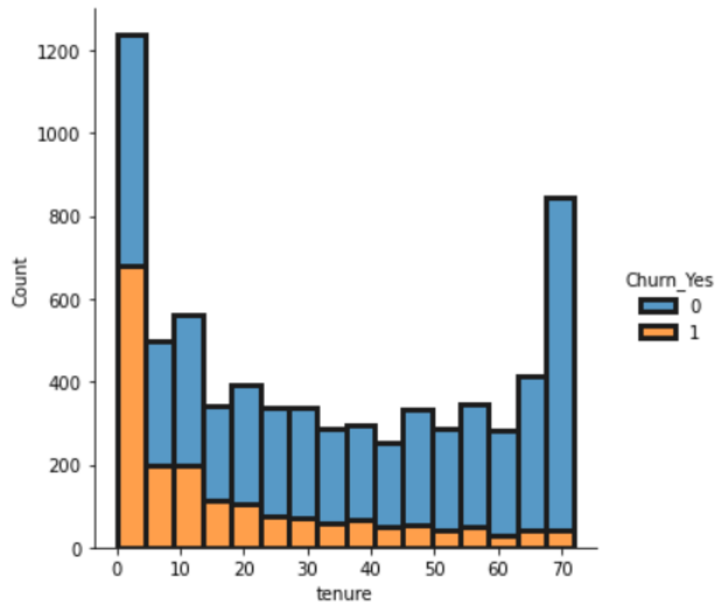
```
<AxesSubplot:xlabel='MonthlyCharges', ylabel='Count'>
```




The code creates a histogram plot using the Seaborn library to show the distribution of monthly charges for customers who churned and those who didn't. The bars are stacked on each other and colored differently based on whether the customer churned. The histogram shows that customers have more churn on monthly charges of 80 and a high number of not churn at a monthly charge of 20, which showed that churned customers tended to have higher monthly charges than non-churned customers.

```
sns.displot(df,x='tenure',hue='Churn_Yes',edgecolor = "#1c1c1c", linewidth = 3
```

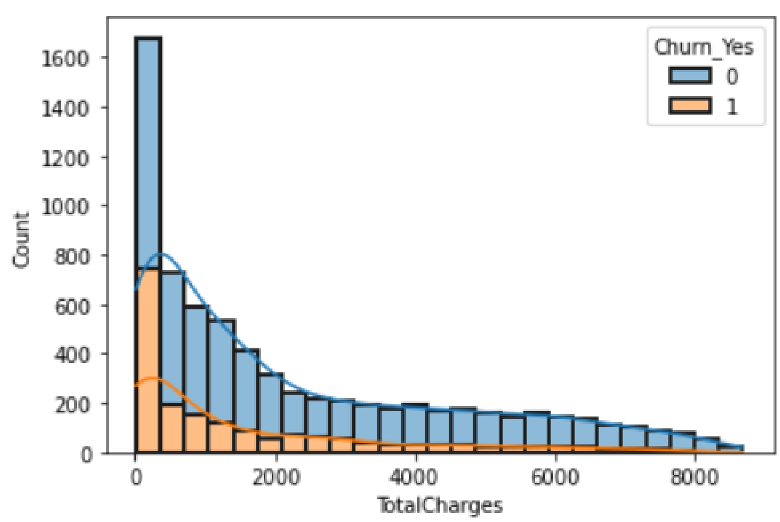
```
<seaborn.axisgrid.FacetGrid at 0x165dc56b880>
```




The graph shows the highest churn between 0 to 10 months tenure and the least churn at 60 to 70 months tenure. The histogram also showed that non-churned customers tended to have higher tenure than churned customers.

```
t(df,x='TotalCharges', hue='Churn_Yes',kde=True,multiple='stack',element='bars'
```

```
<AxesSubplot:xlabel='TotalCharges', ylabel='Count'>
```



total charges rate for both churned and non-churned customers, which showed that churned customers tended to have higher total charges rates than non-churned customers.

Overall, the exploration demonstrated that monthly charges, tenure, and total charges rate were all important predictors of customer churn in the Telco Customer Churn dataset. Specifically, customers with higher monthly charges, lower tenure, and higher total charges rates were found to be more likely to churn.

**Step 4: Data Preparation**

In preparing the data, customer identification is removed because it is not useful in building the model and checking the number of missing data. There are 11 missing data from the total charges' column from the outcome.

```
df=df.drop(['customerID'],axis=1)
```

```
missing_data = df.isnull().sum(axis=0).reset_index()
missing_data.columns=['variable','missing values']
missing_data.sort_values('missing values',ascending=False).reset_index(drop =
```

| | variable | missing values |
|---|---|---|
| 0 | TotalCharges | 11 |
| 1 | gender | 0 |
| 2 | SeniorCitizen | 0 |
| 3 | MonthlyCharges | 0 |
| 4 | PaymentMethod | 0 |
| 5 | PaperlessBilling | 0 |
| 6 | Contract | 0 |
| 7 | StreamingMovies | 0 |
| 8 | StreamingTV | 0 |
| 9 | TechSupport | 0 |
| 10 | DeviceProtection | 0 |
| 11 | OnlineBackup | 0 |
| 12 | OnlineSecurity | 0 |
| 13 | InternetService | 0 |
| 14 | MultipleLines | 0 |
| 15 | PhoneService | 0 |
| 16 | tenure | 0 |
| 17 | Dependents | 0 |
| 18 | Partner | 0 |
| 19 | Churn_Yes | 0 |

filling the missing values in the "TotalCharges" column with the value 0, and then modifying the original DataFrame in place (using the "inplace=True" parameter) and the data is then restructured.

```
df.TotalCharges.fillna(0,inplace=True)
```

```
df2=df
```

```
catcol = [col for col in df2.columns if df2[col].dtype == "object"] #encoding
le = LabelEncoder()
for col in catcol:
        df2[col] = le.fit_transform(df2[col])
```

```
df2.head()
```

| neBackup | DeviceProtection | TechSupport | StreamingTV | StreamingMovies | Contract | PaperlessBillir |
|---|---|---|---|---|---|---|
| 2 | 0 | 0 | 0 | 0 | 0 | |
| 0 | 2 | 0 | 0 | 0 | 1 | |
| 2 | 0 | 0 | 0 | 0 | 0 | |
| 0 | 2 | 2 | 0 | 0 | 1 | |
| 0 | 0 | 0 | 0 | 0 | 0 | |

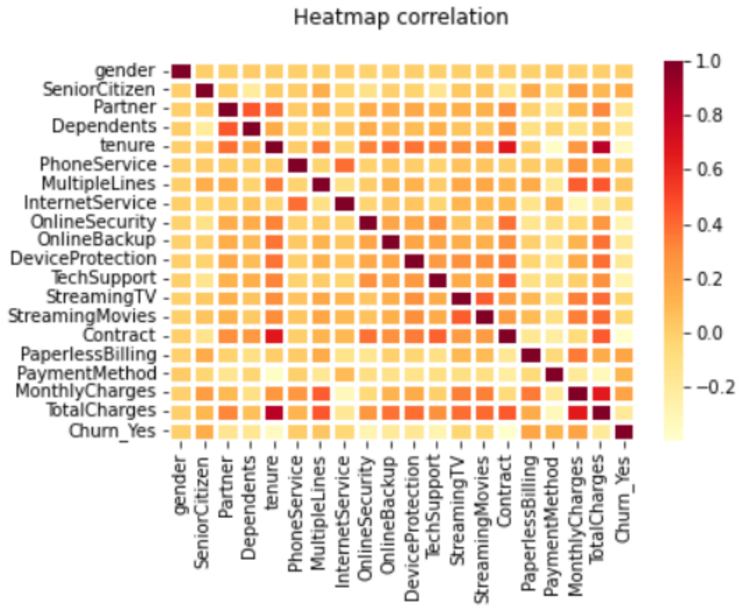
Checking the correlation of all the features with customers that churned (churn_Yes). The table below shows a strong correlation between customers that churned and their contracts with the company, and also a strong correlation between customers that churned and the time frame they were with the company (tenure).

```
df3=df2.corr().Churn_Yes.sort_values(ascending=False).reset_index()
df3
```

| | index | Churn_Yes |
|---|---|---|
| **0** | Churn_Yes | 1.000000 |
| **1** | MonthlyCharges | 0.193356 |
| **2** | PaperlessBilling | 0.191825 |
| **3** | SeniorCitizen | 0.150889 |
| **4** | PaymentMethod | 0.107062 |
| **5** | MultipleLines | 0.038037 |
| **6** | PhoneService | 0.011942 |
| **7** | gender | -0.008612 |
| **8** | StreamingTV | -0.036581 |
| **9** | StreamingMovies | -0.038492 |
| **10** | InternetService | -0.047291 |
| **11** | Partner | -0.150448 |
| **12** | Dependents | -0.164221 |
| **13** | DeviceProtection | -0.178134 |
| **14** | OnlineBackup | -0.195525 |
| **15** | TotalCharges | -0.198324 |
| **16** | TechSupport | -0.282492 |
| **17** | OnlineSecurity | -0.289309 |
| **18** | tenure | -0.352229 |
| **19** | Contract | -0.396713 |

```
sns.heatmap(df2.corr(),cmap="YlOrRd", edgecolor = "#1c1c1c", linewidth = 3)
plt.title(f'\nHeatmap correlation\n')
```

```
Text(0.5, 1.0, '\nHeatmap correlation\n')
```




Testing and training the dataset before building the models.

```
X=df2.drop(['Churn_Yes'],1)
y=df2[['Churn_Yes']]
```

```
C:\Users\MODUPE~1\AppData\Local\Temp/ipykernel_38084/2004873096.py:1: FutureW
arning: In a future version of pandas all arguments of DataFrame.drop except
for the argument 'labels' will be keyword-only
  X=df2.drop(['Churn_Yes'],1)
```

```
oversample = RandomOverSampler(sampling_strategy=1)
X_over, y_over = oversample.fit_resample(X, y)
```

```
train_X,test_X,train_y,test_y = train_test_split(X_over,y_over,test_size=0.2,r
```

```
test_X.shape,test_y.shape
```

```
((2070, 19), (2070, 1))
```

**Steps 5 & 6: Explore many models and shortlist the best ones.**

Building and evaluating a Decision Tree Model using the **DecisionTreeClassifier**

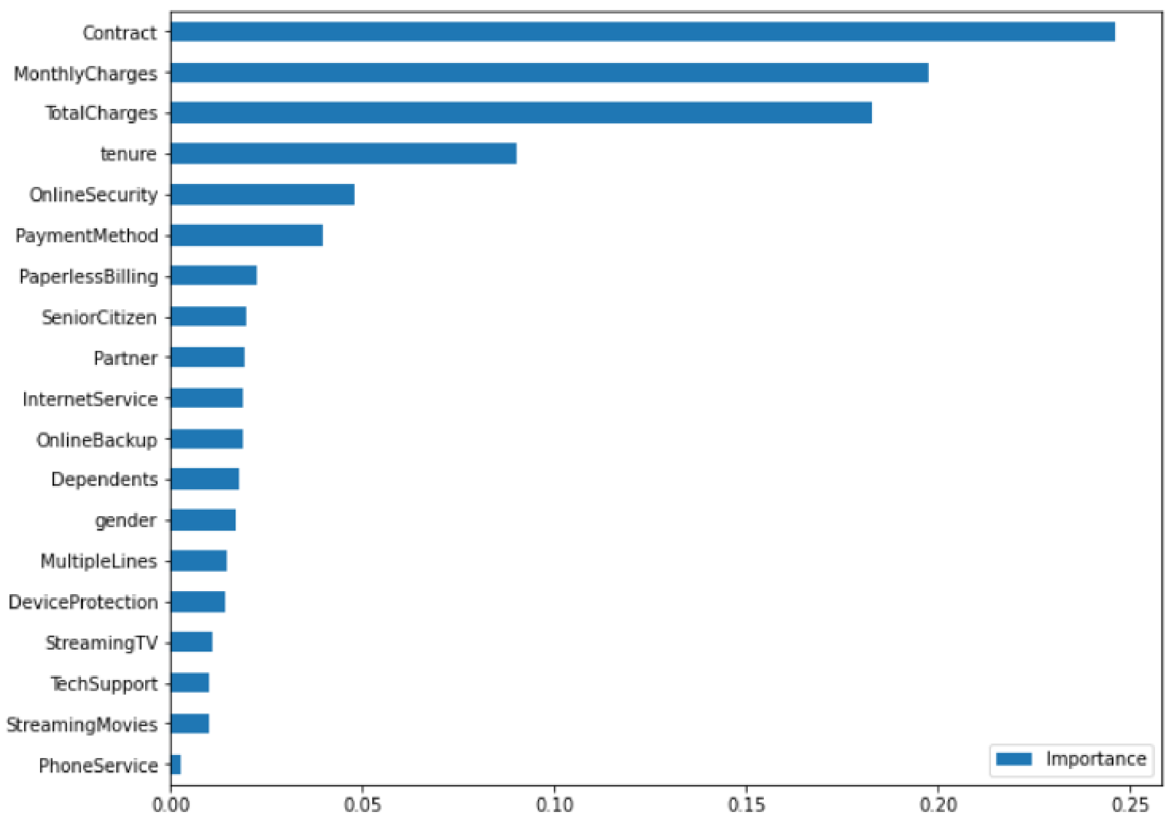
```
dtr=DecisionTreeClassifier()

dtr.fit(train_X,train_y)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

Visualizing the feature importance of the Decision Tree Model

```
feat_importances = pd.DataFrame(dtr.feature_importances_, index=test_X.columns
feat_importances.sort_values(by='Importance', ascending=True, inplace=True)
feat_importances.plot(kind='barh',figsize=(10,8))
```

```
<AxesSubplot:>
```



```
dtr_pred=dtr.predict(test_X)
dtr_conf=confusion_matrix(test_y,dtr_pred)
dtr_report=classification_report(test_y,dtr_pred)
dtr_acc=round(accuracy_score(test_y,dtr_pred)*100,ndigits=3)
dtr_rocauc=roc_auc_score(test_y, dtr_pred)
print(f"Confusion Matrix : \n\n{dtr_conf}")
print(f"\nClassification Report : \n\n{dtr_report}")
print(f"\nThe Accuracy of Decision Tree is {dtr_acc} %")
print(f'ROC AUC Score with Decision Tree: {dtr_rocauc}')
```

```
Confusion Matrix :

[[821 212]
```

to scroll output; double click to hide

```
Classification Report :

              precision    recall  f1-score   support

           0       0.94      0.79      0.86      1033
           1       0.82      0.95      0.88      1037

    accuracy                           0.87      2070
   macro avg       0.88      0.87      0.87      2070
weighted avg       0.88      0.87      0.87      2070


The Accuracy of Decision Tree is 87.101 %
ROC AUC Score with Decision Tree: 0.8708674493871946
```

Building and evaluating a Random Forest using the **RandomForestClassifier**

```
rfc = RandomForestClassifier(n_estimators = 100, random_state = 42)
rfc.fit(train_X,train_y)
```

```
C:\Users\MODUPE~1\AppData\Local\Temp/ipykernel_38084/3021051387.py:2: DataCon
versionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples,), for example using ravel().
  rfc.fit(train_X,train_y)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
rfc_pred = rfc.predict(test_X)
rfc_conf = confusion_matrix(test_y, rfc_pred)
rfc_report = classification_report(test_y, rfc_pred)
rfc_acc = round(accuracy_score(test_y, rfc_pred)*100, ndigits = 2)
rfc_rocauc=roc_auc_score(test_y, rfc_pred)
print(f"Confusion Matrix : \n\n{rfc_conf}")
print(f"\nClassification Report : \n\n{rfc_report}")
print(f"\nThe Accuracy of Random Forest Classifier is {rfc_acc} %")
print(f'ROC AUC score wiht Random Forest Classifier: {rfc_rocauc}')
```

```
Confusion Matrix :

[[858 175]
 [ 50 987]]

Classification Report :

              precision    recall  f1-score   support

           0       0.94      0.83      0.88      1033
           1       0.85      0.95      0.90      1037

    accuracy                           0.89      2070
   macro avg       0.90      0.89      0.89      2070
weighted avg       0.90      0.89      0.89      2070


The Accuracy of Random Forest Classifier is 89.13 %
ROC AUC score wiht Random Forest Classifier: 0.8911872526770854
```

Building and evaluating a Logistic Regression using the **LogisticRegression**

```
lr=LogisticRegression()
lr.fit(train_X,train_y)
```

```
C:\Users\Modupe Olayinka\anaconda3\lib\site-packages\sklearn\utils\validatio
n.py:1143: DataConversionWarning: A column-vector y was passed when a 1d arra
y was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Modupe Olayinka\anaconda3\lib\site-packages\sklearn\linear_model\_lo
gistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

```
▾ LogisticRegression
LogisticRegression()
```

```
lr_pred = lr.predict(test_X)
lr_conf = confusion_matrix(test_y, lr_pred)
lr_report = classification_report(test_y, lr_pred)
lr_acc = round(accuracy_score(test_y, lr_pred)*100, ndigits = 2)
lr_rocauc=roc_auc_score(test_y, lr_pred)
print(f"Confusion Matrix : \n\n{lr_conf}")
print(f"\nClassification Report : \n\n{lr_report}")
print(f"\nThe Accuracy of Logistic Regresion is {lr_acc} %")
print(f'ROC AUC score wiht Logistic Regresion: {lr_rocauc}')
```

```
Confusion Matrix :

[[742 291]
 [203 834]]

Classification Report :

              precision    recall  f1-score   support

           0       0.79      0.72      0.75      1033
           1       0.74      0.80      0.77      1037

    accuracy                           0.76      2070
   macro avg       0.76      0.76      0.76      2070
weighted avg       0.76      0.76      0.76      2070


The Accuracy of Logistic Regresion is 76.14 %
ROC AUC score wiht Logistic Regresion: 0.7612696166337292
```

Building and evaluating a Gradient Boost using the **GradientBoostingClassifier.**

```
gradien=GradientBoostingClassifier()

gradien.fit(train_X,train_y)
```

```
C:\Users\Modupe Olayinka\anaconda3\lib\site-packages\sklearn\ensemble\_gb.py:
437: DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using ra
vel().
  y = column_or_1d(y, warn=True)
```

```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

```
gradien_pred=gradien.predict(test_X)
gradien_conf=confusion_matrix(test_y,gradien_pred)
gradien_report=classification_report(test_y,gradien_pred)
gradien_acc=round(accuracy_score(test_y,gradien_pred)*100,ndigits=3)
gradien_rocauc=roc_auc_score(test_y, gradien_pred)
print(f"Confusion Matrix : \n\n{gradien_conf}")
print(f"\nClassification Report : \n\n{gradien_report}")
print(f"\nThe Accuracy of Gradien Boost is {gradien_acc} %")
print(f'ROC AUC score wiht Gradien Boost: {gradien_rocauc}')
```

```
Confusion Matrix :

[[754 279]
 [167 870]]

Classification Report :

              precision    recall  f1-score   support

           0       0.82      0.73      0.77      1033
           1       0.76      0.84      0.80      1037

    accuracy                           0.78      2070
   macro avg       0.79      0.78      0.78      2070
weighted avg       0.79      0.78      0.78      2070


The Accuracy of Gradien Boost is 78.454 %
ROC AUC score wiht Gradien Boost: 0.784435704677186
```

Building and evaluating SGD using the **SGDClassifier.**

```
sgd=SGDClassifier()

sgd.fit(train_X,train_y)
```

```
C:\Users\Modupe Olayinka\anaconda3\lib\site-packages\sklearn\utils\validatio
n.py:1143: DataConversionWarning: A column-vector y was passed when a 1d arra
y was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
  y = column_or_1d(y, warn=True)
```

```
▾ SGDClassifier
SGDClassifier()
```

```
sgd_pred=sgd.predict(test_X)
sgd_conf=confusion_matrix(test_y,sgd_pred)
sgd_report=classification_report(test_y,sgd_pred)
sgd_acc=round(accuracy_score(test_y,sgd_pred)*100,ndigits=3)
sgd_rocauc=roc_auc_score(test_y, sgd_pred)
print(f"Confusion Matrix : \n\n{sgd_conf}")
print(f"\nClassification Report : \n\n{sgd_report}")
print(f"\nThe Accuracy of SGD is {sgd_acc} %")
print(f'ROC AUC Score with SGD: {sgd_rocauc}')
```

```
Confusion Matrix :

[[890 143]
 [525 512]]

Classification Report :

              precision    recall  f1-score   support

           0       0.63      0.86      0.73      1033
           1       0.78      0.49      0.61      1037

    accuracy                           0.68      2070
   macro avg       0.71      0.68      0.67      2070
weighted avg       0.71      0.68      0.67      2070


The Accuracy of SGD is 67.729 %
ROC AUC Score with SGD: 0.6776500834094925
```

Building and evaluating Gaussian Naïve Bayes using the **GaussianNB.**

```
gnb=GaussianNB()

gnb.fit(train_X,train_y)
```

```
C:\Users\Modupe Olayinka\anaconda3\lib\site-packages\sklearn\utils\validatio
n.py:1143: DataConversionWarning: A column-vector y was passed when a 1d arra
y was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
  y = column_or_1d(y, warn=True)
```

```
▾ GaussianNB
GaussianNB()
```

```
gnb_pred=gnb.predict(test_X)
gnb_conf=confusion_matrix(test_y,gnb_pred)
gnb_report=classification_report(test_y,gnb_pred)
gnb_acc=round(accuracy_score(test_y,gnb_pred)*100,ndigits=3)
gnb_rocauc=roc_auc_score(test_y, gnb_pred)
print(f"Confusion Matrix : \n\n{gnb_conf}")
print(f"\nClassification Report : \n\n{gnb_report}")
print(f"\nThe Accuracy of Gaussian is {gnb_acc} %")
print(f'ROC AUC Score with Gaussian Naive Bayes: {gnb_rocauc}',)
```

```
Confusion Matrix :

[[730 303]
 [227 810]]

Classification Report :

              precision    recall  f1-score   support

           0       0.76      0.71      0.73      1033
           1       0.73      0.78      0.75      1037

    accuracy                           0.74      2070
   macro avg       0.75      0.74      0.74      2070
weighted avg       0.75      0.74      0.74      2070


The Accuracy of Gaussian is 74.396 %
ROC AUC Score with Gaussian Naive Bayes: 0.7438894495160195
```

Building and evaluating XGBoost using the **XGBClassifier.**

```
xgboost=XGBClassifier(objective='binary:logistic',eval_metric = 'auc', n_jobs=
xgboost.fit(train_X,train_y)
```

```
XGBClassifier
             eval_metric='auc', gamma=0, gpu_id=-1, grow_policy='depthwis
e',
             importance_type=None, interaction_constraints='',
             learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
             max_delta_step=0, max_depth=6, max_leaves=0, min_child_weigh
t=1,
             missing=nan, monotone_constraints='()', n_estimators=100,
             n_jobs=-1, num_parallel_tree=1, predictor='auto', random_sta
te=0,
             reg_alpha=0, reg_lambda=1, ...)
```

```
xgboost_pred=xgboost.predict(test_X)
xgboost_conf=confusion_matrix(test_y,xgboost_pred)
xgboost_report=classification_report(test_y,xgboost_pred)
xgboost_acc=round(accuracy_score(test_y,xgboost_pred)*100,ndigits=3)
xgboost_rocauc=roc_auc_score(test_y, xgboost_pred)
print(f"Confusion Matrix : \n\n{xgboost_conf}")
print(f"\nClassification Report : \n\n{xgboost_report}")
print(f"\nThe Accuracy of XGB is {xgboost_acc} %")
print(f'ROC AUC Score with XGBOOST: {xgboost_rocauc}')
```

```
Confusion Matrix :

[[809 224]
 [ 98 939]]

Classification Report :

              precision    recall  f1-score   support

           0       0.89      0.78      0.83      1033
           1       0.81      0.91      0.85      1037

    accuracy                           0.84      2070
   macro avg       0.85      0.84      0.84      2070
weighted avg       0.85      0.84      0.84      2070


The Accuracy of XGB is 84.444 %
ROC AUC Score with XGBOOST: 0.8443262408037184
```

Building and evaluating Support Vector Machine using the **SVClassifier.**

```
SVM=svm.SVC()

SVM.fit(train_X,train_y)
```

```
C:\Users\Modupe Olayinka\anaconda3\lib\site-packages\sklearn\utils\validatio
n.py:1143: DataConversionWarning: A column-vector y was passed when a 1d arra
y was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
  y = column_or_1d(y, warn=True)
```

```
▾ SVC
SVC()
```

```python
SVM_pred=SVM.predict(test_X)
SVM_conf=confusion_matrix(test_y,SVM_pred)
SVM_report=classification_report(test_y,SVM_pred)
SVM_acc=round(accuracy_score(test_y,SVM_pred)*100,ndigits=3)
SVM_rocauc=roc_auc_score(test_y, SVM_pred)
print(f"Confusion Matrix : \n\n{SVM_conf}")
print(f"\nClassification Report : \n\n{SVM_report}")
print(f"\nThe Accuracy of SVM is {SVM_acc} %")
print(f'ROC AUC Score with SVM: {SVM_rocauc}')
```

```
Confusion Matrix :

[[723 310]
 [436 601]]

Classification Report :

              precision    recall  f1-score   support

           0       0.62      0.70      0.66      1033
           1       0.66      0.58      0.62      1037

    accuracy                           0.64      2070
   macro avg       0.64      0.64      0.64      2070
weighted avg       0.64      0.64      0.64      2070


The Accuracy of SVM is 63.961 %
ROC AUC Score with SVM: 0.6397298036539613
```

**Steps 7 and 8: Present your solution, Launch, Monitor, and Maintain your system.**

```
results = pd.DataFrame([["XGBoost Classifier", xgboost_acc, xgboost_rocauc],
                        ["Decision Tree Classifier", dtr_acc, dtr_rocauc],
                        ["Gaussian naive bayes classifier", gnb_acc, gnb_rocau
                        ["Gradien Boost Classifier", gradien_acc, gradien_roca
                        ["Random Forest Classifier", rfc_acc, rfc_rocauc],
                        ["Logistic Regression",lr_acc,lr_rocauc],
                        ["Support Vector Machine",SVM_acc,SVM_rocauc]],
                        columns = ["Models", "Testing Accuracy Score", "ROC AU
results.sort_values(by=['Testing Accuracy Score'], ascending=False).style.back
```

| | Models | Testing Accuracy Score | ROC AUC Score |
|---|---|---|---|
| **4** | Random Forest Classifier | 89.130000 | 0.891187 |
| **1** | Decision Tree Classifier | 87.101000 | 0.870867 |
| **0** | XGBoost Classifier | 84.444000 | 0.844326 |
| **3** | Gradien Boost Classifier | 78.454000 | 0.784436 |
| **5** | Logistic Regression | 76.140000 | 0.761270 |
| **2** | Gaussian naive bayes classifier | 74.396000 | 0.743889 |
| **6** | Support Vector Machine | 63.961000 | 0.639730 |

The models include logistic regression, decision trees, random forest, XGBoost, and more. The evaluation metric used to compare the performance of the models is the accuracy score, which measures a model's accuracy, and the ROC AUC Score.

Based on these results, we can see that the logistic regression, random forest, XGBoost models, decision tree models, and random forest model all have great performance, with accuracy scores of 78-89. The support vector machine model, on the other hand, has lower performance, with an accuracy score of only 63.

In conclusion, our machine learning models were able to accurately predict customer churn and provide insights into potential retention strategies for the telecommunications company. The random forest model had the best overall performance. Still, the decision tree and XGBoost models are also viable options depending on the specific needs and constraints of the company. Using these models and insights, the company can take proactive steps to reduce churn and improve customer retention, increasing revenue and customer satisfaction.