CUSTOMER CHURN ANALYSIS

1. Problem Definition:

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

2. Data Analysis:

I. Importing important libraries:

```
import pandas as pd
import numpy as np
from numpy import array
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import LabelEncoder,OrdinalEncoder
from sklearn.model_selection import  train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import OrdinalEncoder,power_transform
from sklearn.model_selection import  train_test_split, cross_val_score
from sklearn.metrics import accuracy_score,confusion_matrix,roc_curve,roc_auc_score,classification_report
from sklearn.metrics import plot_roc_curve
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
import warnings
warnings.filterwarnings('ignore')
```

II. Importing data:

Click on this link below ↓

https://raw.githubusercontent.com/dsrscientist/DSData/master/Telecom_customer_churn.csv

```
df=pd.read_csv('https://raw.githubusercontent.com/dsrscientist/DSData/master/Telecom_customer_churn.csv')
df.head()
```

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

5 rows × 21 columns

Checking the shape of the data frame :

```
df.shape
```

```
(7043, 21)
```

checking columns in the dataset:

```
df.columns
```

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

Description of the dataframe:

```
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

Information about the dataframe:

```
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
```

In the dataset there are some columns whose data type is object datatype….so we will be performing some encoding techniques later to transform the features and label .

Now,checking for null values:
There is no null value in the dataset, so we can go ahead

```
df.isnull().sum()
```

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

```
df.isnull().sum().sum()
```

```
0
```

Checking skewness in the dataset

```
df.skew()
```

```
SeniorCitizen     1.833633
tenure            0.239540
MonthlyCharges   -0.220524
dtype: float64
```

there is no skewness in the continuous variables of dataset,so we will move ahead
Columns removed: customerID(not relevant)
No action: SeniorCitizen
Label encoding The following feature is categorical and each take on 2 values (mostly yes/no) — therefore are transformed to binary integers: Churn

```
#using label encoder for categorical target (churn) for encoding
le=LabelEncoder()
df['Churn']=le.fit_transform(df['Churn'])
```

```
le.inverse_transform([0,1])
```

```
array(['No', 'Yes'], dtype=object)
```

One-Hot Encoding The following features are categorical, yet not ordinal (no ranking) but take on more than 2 values. For each value, a new variable is created with a binary integer indicating if the value occured in a data entry or not (1 or 0).

- gender
- Partner
- PhoneService

- PaperlessBilling
- Dependents
- MultipleLines
- InternetService
- OnlineSecurity
- OnlineBackup
- DeviceProtection
- TechSupport
- StreamingTV
- StreamingMovies
- Contract
- PaymentMethod

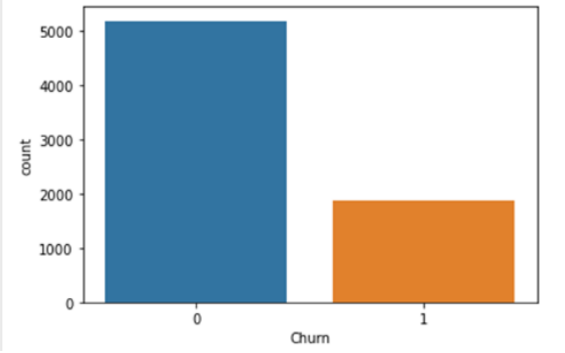
```
enc=OrdinalEncoder()
for i in df.columns:
    if df[i].dtypes=="object":
        df[i]=enc.fit_transform(df[i].values.reshape(-1,1))
df
```

EXPLORATORY DATA ANALYSIS :

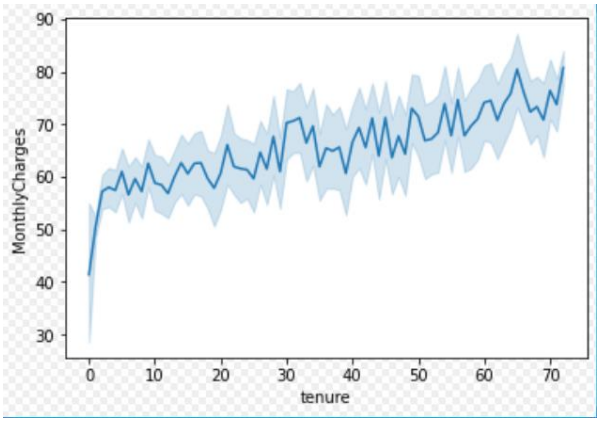
We need to explore the data to find some patterns:
lets see **the count of target variable 'churn':**

```
sns.countplot(df['Churn'])
df['Churn'].value_counts()
```

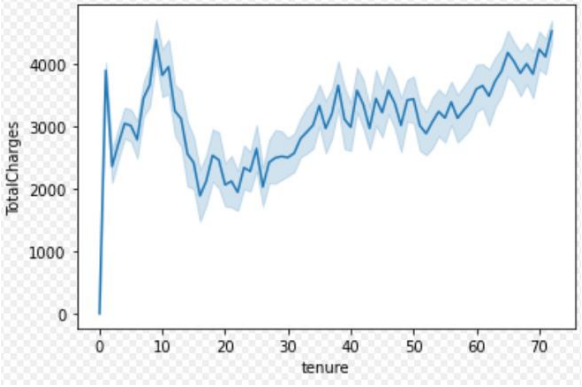
```
0    5174
1    1869
Name: Churn, dtype: int64
```



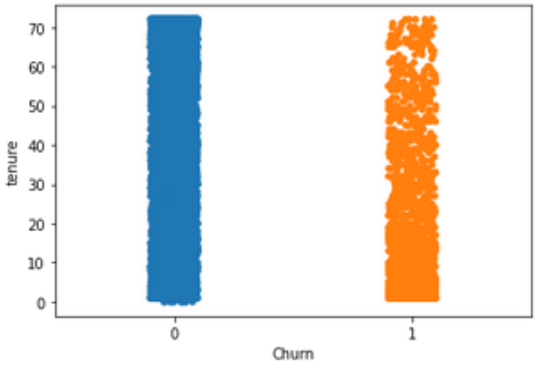
We will plot some seaborn plots for continuous or numerical columns:
sns.lineplot(y='MonthlyCharges',x='tenure', data=df)



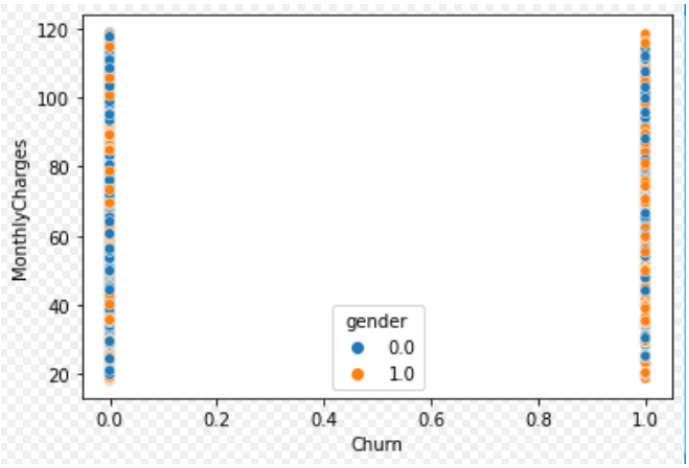
sns.lineplot(y='TotalCharges',x='tenure', data=df)



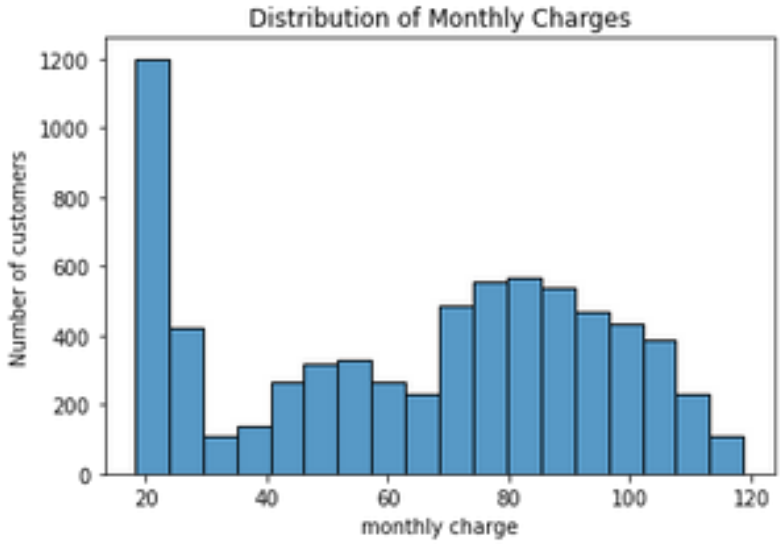
sns.stripplot(x='Churn',y='tenure', data=df)



sns.scatterplot(x="Churn", y="MonthlyCharges", hue="gender", data=df)

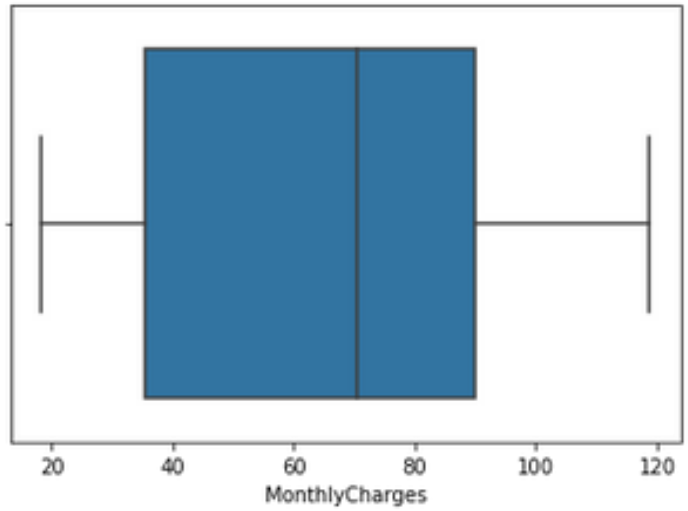



```
sns.countplot(df['MonthlyCharges'])
plt.title('Distribution of Monthly Charges')
plt.xlabel('monthly charge')
plt.ylabel('Number of customers)
plt.show()
```



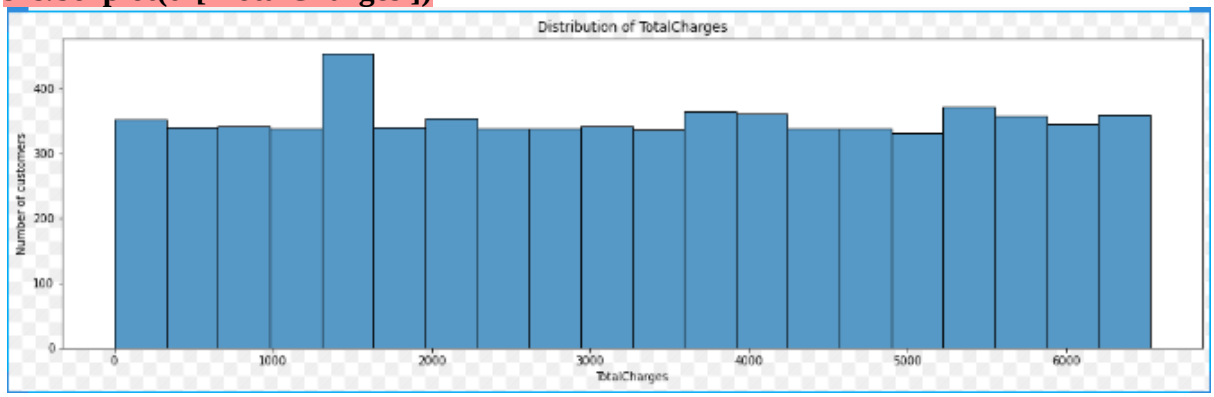
checking outliers in the continuous variables of dataset:

sns.boxplot(df['MonthlyCharges'])

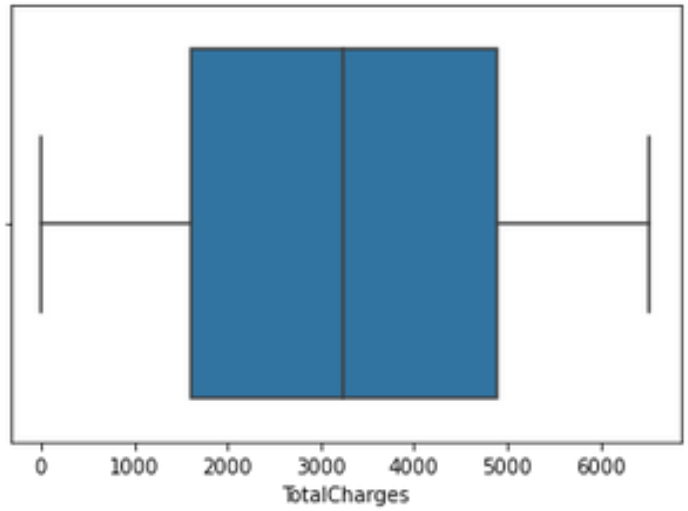


No outliers in numerical features detected with the IQR method — no adjustments made.

```
plt.figure(figsize=(18, 5)
sns.countplot(df['TotalCharges'])
plt.title('Distribution of TotalCharges')
plt.xlabel('TotalCharges')
plt.ylabel('Number of customers')
plt.show()
sns.boxplot(df['TotalCharges'])
```

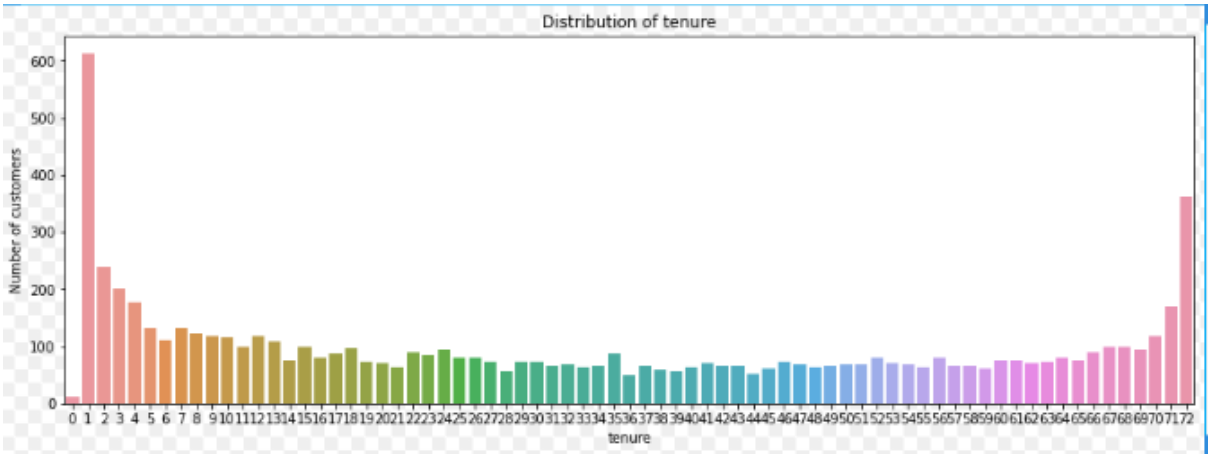



```
sns.boxplot(df['TotalCharges'])
```



No outliers in numerical features detected with the IQR method — no adjustments made.

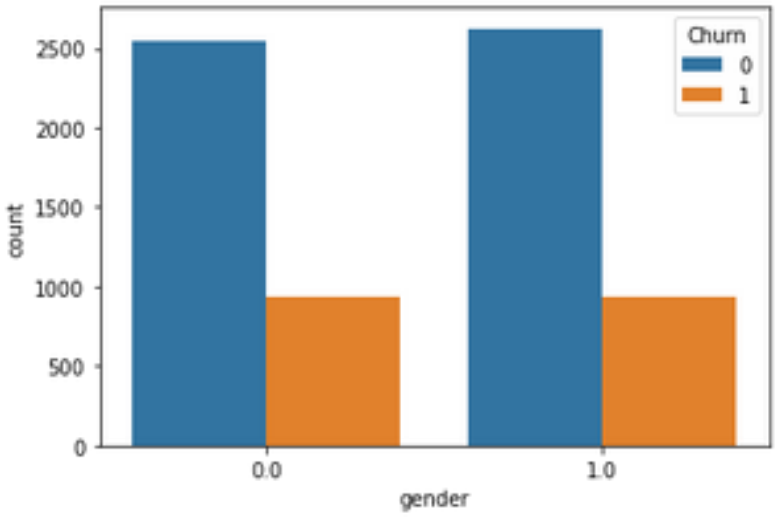
```
plt.figure(figsize=(15, 5))
sns.countplot(df['tenure'])
plt.title('Distribution of tenure')
plt.xlabel('tenure')
plt.ylabel('Number of customers')
plt.show()
```

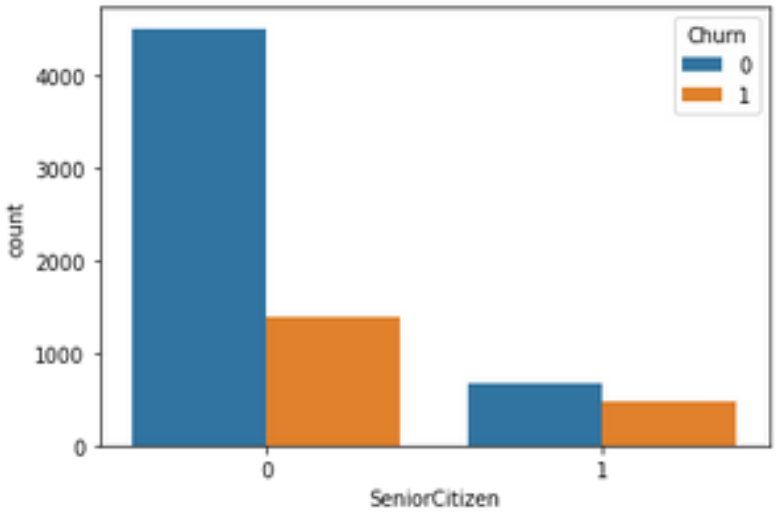


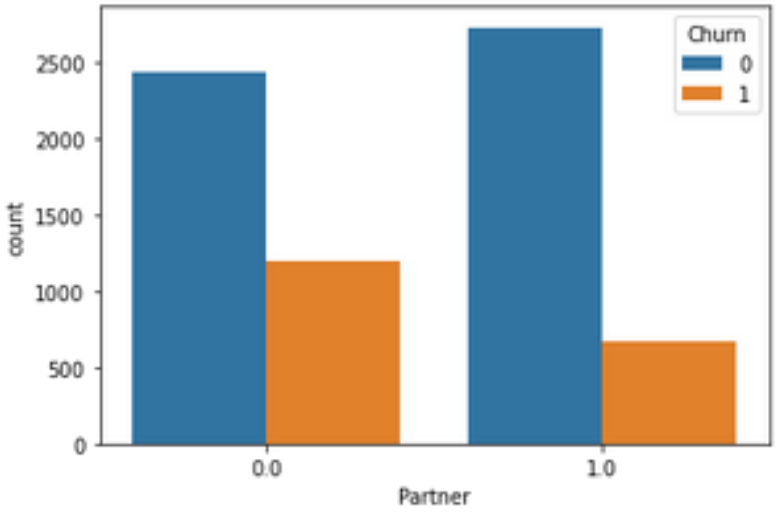
Now,we will plot some seaborn plots for categorical columns:

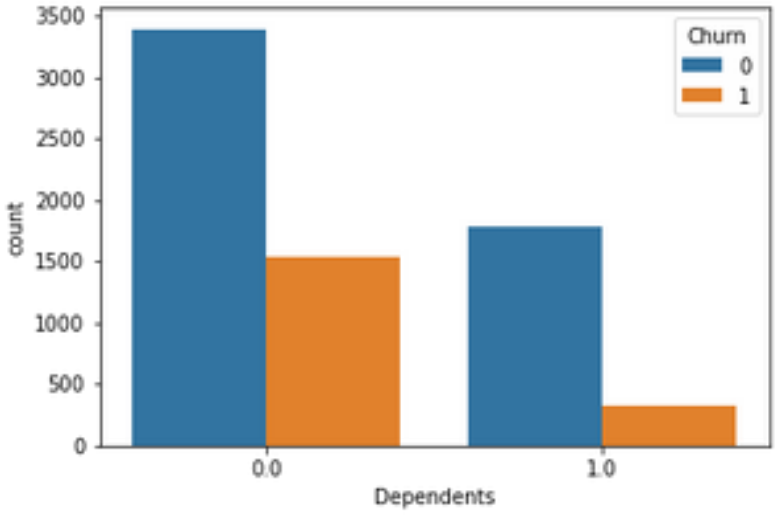
```
for i,predictor in enumerate(df.drop(columns=['Churn', 'TotalCharges','MonthlyCharges'])):

plt.figure(i)

sns.countplot(data=df, x=predictor, hue='Churn'
```

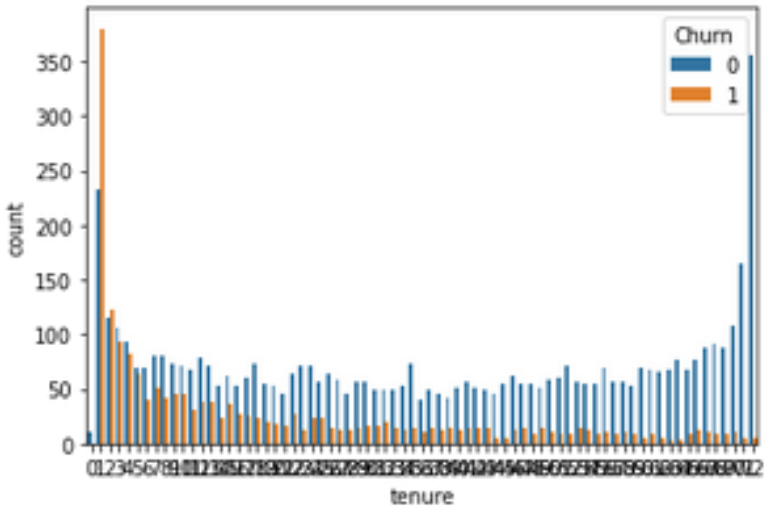
Output:

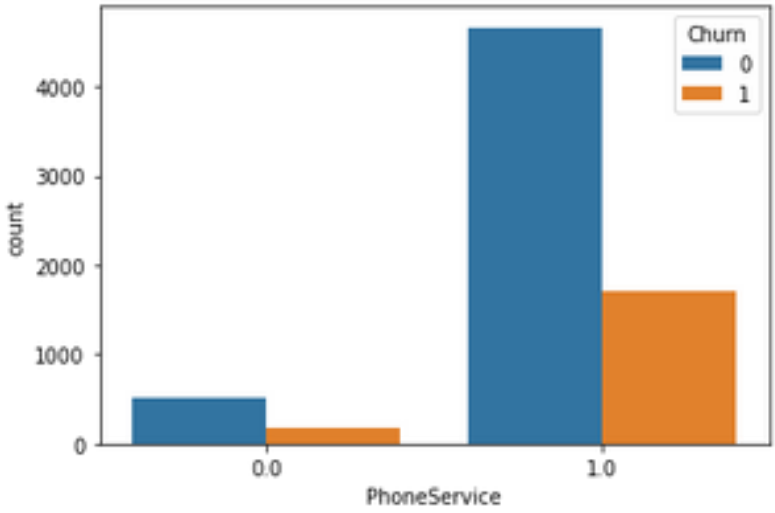


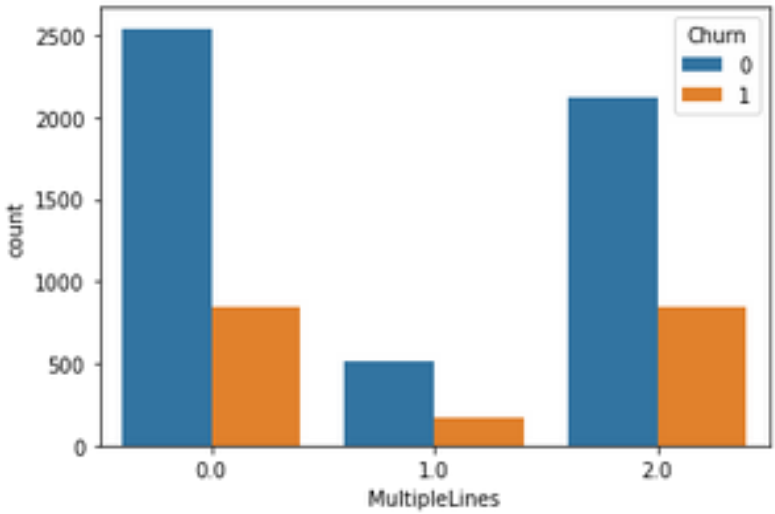

Churn
0
1
4000
3000
2000
1000
0
count
0
1
SeniorCitizen

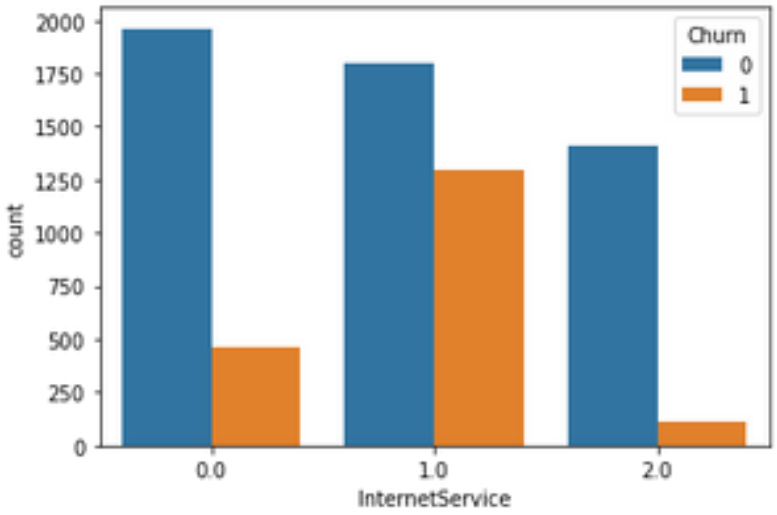

Churn
0
1
2500
2000
1500
1000
500
0
count
0.0
1.0
Partner

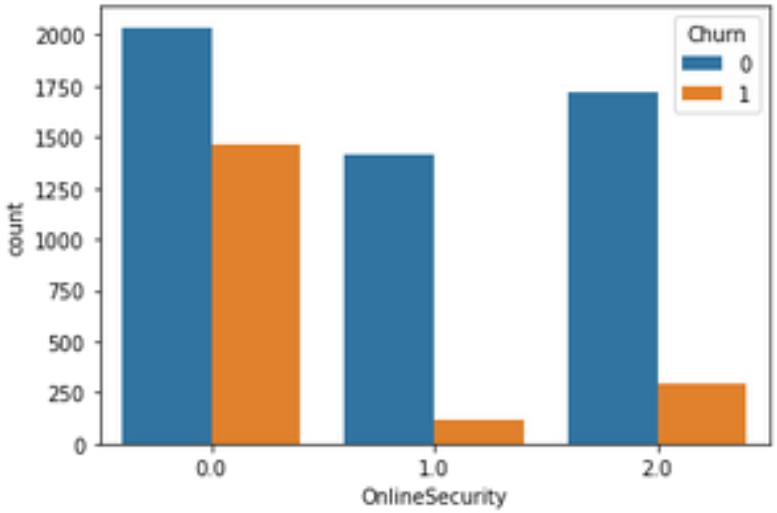

Churn
0
1
3500
3000
2500
2000
1500
1000
500
0
count
0.0
1.0
Dependents

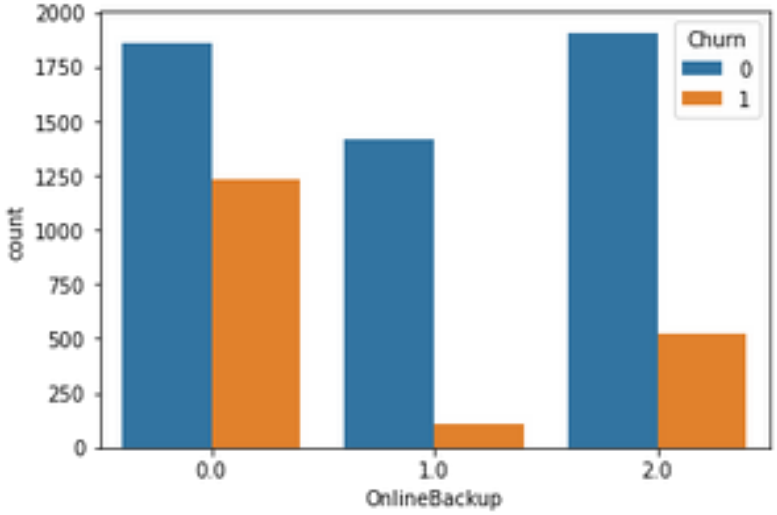

Churn
0
1
count
tenure

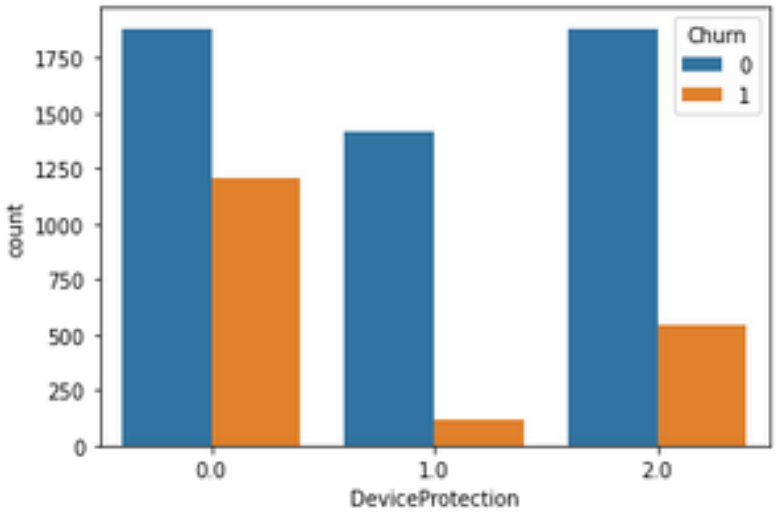

Churn
0
1
count
PhoneService

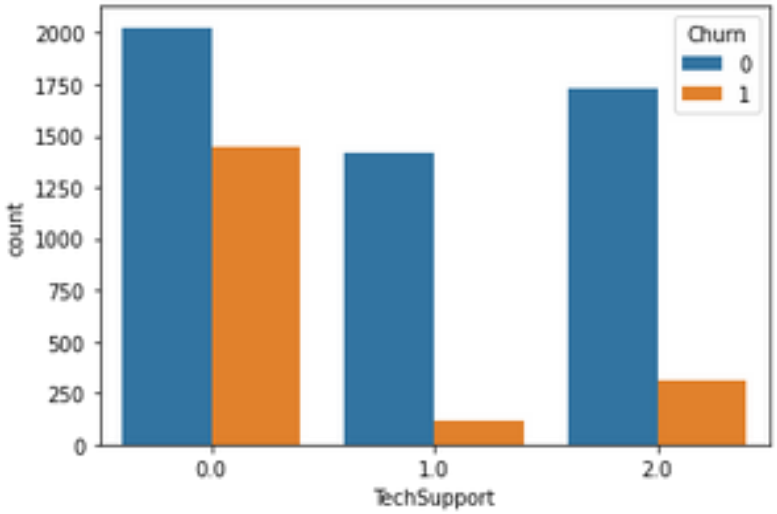

Churn
0
1
count
MultipleLines

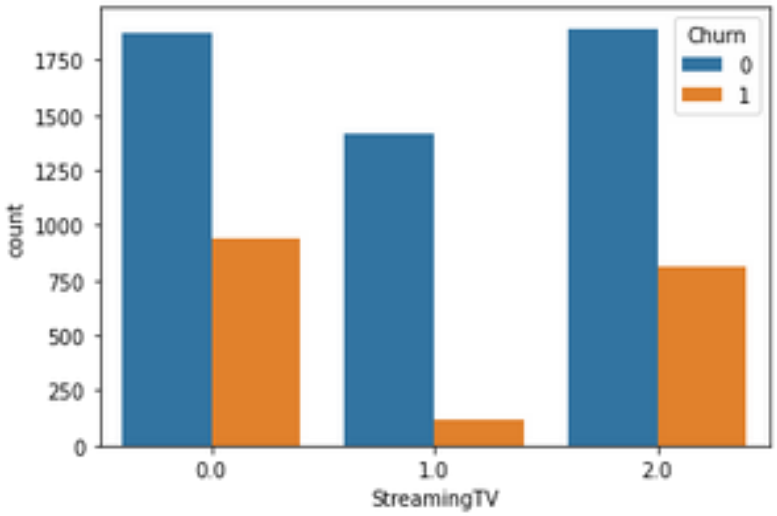

Churn
0
1
count
2000
1750
1500
1250
1000
750
500
250
0
0.0
1.0
2.0
InternetService

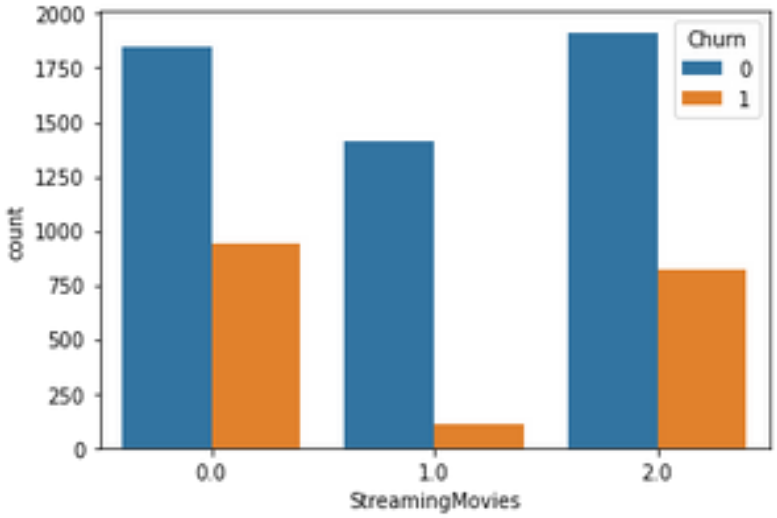

Churn
0
1
count
2000
1750
1500
1250
1000
750
500
250
0
0.0
1.0
2.0
OnlineSecurity

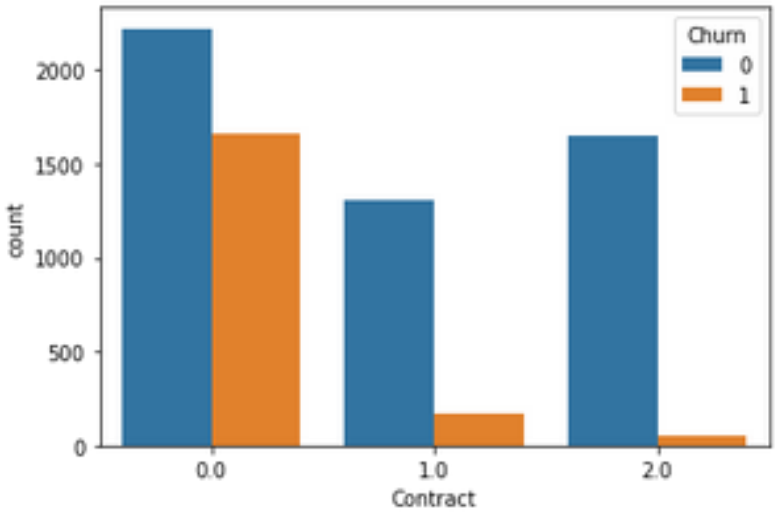

Churn
0
1
count
2000
1750
1500
1250
1000
750
500
250
0
0.0
1.0
2.0
OnlineBackup

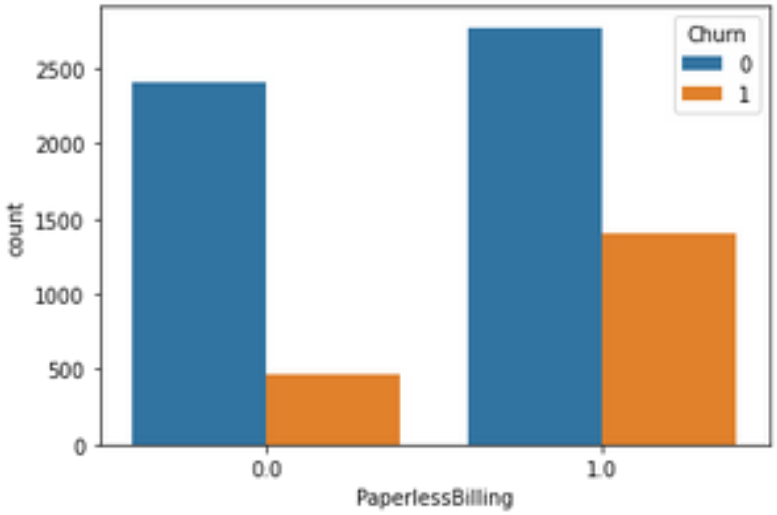

Churn
0
1
count
1750
1500
1250
1000
750
500
250
0
0.0
1.0
2.0
DeviceProtection

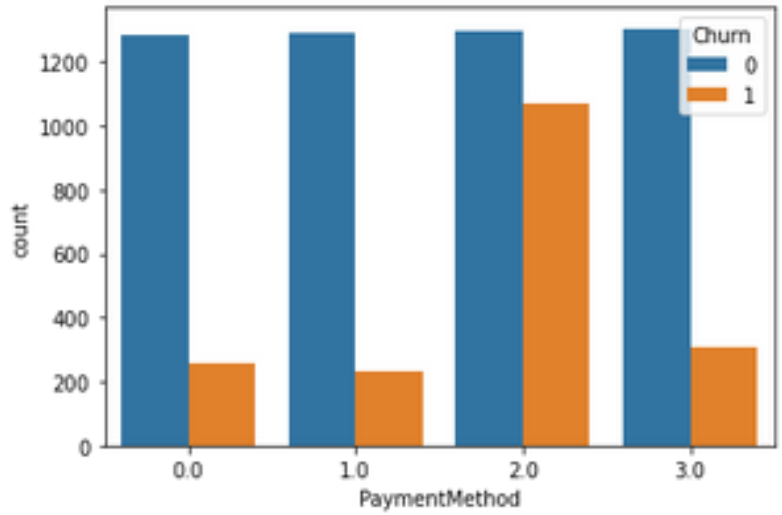

Churn
0
1
count
2000
1750
1500
1250
1000
750
500
250
0
0.0
1.0
2.0
TechSupport


Churn
0
1
count
1750
1500
1250
1000
750
500
250
0
0.0
1.0
2.0
StreamingTV


Churn
0
1
count
2000
1750
1500
1250
1000
750
500
250
0
0.0
1.0
2.0
StreamingMovies


Churn
0
1
count
2000
1500
1000
500
0
0.0
1.0
2.0
Contract

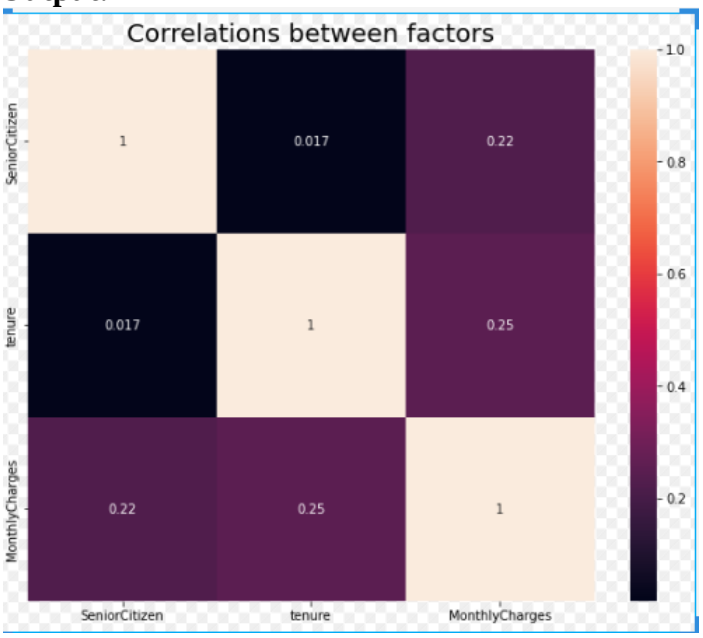

Churn
0
1
count
2500
2000
1500
1000
500
0
0.0
1.0
PaperlessBilling



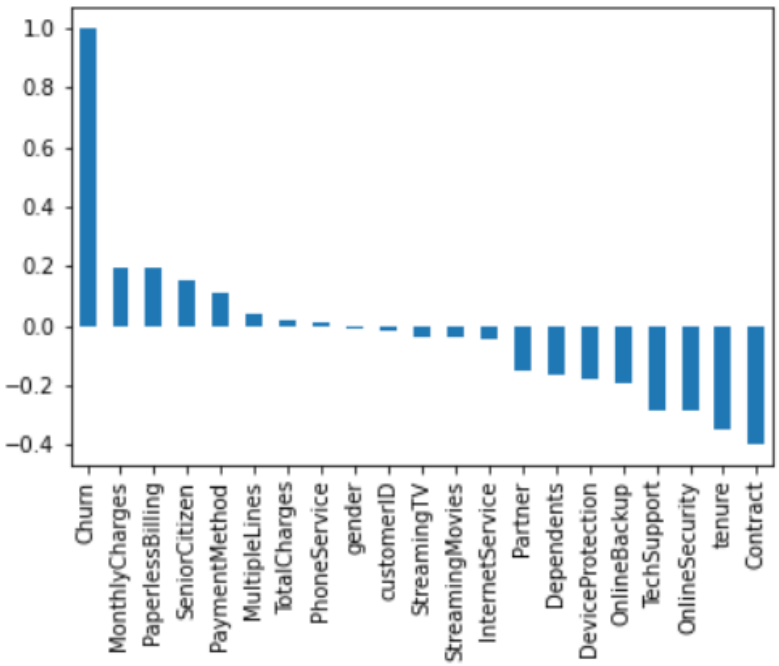
Correlation analysis:

```
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot = True)
plt.title('Correlations between factors',fontsize=20)
plt.savefig('correlation_between _factors.jpg')
plt.show()
```

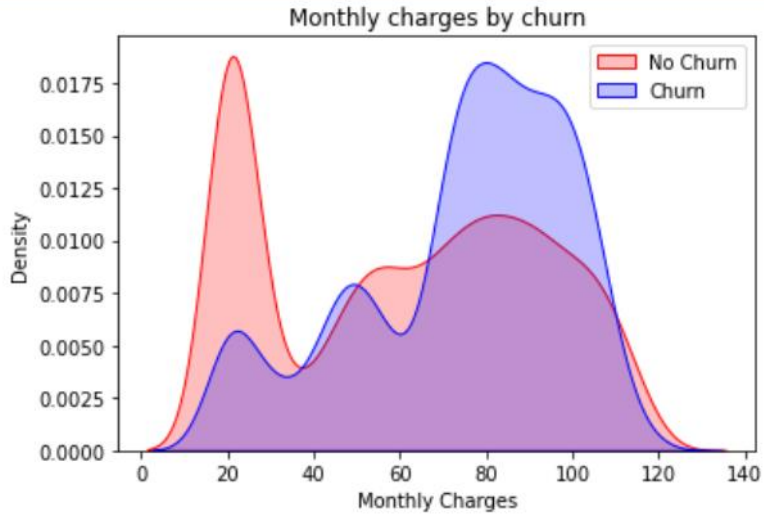
Output:




```
correlations = df.corr()['Churn'].sort_values(ascending=False)
correlations.plot(kind='bar')
```



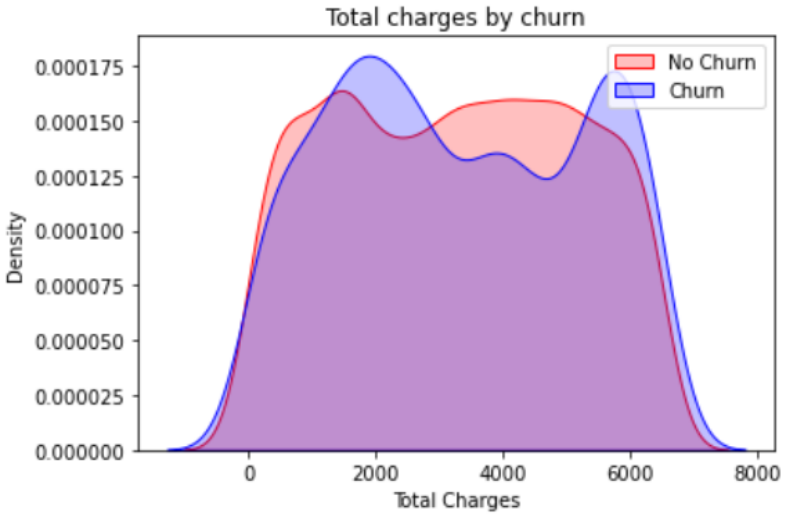

```
MC = sns.kdeplot(df.MonthlyCharges[(df["Churn"] == 0) ],
color="Red", shade = True)
MC = sns.kdeplot(df.MonthlyCharges[(df["Churn"] == 1) ],
ax =MC, color="Blue", shade= True)
MC.legend(["No Churn","Churn"],loc='upper right')
MC.set_ylabel('Density')
MC.set_xlabel('Monthly Charges')
MC.set_title('Monthly charges by churn')
```



Insight: Here it is evident that when the churn is high then the charges are high.

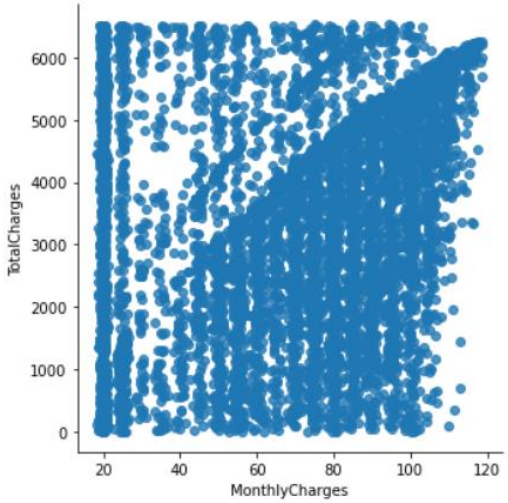
```
TC = sns.kdeplot(df.TotalCharges[(df["Churn"] == 0) ],
color="Red", shade = True)
TC= sns.kdeplot(df.TotalCharges[(df["Churn"] == 1) ],
```

```
ax =TC, color="Blue", shade= True)
TC.legend(["No Churn","Churn"],loc='upper right')
TC.set_ylabel('Density')
TC.set_xlabel('Total Charges')
TC.set_title('Total charges by churn')
```



Insights: Here we get the surprising insight that as we can see more churn is there with lower charges.Tenure, Monthly Charges & Total Charges then the picture is a bit clear:- Higher Monthly Charge at lower tenure results into lower Total Charge. Hence, all these 3 factors viz **Higher Monthly Charge**, **Lower tenure**, and **Lower Total Charge** are linked to **High Churn**.

```
sns.lmplot(data=telco_data_dummies, x='MonthlyCharges', y='TotalCharges', fit_reg=False)
```



it is clear by the plot that total charges and monthly charges are linearly correlated to each other.

Concluding Remarks

We went through the various tasks involved in Churn prediction in this article. It is important to note that finding patterns in Exploratory Data Analysis (EDA) is as important as the final prediction.

These are some of the quick insights on churn analysis from this exercise:

- Contract Type – Monthly customers are more likely to churn because of no contract terms, as they are free-to-go customers.
- Electronic check mediums are the highest churners
- No Online security, No Tech Support category are high churners
- Non-senior Citizens are high churners

Pre-Processing Pipeline

Data preparation and filtering steps can take considerable amount of processing time. Examples of data preprocessing include **cleaning, instance selection, normalization, one hot encoding, data transformation, feature extraction and selection**, etc. The product of data preprocessing is the final training set.
As the dataset have mostly categorical value except columns -Total charges,Monthly charges and Tenure.These 3 columns have continuous value. We will be encoding categorical columns.Encoding is a technique of **converting categorical variables into numerical values so that it could be easily fitted to a machine learning model**.
for this dataset we will be using label encoder and ordinal encoder to transform our categorical columns.
Standardization is a method to convert the data set into a normal distribution or remove skewness. There are many methods to remove skewness from the dataset.
1. Log-transformation — if the data set is highly rightly skewed then it is best to use Log-transformation
2. Square root transformation — if data is a little bit right-skewed then it is best to use square root transform
3. Cube root transformation
4. Reciprocal transformation
5. Box-cox transformation
6. Power transformation

And much more mathematical transformation is there to standardize the data.
Normalization is a mathematical technic that scales the data into some range it is required for some machine learning algorithms such as Linear regression, KNN, etc.
Normalization minimizes the difference between the low valve and high value which helps in better prediction. Some Normalization methods are mentioned below
1. Min-Max-Scaler- it converter the dataset into the range of 0 to 1
2. Mean Normalization- it converts the data set into the range of -1 ti 1 and the mean is 0.
3. Standard scaling(z-score)- its converts data set as mean=0, standard deviation as 1.

we will do scaling od the dataset using standard scalar:

```
#Data scaling Z=(X-mean)/std:
scaler=StandardScaler()
X_scaled=scaler.fit_transform(X)
X_scaled.shape[1]
```

Imbalanced data refers to those types of datasets where the target class has an uneven distribution of observations, i.e one class label has a very high number of observations and the other has a very low number of observations.
We will Dealing with the imbalanced dataset using Random over sampler:

```
round(y.value_counts(normalize=True) * 100, 2).astype('str') + ' %'
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=42)
ros.fit(X, y)
RandomOverSampler(random_state=42)
X_resampled, y_resampled = ros.fit_resample(X, y)
round(y_resampled.value_counts(normalize=True) * 100, 2).astype('str') + ' %'
```

Pre-Processing pipeline is a step-by-step process of pre-processing required for a dataset for making it more understandable and recognizable by the computer to perform machine learning.
The whole step-by-step process is packed in a set that can be created in function, class, or using pipeline from sklearn library.Preprocessing helps in reducing the noise or randomness from data. It consists of the following elements
1. **Data Cleaning:-** removing the Null value
2. **Data Integration:-** integrates data from a multitude of sources into a single data warehouse.
3. **Data Transformation:-** Transforming data such as standardization, normalization, encoding, etc
4. **Data Reduction:-** removing redundant data.

```
def pipeline_viz(df):
num,cat=num_cat(df)
num_plot(df,num)
df[num]=pow_tran(df,num)
df[num]=stan_sc(df,num)
```

Enable function step by step prepossessing functions are called to make a pipeline first function segregates numerical and categorical data, the second function plots the numerical data for the analysis, The third function standardizes the numerical data, the Forth function normalizes the numerical data.This function is the pipeline for preprocessing and visualization.

```
def num_cat(df):
num=[]
cat=[]
count=df.nunique()
for i in df.columns:
if count[i]>5:
num.append(i)
else:cat.append(i)
return(num,cat)
from sklearn.preprocessing import power_transform as PT
def pow_tran(df,num):
pt=pd.DataFrame()
for i in num:
if df[i].min()<=0:
pt1=(df[i]df[i].min()+0.0001)
else:pt1=df[i]
pt=pd.concat([pt,pd.DataFrame(pt1)],axis=1)
pt1=PT(pt)
pt1=pd.DataFrame(pt1,columns=num)
```

```
return(pt1)
def num_plot(df,num):
for i,predictor in enumerate(df.drop(columns=['Churn', 'TotalCharges',
'MonthlyCharges'])):
plt.figure(i)
(sns.countplot(data=df, x=predictor, hue='Churn')
def stan_sc(df,num):
ss=StandardScaler()
x=df[num].to_numpy()
x1=pd.DataFrame(ss.fit_transform(x),columns=num)
return(x1)
```

BUILDING MACHINE LEARNING MODELS

Algorithms of machine learning for classification.

1. Logistic regression — Logistic regression estimates the probability of an event occurring, such as voted or didn't vote, based on a given dataset of independent variables. Since the outcome is a probability, the dependent variable is bounded between 0 and 1.
2. Decision Tree classifier— The decision tree classifier creates the classification model by building a decision tree. Each node in the tree specifies a test on an attribute, each branch descending from that node corresponds to one of the possible values for that attribute.
3. Random forest classifier- A random forest classifier. A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.It is a multiple trees algorithm that uses bootstrap sampling and bagging for classification.
4. Extreme Gradient boosting classifier— Gradient boosting classifiers are a group of machine learning algorithms that combine many weak learning models together to create a strong predictive model. Decision trees are usually used when doing gradient boosting.It is a boosting algorithm that uses a multi-decision tree.
5. Support vector classifier- The objective of a Linear SVC (Support Vector Classifier) is to fit to the data you provide, returning a "best fit" hyperplane that divides, or categorizes, your data. From there, after getting the hyperplane, you can then feed some features to your classifier to see what the "predicted" class is.it constructs Hyperplane and marginal plane and distance between them is optimized

There are multiple machine learning algorithms to select from. Based on the quite best method to select the algorithm having the minimum difference between predicted score and cross-validation score.

Model Evaluation Metrics

For performance assessment of the chosen models, various metrics are used:

- **Feature weights:** Indicates the top features used by the model to generate the predictions

- **Confusion matrix:** Shows a grid of true and false predictions compared to the actual values
- **Accuracy score:** Shows the overall accuracy of the model for training set and test set
- **ROC Curve:** Shows the diagnostic ability of a model by bringing together true positive rate (TPR) and false positive rate (FPR) for different thresholds of class predictions (e.g. thresholds of 10%, 50% or 90% resulting to a prediction of churn)
- **AUC (for ROC):** Measures the overall separability between classes of the model related to the ROC curve
- **Precision-Recall-Curve:** Shows the diagnostic ability by comparing false positive rate (FPR) and false negative rate (FNR) for different thresholds of class predictions. It is suitable for data sets with high class imbalances (negative values overrepresented) as it focuses on precision and recall, which are not dependent on the number of true negatives and thereby excludes the imbalance
- **F1 Score:** Builds the harmonic mean of precision and recall and thereby measures the compromise between both.
- **AUC (for PRC):** Measures the overall separability between classes of the model related to the Precision-Recall curve

Pipeline:

```
def clf(model,x,y,x_test,y_test):
lr = model.fit(x, y)
y_pred=lr.predict(x_test)
print(confusion_matrix(y_test,y_pred))
print(accuracy_score(y_test,y_pred))
print(classification_report(y_test,y_pred))
lr_acc=f1_score(y_test,y_pred)
cv_results = cross_validate(model, x, y, cv=5)
lr_score=cv_results['test_score'].mean()
return(lr_acc,lr_score)
lr_acc,lr_score=clf(LogisticRegression(),x_train_scaler,y_train,x_test_scal
er,y_test)
rfc_acc,rfc_score=clf(RFR(),x_train_scaler,y_train,x_test_scaler,y_test)
xgc_acc,xgc_score=clf(XBR(),x_train_scaler,y_train,x_test_scaler,y_test)
etc_acc,etc_score=clf(ETR(),x_train_scaler,y_train,x_test_scaler,y_test)
svc_acc,svc_score=clf(SVC(),x_train_scaler,y_train,x_test_scaler,y_test)
score=[lr_score,rfc_score,xgc_score,etc_score,svc_score]
error=[lr_acc,rfc_acc,xgc_acc,etc_acc,svc_acc]
name=['LR','RFC','XGC','ETC','SVC']
diff=[]
for i in range(5):
diff.append(score[i]-error[i])
pd.DataFrame([name,score,error,diff]).T
```

Now after selecting the best algorithm now we should do Hyperparameter tuning for logistic regression.

1. Random search cv:- It randomly Search the grid for the best scoring. Processing time is less but the accuracy is not that great
2. Grid search cv:- it searches the grid one by one for the best scoring. Processing time is more but the accuracy is great

```
from sklearn.model_selection import GridSearchCV
ridge_params = {'alpha':[1,2,3,4,5,6,7,8,9,10]}
lr_grid = GridSearchCV(Ridge(), ridge_params, cv=5)
lr_grid.fit(x_train_scaler, y_train)
```

```
print('Best score:', lr_grid.best_score_)
print('Best score:', lr_grid.best_params_)
print('Best score:', lr_grid.best_estimator_)
```

hypertuned model (logistic regression)

```
reg=LogisticRegression(C=1.0,dual=False,fit_intercept= True, intercept_scaling=1, max_iter=100, tol= 0.0001, verbose= 0, warm_sta
reg.fit(X_train,y_train)
reg.score(X_train,y_train)
y_pred=reg.predict(X_test)
regs=accuracy_score(y_test,y_pred)
print('accuracy score:',regs*100)

regscore=cross_val_score(reg,X,y,cv=5)
regc=regscore.mean()
print('cross val score:',regc*100)
```

```
accuracy score: 81.90205819730305
cross val score: 79.95184165107425
```

Evaluation metrics for logistic regression model:

```
#confusion matrix
conf_mat=confusion_matrix(y_test,y_pred)
conf_mat
```

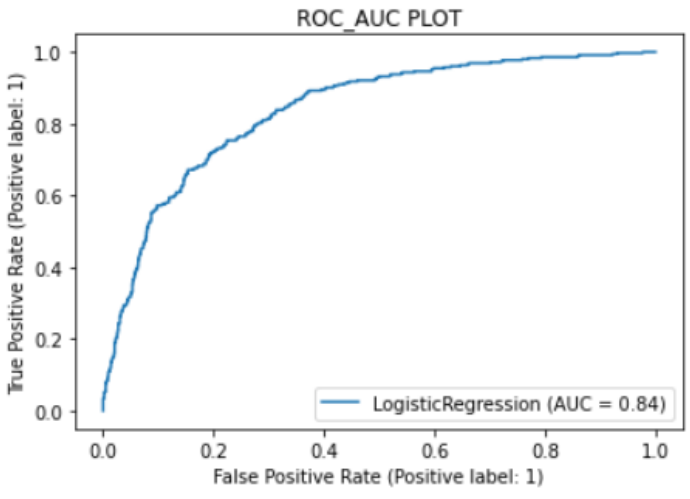
```
array([[953, 103],
       [152, 201]], dtype=int64)
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.90      0.88      1056
           1       0.66      0.57      0.61       353

    accuracy                           0.82      1409
   macro avg       0.76      0.74      0.75      1409
weighted avg       0.81      0.82      0.81      1409
```

The roc auc plot of the model :

```
from sklearn.metrics import plot_roc_curve
plot_roc_curve(clf.best_estimator_,X_test,y_test)
plt.title("ROC_AUC PLOT")
plt.show()
```



Concluding Remarks:

- Senior citizens churn rate is much higher than non-senior churn rate.
- Churn rate for month-to-month contracts much higher that for other contract durations.
- Moderately higher churn rate for customers without partners.
- Much higher churn rate for customers without children.
- Payment method electronic check shows much higher churn rate than other payment methods.
- Customers with InternetService fiber optic as part of their contract have much higher churn rate.
- **we conclude this Logistic regression model as best model with auc value of 84%,accuracy score as 81% and f1 score as 88% for no churn**