Problem Statement : Customer churn prediction refers to the process of identifying customers who are likely to stop using a product or service in the near future. It is a valuable predictive analytics technique used by businesses to forecast customer behavior and take proactive measures to retain customers.

Objective : objective of this project is to predict wather customer is about to churn or not.

Kaggle Dataset Link : [https://www.kaggle.com/datasets/blastchar/telco-customer-churn (https://www.kaggle.com/datasets/blastchar/telco-customer-churn)](https://www.kaggle.com/datasets/blastchar/telco-customer-churn)

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import pickle
from matplotlib import pyplot as plt
import scipy
from sklearn.model_selection import train_test_split,RandomizedSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score,
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.feature_selection import SelectKBest
from collections import Counter
from imblearn.combine import SMOTEENN
plt.style.use('default')
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
#import the dataset
df=pd.read_csv(r"Telco-Customer-Churn.csv")
df.head()
```

Out[2]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetS
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	Fib

5 rows × 21 columns

In [3]:
```python
#print concise summary of the dataset
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
```

In [4]:
```python
#check for missing values
df.isnull().sum()
```

Out[4]:
```
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
```

In [5]:
```python
#check for duplicate records
df[df.duplicated()].shape[0]
```

Out[5]:
```
0
```

In [6]:
```
#check datatype
df.dtypes
```

Out[6]:
```
customerID           object
gender               object
SeniorCitizen         int64
Partner              object
Dependents           object
tenure                int64
PhoneService         object
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
MonthlyCharges      float64
TotalCharges         object
Churn                object
dtype: object
```

In [7]:
```
#since customerId is not required for prediction so drop it
df.drop('customerID',axis=1,inplace=True)
```

In [8]:
```
#since total changes is having numerical value but dtype is object to change it i
df['TotalCharges']=pd.to_numeric(df['TotalCharges'],errors='coerce')
```

In [9]:
```
#print last 5 records of the  dataset
df.tail(5)
```

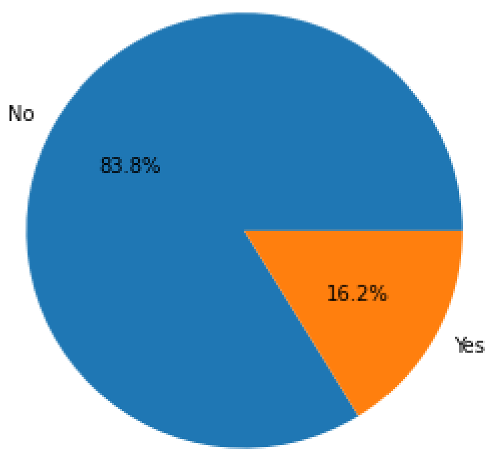
Out[9]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	O
7038	Male	0	Yes	Yes	24	Yes	Yes	DSL	
7039	Female	0	Yes	Yes	72	Yes	Yes	Fiber optic	
7040	Female	0	Yes	Yes	11	No	No phone service	DSL	
7041	Male	1	Yes	No	4	Yes	Yes	Fiber optic	
7042	Male	0	No	No	66	Yes	No	Fiber optic	

Exploratory Data Analysis :

In [10]:

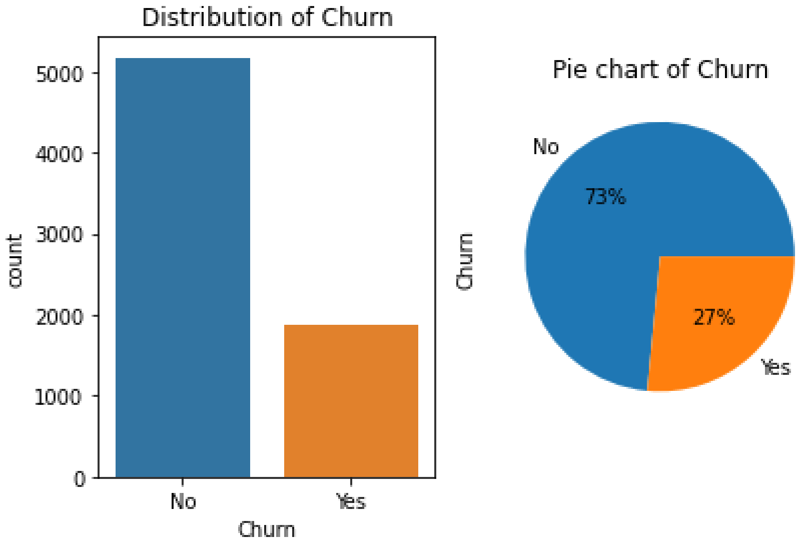
```
#pie chart to count senior citizen
plt.figure(figsize=(10,5))
plt.pie(df["SeniorCitizen"].value_counts(),autopct="%.1f%%",labels=["No","Yes"])
plt.show()
```



as we can see 83.8 % of the customers are senior citizen and only 16.2% are adult customer.

In [11]:

```
#check the distibution of churn class
plt.subplot(121)
sns.countplot(data=df,x="Churn")
plt.title("Distribution of Churn")
plt.subplot(122)
df['Churn'].value_counts().plot(kind='pie',autopct="%1.f%%",labels=['No','Yes'])
plt.title('Pie chart of Churn')
plt.tight_layout()
plt.show()
```



In [12]:

```
#perentage of each class sample distribution
print("Customer Churn : {}%".format(np.round((len(df[df["Churn"]=="Yes"])/len(df)
print("Customer Not Churn : {}%".format(np.round((len(df[df["Churn"]=="No"])/len(
```

```
Customer Churn : 26.54%
Customer Not Churn : 73.46%
```

Since our dataset is highly imbalance we need to balance before fitting it into model

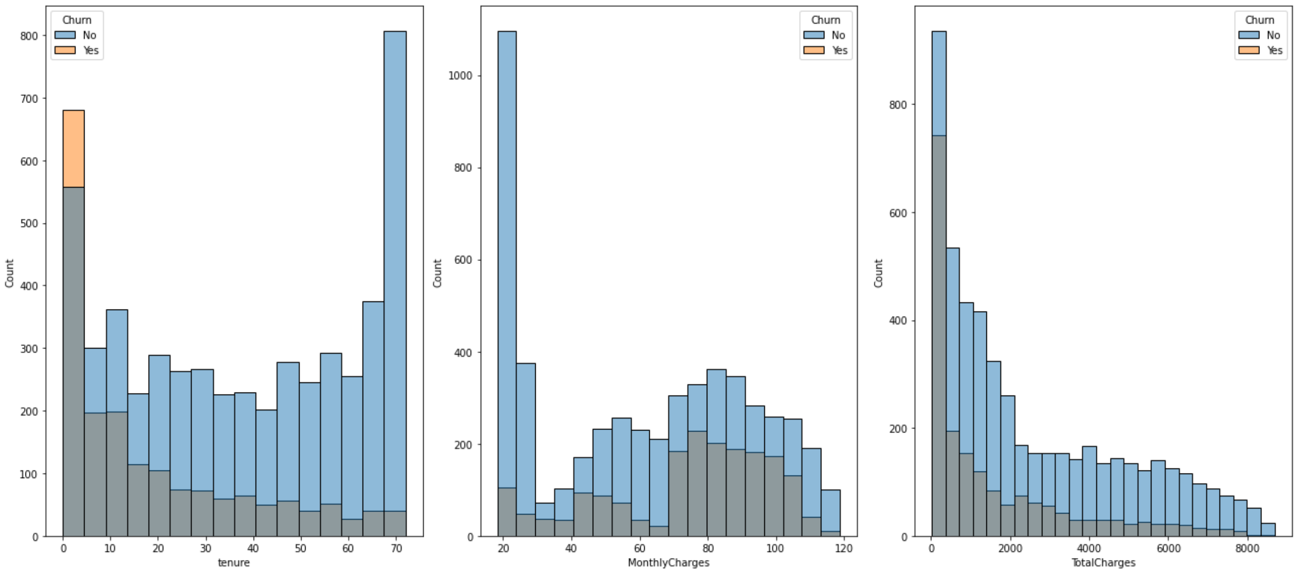
In [13]:

```
#how much loss we are having because of customer churn
churn_customers=df[df["Churn"]=="Yes"]
loss=churn_customers["TotalCharges"].sum()
total_revenue=df["TotalCharges"].sum()
print("We have lost arround {}$ due to customer churn".format(loss))
print("We have lost arround {} percentage of revengue due to customer churn".form
```

```
We have lost arround 2862926.9$ due to customer churn
We have lost arround 17.83 percentage of revengue due to customer churn
```

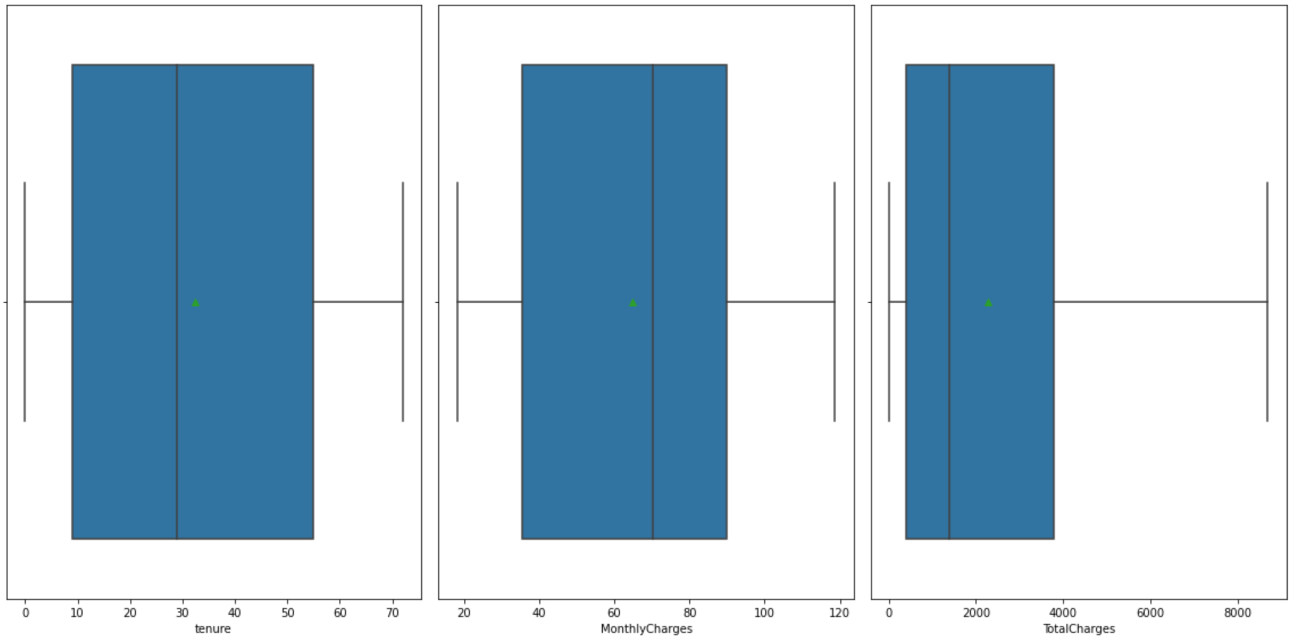
In [14]:

```
#plot numerical features with histogram
fig,axs=plt.subplots(nrows=1,ncols=3,figsize=(18,8))
axes=axs.flatten()
num_columns=['tenure', 'MonthlyCharges', 'TotalCharges']
for i,col in enumerate(num_columns):
  if(col!='SeniorCitizen'):
    sns.histplot(x=col,data=df,hue='Churn',ax=axes[i])
fig.tight_layout()
plt.show()
```



In [15]:

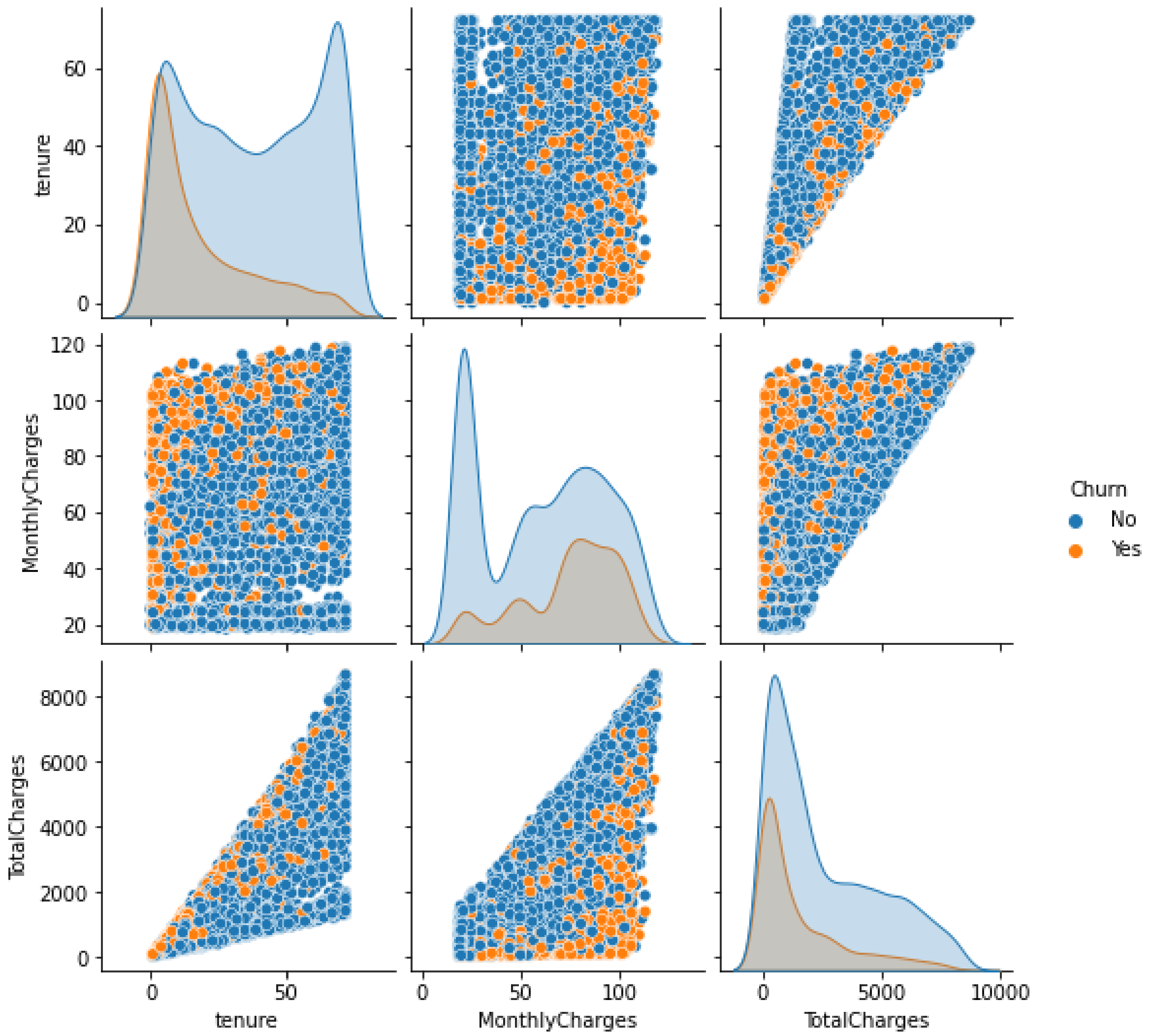
```
#plot numerical features with boxplot
fig,axs=plt.subplots(nrows=1,ncols=3,figsize=(16,8))
axes=axs.flatten()
num_columns=['tenure', 'MonthlyCharges', 'TotalCharges']
for i,col in enumerate(num_columns):
  if(col!='SeniorCitizen'):
    sns.boxplot(x=col,data=df,showmeans=True,ax=axes[i])
fig.tight_layout()
plt.show()
```



after plotting histogram and boxplot we found that there is no outlier present in numeric dataset so we don't need to do any kind of outlier treatment.

In [16]:

```
sns.pairplot(df.drop(columns="SeniorCitizen"),hue="Churn",kind="scatter")
plt.show()
```

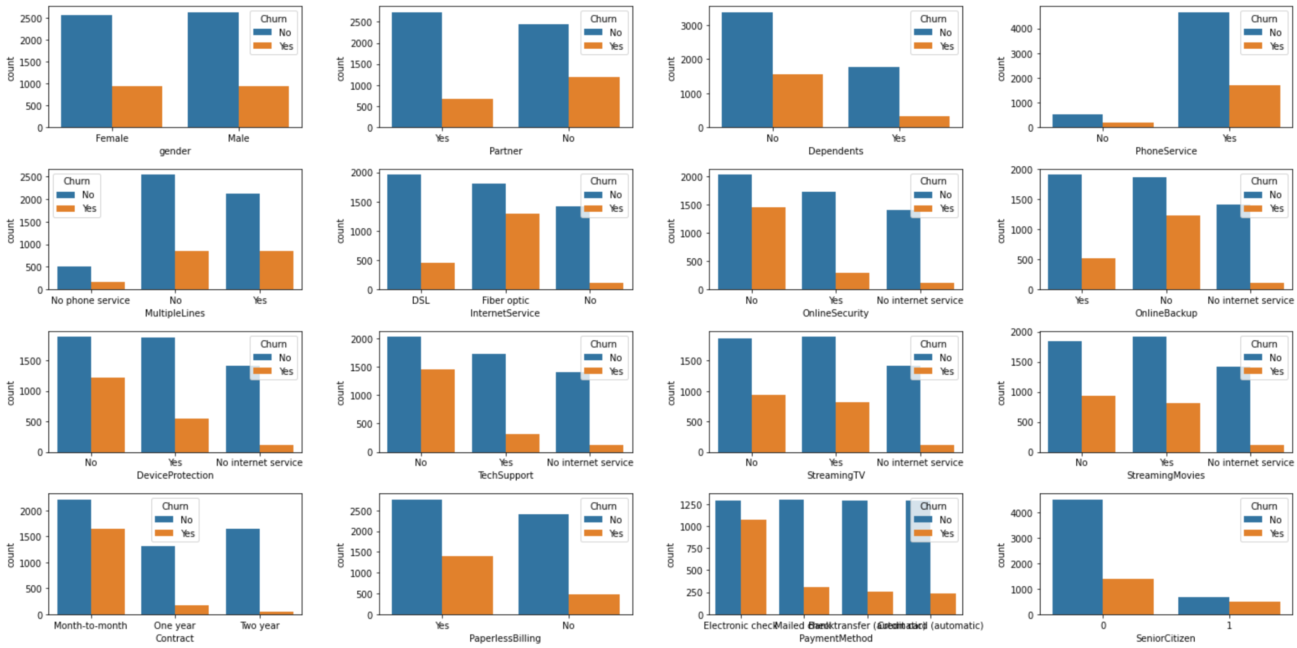


Univariate Analysis

In [17]:

```python
#plot cateogrical features :
cat_features=list(df.select_dtypes(include='object').columns)
cat_features.remove('Churn')
cat_features.append('SeniorCitizen')

fig,axs=plt.subplots(nrows=4,ncols=4,figsize=(20,10))
axes=axs.flatten()
for i,col in enumerate(cat_features):
    sns.countplot(x=col,hue="Churn",data=df,ax=axes[i])
#adjust spacing between subplots
fig.tight_layout()
plt.show()
```



Data Cleaning

In [18]:

```python
df.head(5)
```

Out[18]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	Onlir
0	Female	0	Yes	No	1	No	No phone service	DSL	
1	Male	0	No	No	34	Yes	No	DSL	
2	Male	0	No	No	2	Yes	No	DSL	
3	Male	0	No	No	45	No	No phone service	DSL	
4	Female	0	No	No	2	Yes	No	Fiber optic	

In [19]:
```
#check for null values
df.isnull().sum()
```

Out[19]:
```
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges       11
Churn               0
dtype: int64
```

In [20]:
```
df["TotalCharges"].fillna(df["TotalCharges"].mean(),inplace=True)
```

In [21]:
```
df.isnull().sum().sum()
```

Out[21]:
```
0
```

In [22]:
```
#encoding categorical values into numeric using label encoder
encoder=LabelEncoder()
for feature in df.select_dtypes(include='object').columns:
    df[feature]=encoder.fit_transform(df[feature])
```

In [23]:
```
df.head()
```

Out[23]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	Onlir
0	0	0	1	0	1	0	1	0	
1	1	0	0	0	34	1	0	0	
2	1	0	0	0	2	1	0	0	
3	1	0	0	0	45	0	1	0	
4	0	0	0	0	2	1	0	1	

In [24]:

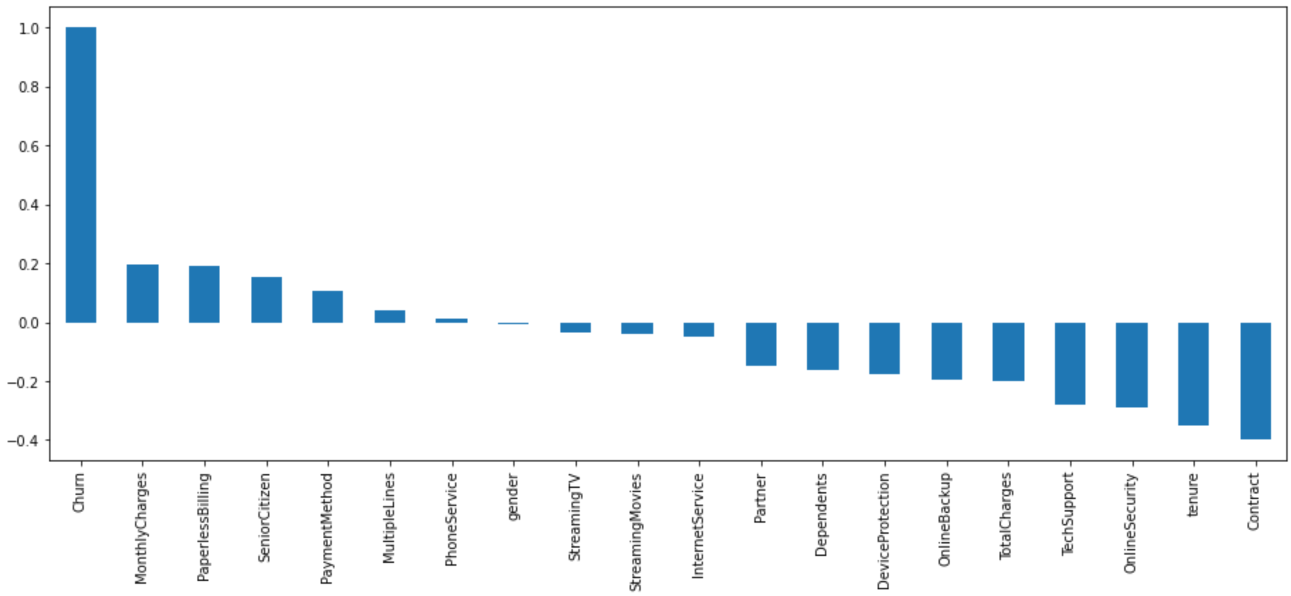
```
df.dtypes
```

Out[24]:

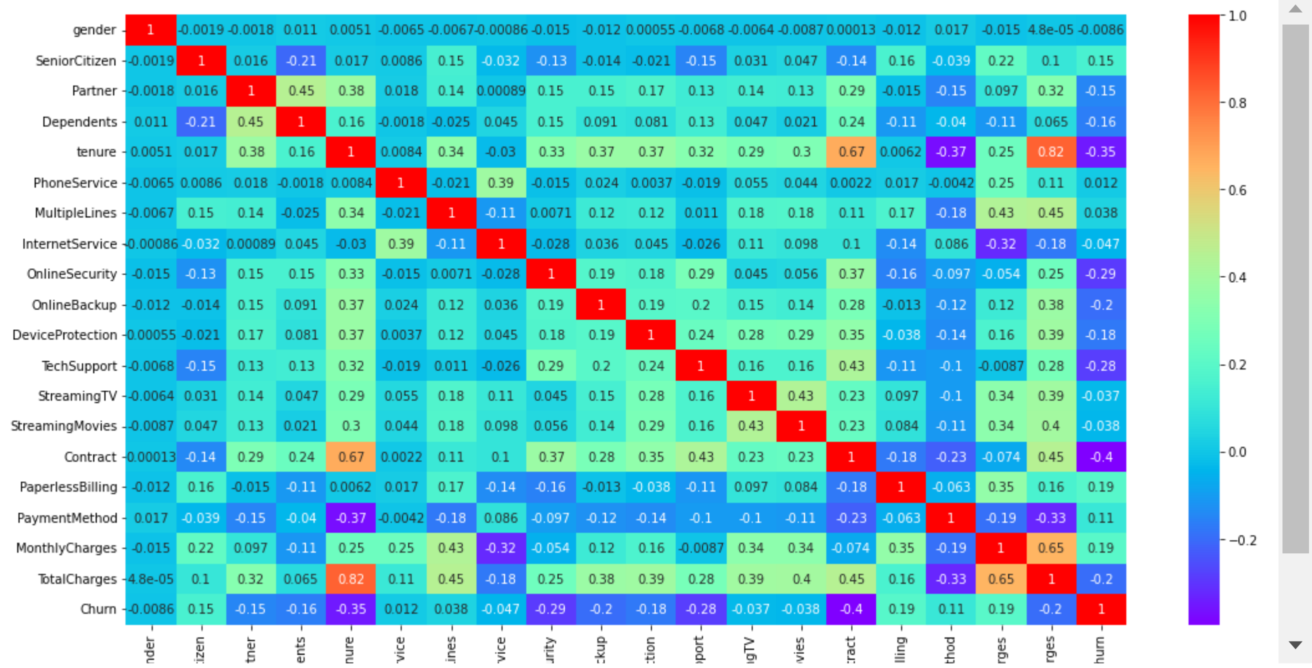
```
gender                int32
SeniorCitizen         int64
Partner               int32
Dependents            int32
tenure                int64
PhoneService          int32
MultipleLines         int32
InternetService       int32
OnlineSecurity        int32
OnlineBackup          int32
DeviceProtection      int32
TechSupport           int32
StreamingTV           int32
StreamingMovies       int32
Contract              int32
PaperlessBilling      int32
PaymentMethod         int32
MonthlyCharges      float64
TotalCharges        float64
Churn                 int32
dtype: object
```

In [25]:

```
#get correlation of churn with other variables
plt.figure(figsize=(16,6))
df.corr()["Churn"].sort_values(ascending=False).plot(kind="bar")
plt.show()
```



In [26]:
```python
plt.figure(figsize=(18,9))
sns.heatmap(df.corr(),annot=True,cmap="rainbow")
plt.show()
```



since we are using ensemble methods for model building so there is no need of feature scaling as its prediction is based on creating multiple decision tree

In [27]:
```python
#seperating independent variables and target variable
x=df.drop("Churn",axis=1)
y=df["Churn"]
```

In [28]:
```python
x.shape
```

Out[28]:
```
(7043, 19)
```

Feature Selection

selecting only 10 features which has higher correlation with churn

In [29]:
```python
select_feature=SelectKBest(k=10) #no of features to be select
select_feature.fit(x,y)
```

Out[29]:
```
▾ SelectKBest
SelectKBest()
```

In [30]:
```python
#Top 10 high correlated features
select_feature.get_feature_names_out()
```

Out[30]:
```
array(['Dependents', 'tenure', 'OnlineSecurity', 'OnlineBackup',
       'DeviceProtection', 'TechSupport', 'Contract', 'PaperlessBilling',
       'MonthlyCharges', 'TotalCharges'], dtype=object)
```

In [31]:
```python
x=x[select_feature.get_feature_names_out()]
```

In [32]:
```
x.shape
```
Out[32]:
```
(7043, 10)
```

according to the feature selection we have selected 10 top features out of 19 features

split data into training and validation set in 80:20 ratio

In [33]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

In [34]:
```
x_train.shape,y_train.shape,x_test.shape,y_test.shape
```
Out[34]:
```
((5634, 10), (5634,), (1409, 10), (1409,))
```

In [35]:
```
#its imbalance dataset
y.value_counts()
```
Out[35]:
```
0    5174
1    1869
Name: Churn, dtype: int64
```

In [36]:
```
def evaluate_model_performance(model,test_data):
    prediction=model.predict(test_data)
    #print("Training Accurary : ",model.score(x_train,y_train))
    print("Validation Accurary : {:.2f} %".format(accuracy_score(y_test,predictic
    print("Precision Score : {:.2f} %".format(precision_score(y_test,prediction))
    print("Recall Score : {:.2f} %".format(recall_score(y_test,prediction)))
    print("F1 Score : {:.2f} %".format(f1_score(y_test,prediction)))
    print(classification_report(y_test,prediction))
```

In [37]:
```
#Random Forest Model without balancing dataset and without hyper paramter tuning
rand_forest=RandomForestClassifier()
rand_forest.fit(x_train,y_train)
```
Out[37]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [38]:
```
#measure the performance of random forest model
evaluate_model_performance(rand_forest,x_test)
```
```
Validation Accurary : 0.79 %
Precision Score : 0.64 %
Recall Score : 0.47 %
F1 Score : 0.54 %
              precision    recall  f1-score   support

           0       0.83      0.90      0.86      1036
           1       0.64      0.47      0.54       373

    accuracy                           0.79      1409
   macro avg       0.73      0.69      0.70      1409
weighted avg       0.78      0.79      0.78      1409
```

In [39]:
```
#GradientBoostingClassifier without balancing dataset and without hyper paramter
gbc_model=GradientBoostingClassifier( )
gbc_model.fit(x_train,y_train)
```

Out[39]:
```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

In [40]:
```
#measure the performance of GradientBoostingClassifier
evaluate_model_performance(gbc_model,x_test)
```

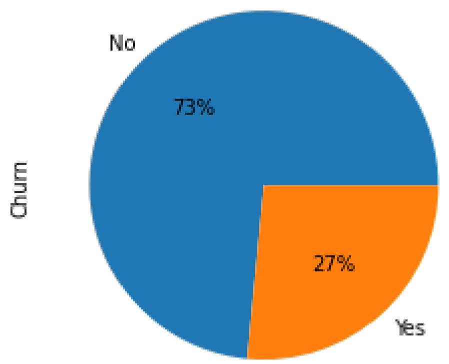
```
Validation Accurary : 0.81 %
Precision Score : 0.67 %
Recall Score : 0.53 %
F1 Score : 0.59 %
              precision    recall  f1-score   support

           0       0.84      0.90      0.87      1036
           1       0.67      0.53      0.59       373

    accuracy                           0.81      1409
   macro avg       0.75      0.72      0.73      1409
weighted avg       0.80      0.81      0.80      1409
```

as we can see our model is not performing up to the mark because of imbalance nature of dataset so we will balance it to reduce TN,FN and increase TP,FP

In [41]:
```
plt.figure(figsize=(8,4))
y.value_counts().plot(kind="pie",autopct="%1.f%%",labels=['No','Yes'])
plt.show()
```



we have 2 classes class 0 and class 1. class 0 - majority class class 1 -minority class

In [42]:
```
smote=SMOTEENN()
x_st,y_st=smote.fit_resample(x,y)
```

In [43]:

```
y_st.value_counts().plot(kind="bar")
plt.title("target class distribution after under sampling")
plt.show()
```

In [44]:

```
y_st.value_counts()
```

Out[44]:

```
1    3101
0    2666
Name: Churn, dtype: int64
```

since we have performed SMOTEENN (combination of Smote + ENN) sampling method and we can see our dataset is nearly balanced

In [45]:

```
#now split training and validation set using balanced dataset
x_train,x_test,y_train,y_test=train_test_split(x_st,y_st,test_size=0.2,random_sta
```

In [46]:

```
x_train.shape,y_train.shape,x_test.shape,y_test.shape
```

Out[46]:

```
((4613, 10), (4613,), (1154, 10), (1154,))
```

Building Model with Balanced Dataset and performance hyper parameter tuning using RandomSearchCV

In [47]:

```
param_grid={'n_estimators':[40,80,120,160,200],
            'max_depth':[2,4,6,8,10],
            "criterion":['gini'],
            "random_state":[27,42,43]
            }
random_search_cv=RandomizedSearchCV( estimator=RandomForestClassifier(), param_di
random_search_cv.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

Out[47]:

```
▸ RandomizedSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [48]:
```
random_search_cv.best_params_
```

Out[48]:
```
{'random_state': 27, 'n_estimators': 160, 'max_depth': 10, 'criterion': 'gini'}
```

In [49]:
```
#Get final model with best param from RandomizedSearchCV
rf_final_model=random_search_cv.best_estimator_
```

In [50]:
```
#evaluate Random Forest Classifier
evaluate_model_performance(rf_final_model,x_test)
```

```
Validation Accurary : 0.97 %
Precision Score : 0.95 %
Recall Score : 0.98 %
F1 Score : 0.97 %
              precision    recall  f1-score   support

           0       0.98      0.95      0.96       541
           1       0.95      0.98      0.97       613

    accuracy                           0.97      1154
   macro avg       0.97      0.97      0.97      1154
weighted avg       0.97      0.97      0.97      1154
```

In [51]:
```
param_grid2 = {'n_estimators':[100, 150, 200, 250, 300],
              'criterion': ['friedman_mse', 'squared_error', 'mse', 'mae'],
               'max_depth': [2,4,6,8],
               'learning_rate': [0.001, 0.01, 0.1, 0.2],
               'loss': ['deviance', 'exponential']
               }
```

In [52]:
```
random_search_cv2=RandomizedSearchCV(estimator=GradientBoostingClassifier(random_
random_search_cv2.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

Out[52]:
```
RandomizedSearchCV
estimator: GradientBoostingClassifier
GradientBoostingClassifier
```

In [53]:
```
random_search_cv2.best_params_
```

Out[53]:
```
{'n_estimators': 250,
 'max_depth': 6,
 'loss': 'deviance',
 'learning_rate': 0.2,
 'criterion': 'mse'}
```

In [54]:
```
gb_final_model=random_search_cv2.best_estimator_
```

In [55]:
```
#evaluate final GradientBoostingClassifier Performance
evaluate_model_performance(gb_final_model,x_test)
```

```
Validation Accurary : 0.97 %
Precision Score : 0.97 %
Recall Score : 0.98 %
F1 Score : 0.97 %
              precision    recall  f1-score   support

           0       0.98      0.96      0.97       541
           1       0.97      0.98      0.97       613

    accuracy                           0.97      1154
   macro avg       0.97      0.97      0.97      1154
weighted avg       0.97      0.97      0.97      1154
```

Save Final Model Integration with application

In [56]:
```
file=open("trained_model.pkl","wb")
pickle.dump(gb_final_model,file)
file.close()
```

Conclusion : after balancing the dataset using smootenn and hyper paramter tuning model performance has increase and the highest f1 score we are getting is 97%.