```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

df= pd.read_csv('/content/telecom_churn_mock_data.csv')
df.head(10)
```

| | CustomerID | Gender | SeniorCitizen | Partner | Dependents | Tenure | PhoneService | Multipl |
|---|---|---|---|---|---|---|---|---|
| **0** | CUST1000 | Male | 0 | No | No | 30 | Yes | |
| **1** | CUST1001 | Female | 0 | No | Yes | 11 | Yes | |
| **2** | CUST1002 | Female | 1 | No | No | 17 | No | |
| **3** | CUST1003 | Female | 0 | Yes | No | 26 | Yes | |
| **4** | CUST1004 | Male | 0 | Yes | Yes | 23 | Yes | |
| **5** | CUST1005 | Male | 0 | Yes | No | 56 | Yes | |
| **6** | CUST1006 | Male | 0 | Yes | No | 52 | Yes | |
| **7** | CUST1007 | Female | 0 | Yes | Yes | 48 | Yes | |
| **8** | CUST1008 | Female | 0 | No | No | 20 | No | |
| **9** | CUST1009 | Female | 0 | Yes | Yes | 2 | Yes | |

10 rows × 21 columns

## EDA

```
df.shape
```

```
(2000, 21)
```

Start coding or generate with AI.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   CustomerID        2000 non-null   object 
 1   Gender            2000 non-null   object 
 2   SeniorCitizen     2000 non-null   int64  
 3   Partner           2000 non-null   object 
 4   Dependents        2000 non-null   object 
 5   Tenure            2000 non-null   int64  
 6   PhoneService      2000 non-null   object 
 7   MultipleLines     2000 non-null   object 
 8   InternetService   2000 non-null   object 
 9   OnlineSecurity    2000 non-null   object 
 10  OnlineBackup      2000 non-null   object 
 11  DeviceProtection  2000 non-null   object 
 12  TechSupport       2000 non-null   object 
 13  StreamingTV       2000 non-null   object 
 14  StreamingMovies   2000 non-null   object 
 15  Contract          2000 non-null   object 
 16  PaperlessBilling  2000 non-null   object 
 17  PaymentMethod     2000 non-null   object 
 18  MonthlyCharges    2000 non-null   float64
 19  TotalCharges      1980 non-null   float64
 20  Churn             2000 non-null   object 
dtypes: float64(2), int64(2), object(17)
memory usage: 328.3+ KB
```

```
correlation = df.corr(numeric_only=True)
print(correlation)

# in this table we got that tensure and total charges have reltaion AS we can see the tensur
# monthly charges and total charges have relation.
```

```
                SeniorCitizen    Tenure  MonthlyCharges  TotalCharges
SeniorCitizen        1.000000 -0.001378       -0.035339     -0.021054
Tenure              -0.001378  1.000000        0.000447      0.763255
MonthlyCharges      -0.035339  0.000447        1.000000      0.565613
TotalCharges        -0.021054  0.763255        0.565613      1.000000
```

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

| Churn | count |
|---|---|
| No | 1354 |
| Yes | 646 |

dtype: int64

```
df['Partner'].value_counts()
```

| Partner | count |
|---|---|
| No | 1024 |
| Yes | 976 |

dtype: int64

```
df['SeniorCitizen'].value_counts()
```

| SeniorCitizen | count |
|---|---|
| 0 | 1697 |
| 1 | 303 |

dtype: int64

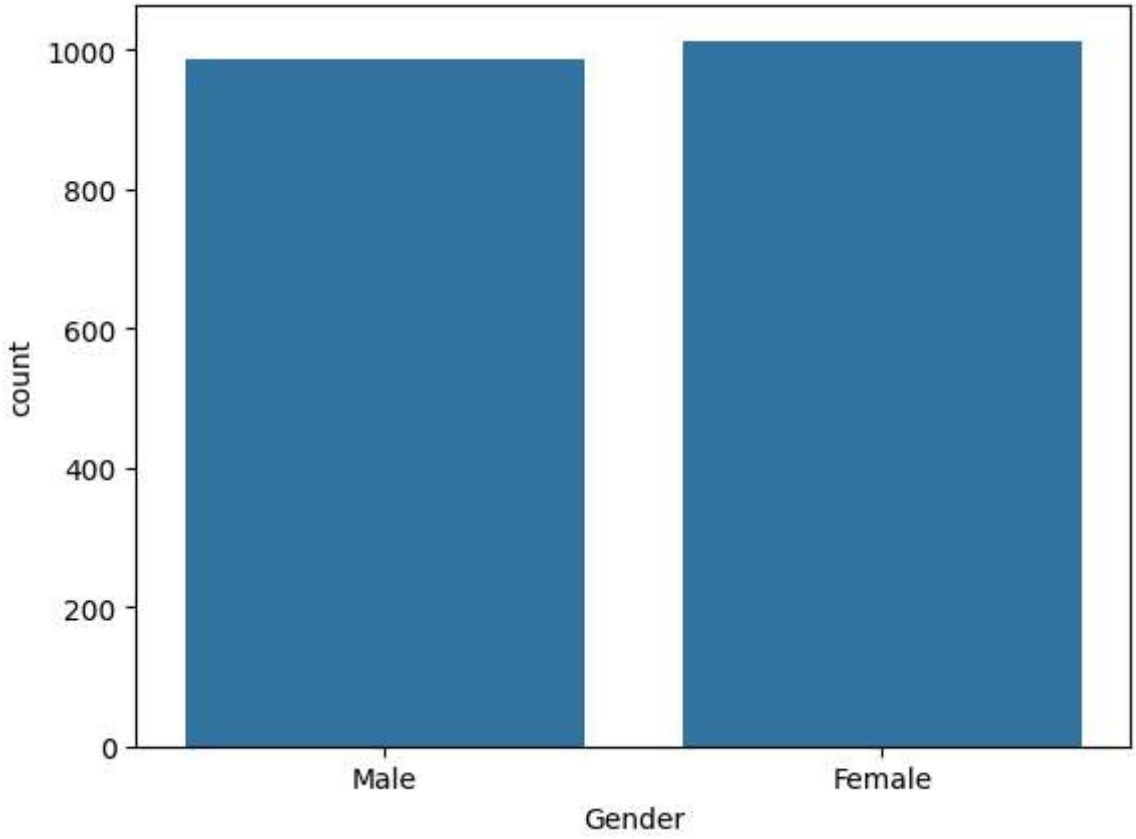
```
df['Gender'].value_counts()
```

| Gender | count |
|---|---|
| Female | 1013 |
| Male | 987 |

dtype: int64

```
# we got gender coulmn so we have to check how many peoples in that group
sns.countplot(x='Gender',data=df)
```

```
<Axes: xlabel='Gender', ylabel='count'>
```




```
# individual coulmns of missing values in percentage.
df.isnull().sum()/df.shape[0]*100
```

| | 0 |
|---|---|
| **CustomerID** | 0.0 |
| **Gender** | 0.0 |
| **SeniorCitizen** | 0.0 |
| **Partner** | 0.0 |
| **Dependents** | 0.0 |
| **Tenure** | 0.0 |
| **PhoneService** | 0.0 |
| **MultipleLines** | 0.0 |
| **InternetService** | 0.0 |
| **OnlineSecurity** | 0.0 |
| **OnlineBackup** | 0.0 |
| **DeviceProtection** | 0.0 |
| **TechSupport** | 0.0 |
| **StreamingTV** | 0.0 |
| **StreamingMovies** | 0.0 |
| **Contract** | 0.0 |
| **PaperlessBilling** | 0.0 |
| **PaymentMethod** | 0.0 |
| **MonthlyCharges** | 0.0 |
| **TotalCharges** | 1.0 |
| **Churn** | 0.0 |

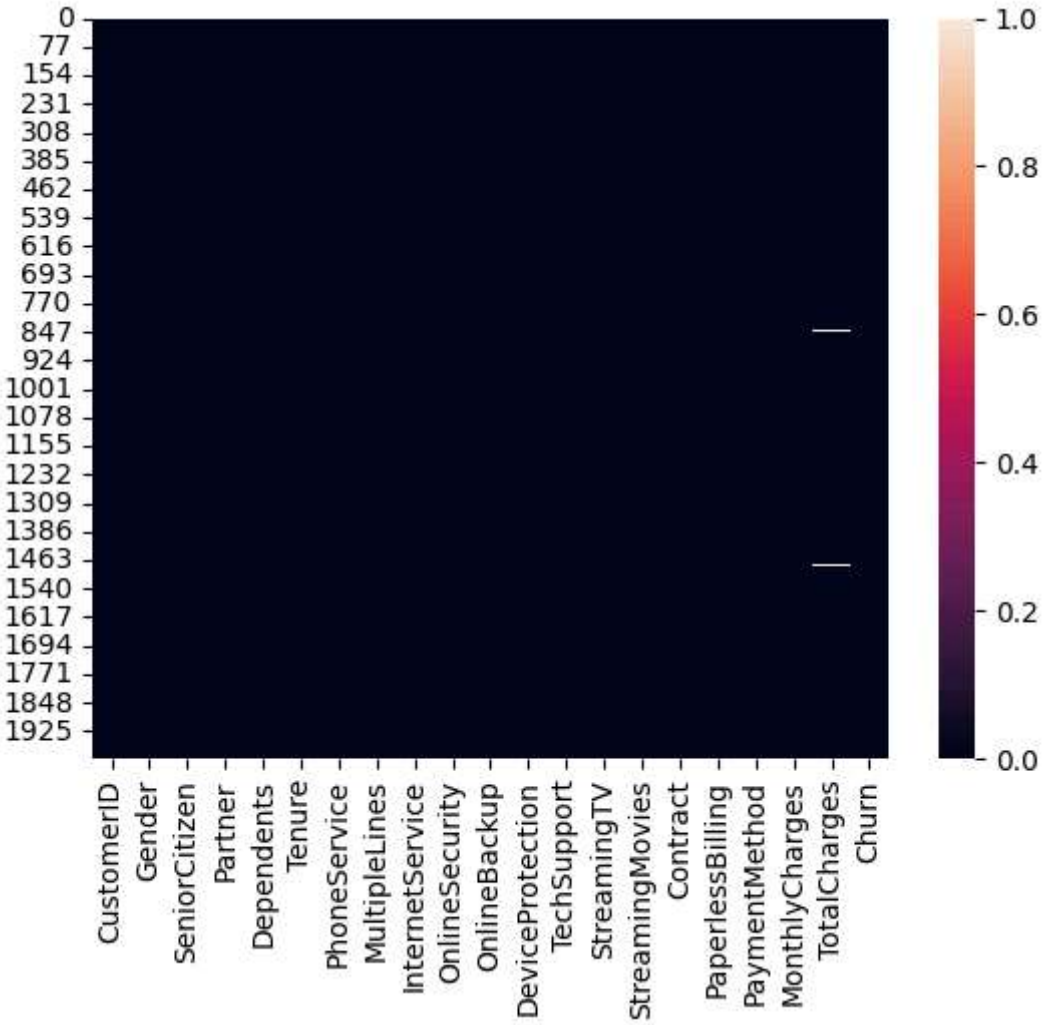
**dtype:** float64

```
# Total percentage of missing values of data type
df.isnull().sum().sum()/df.shape[0]*100
```

```
np.float64(1.0)
```

```
# Graphical represantaion of null values
sns.heatmap(df.isnull())
```

```
<Axes: >
```




```
# removig null values from data set by using dropna
df.dropna(subset=['TotalCharges'], inplace=True)
```

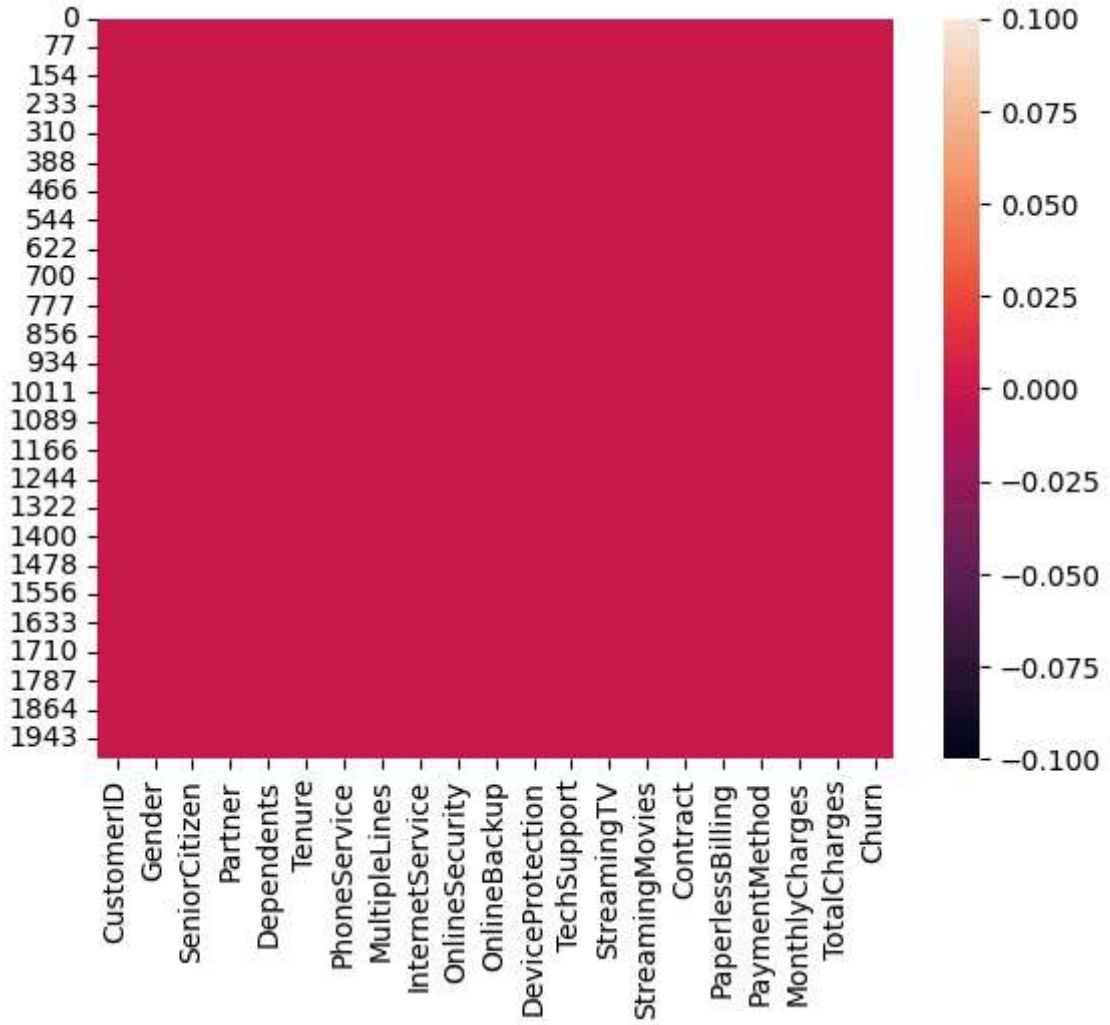
```
# checking the null values again
df.isnull().sum()
```

| | 0 |
|---|---|
| **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

```
sns.heatmap(df.isnull())
```

```
<Axes: >
```




```
df.select_dtypes(include='number').columns
```

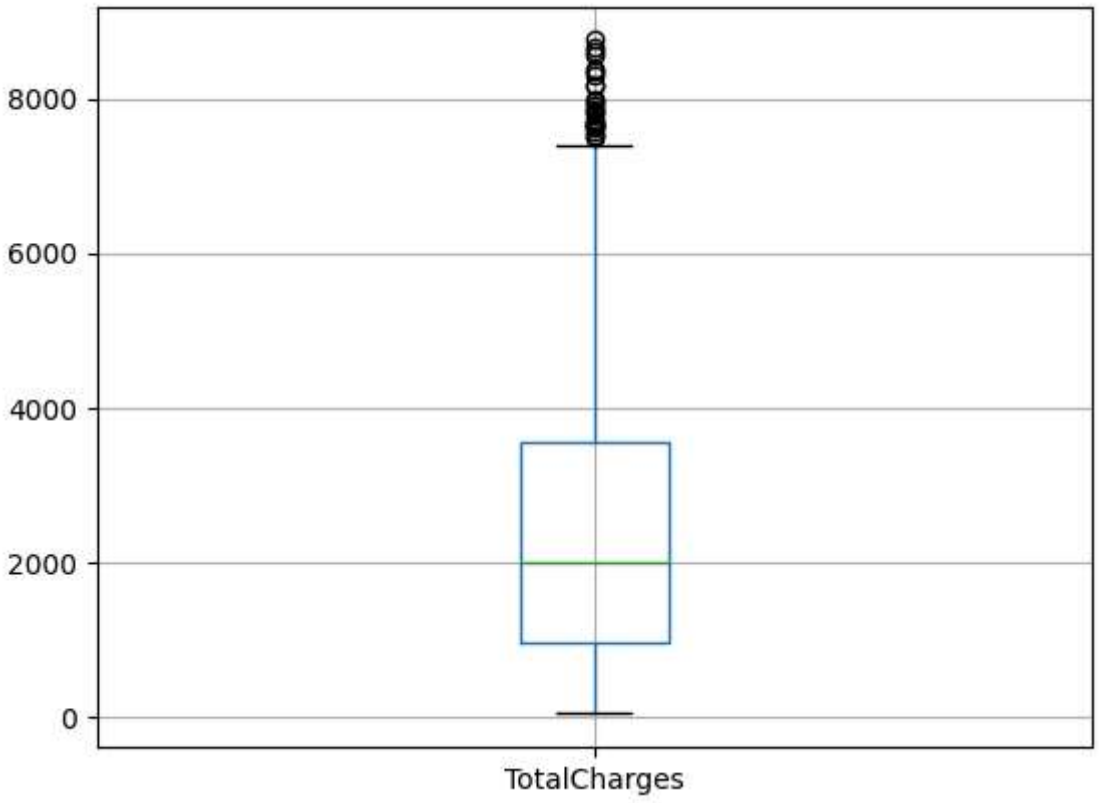
```
Index(['SeniorCitizen', 'Tenure', 'MonthlyCharges', 'TotalCharges'], dtype='object')
```

```
df.describe()
```

| | SeniorCitizen | Tenure | MonthlyCharges | TotalCharges |
|---|---|---|---|---|
| count | 1980.000000 | 1980.000000 | 1980.000000 | 1980.000000 |
| mean | 0.151515 | 36.694949 | 65.846263 | 2417.307414 |
| std | 0.358641 | 20.839236 | 27.166266 | 1802.118466 |
| min | 0.000000 | 1.000000 | 18.000000 | 36.020000 |
| 25% | 0.000000 | 18.000000 | 45.225000 | 954.862500 |
| 50% | 0.000000 | 37.000000 | 65.460000 | 1993.475000 |
| 75% | 0.000000 | 54.000000 | 84.967500 | 3561.940000 |
| max | 1.000000 | 72.000000 | 120.000000 | 8756.020000 |

```
# Detecting outliers
df.boxplot(column=['TotalCharges'])
```

```
<Axes: >
```




```
# checking outliers in total charges
Q1 = df['TotalCharges'].quantile(0.25)
Q3 = df['TotalCharges'].quantile(0.75)
IQR = Q3 - Q1

lower_limit = Q1 - 1.5 * IQR
upper_limit = Q3 + 1.5 * IQR
```

```
print("Lower Limit:", lower_limit)
print("Upper Limit:", upper_limit)

# See how many values are outliers
outliers = df[(df['TotalCharges'] < lower_limit) | (df['TotalCharges'] > upper_limit)]
print("Outliers found:", outliers.shape[0])
```

```
Lower Limit: -2955.7537500000008
Upper Limit: 7472.556250000001
Outliers found: 24
```

```
# Ecoding catagorical variables
le = LabelEncoder()
binary_cols = ['Gender', 'Partner', 'Dependents', 'Churn','PaperlessBilling','StreamingTV','


for col in binary_cols:
    df[col] = le.fit_transform(df[col])


df = pd.get_dummies(
    df,
    columns=[
        'Contract',
        'PaymentMethod',
        'InternetService',
        'OnlineSecurity',
        'OnlineBackup',
        'TechSupport',
        'StreamingMovies'
    ],
    drop_first=True
)



df.head()
```

| | CustomerID | Gender | SeniorCitizen | Partner | Dependents | Tenure | PhoneService | Multipl |
|---|---|---|---|---|---|---|---|---|
| **0** | CUST1000 | 1 | 0 | 0 | 0 | 30 | 1 | |
| **1** | CUST1001 | 0 | 0 | 0 | 1 | 11 | 1 | |
| **2** | CUST1002 | 0 | 1 | 0 | 0 | 17 | 0 | |
| **3** | CUST1003 | 0 | 0 | 1 | 0 | 26 | 1 | |
| **4** | CUST1004 | 1 | 0 | 1 | 1 | 23 | 1 | |

5 rows × 29 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1980 entries, 0 to 1999
Data columns (total 29 columns):
 #   Column                                 Non-Null Count  Dtype
---  ------                                 --------------  -----
 0   CustomerID                             1980 non-null   object
 1   Gender                                 1980 non-null   int64
 2   SeniorCitizen                          1980 non-null   int64
 3   Partner                                1980 non-null   int64
 4   Dependents                             1980 non-null   int64
 5   Tenure                                 1980 non-null   int64
 6   PhoneService                           1980 non-null   int64
 7   MultipleLines                          1980 non-null   int64
 8   DeviceProtection                       1980 non-null   int64
 9   StreamingTV                            1980 non-null   int64
 10  PaperlessBilling                       1980 non-null   int64
 11  MonthlyCharges                         1980 non-null   float64
 12  TotalCharges                           1980 non-null   float64
 13  Churn                                  1980 non-null   int64
 14  Contract_One year                      1980 non-null   bool
 15  Contract_Two year                      1980 non-null   bool
 16  PaymentMethod_Credit card (automatic)  1980 non-null   bool
 17  PaymentMethod_Electronic check         1980 non-null   bool
 18  PaymentMethod_Mailed check             1980 non-null   bool
 19  InternetService_Fiber optic            1980 non-null   bool
 20  InternetService_No                     1980 non-null   bool
 21  OnlineSecurity_No internet service     1980 non-null   bool
 22  OnlineSecurity_Yes                     1980 non-null   bool
 23  OnlineBackup_No internet service       1980 non-null   bool
 24  OnlineBackup_Yes                       1980 non-null   bool
 25  TechSupport_No internet service        1980 non-null   bool
 26  TechSupport_Yes                        1980 non-null   bool
 27  StreamingMovies_No internet service    1980 non-null   bool
 28  StreamingMovies_Yes                    1980 non-null   bool
dtypes: bool(15), float64(2), int64(11), object(1)
memory usage: 261.0+ KB
```

```
# checking the boolean coulmns
bool_cols = df.select_dtypes(include='bool').columns
print(bool_cols)
```

```
Index(['Contract_One year', 'Contract_Two year',
       'PaymentMethod_Credit card (automatic)',
       'PaymentMethod_Electronic check', 'PaymentMethod_Mailed check',
       'InternetService_Fiber optic', 'InternetService_No',
       'OnlineSecurity_No internet service', 'OnlineSecurity_Yes',
       'OnlineBackup_No internet service', 'OnlineBackup_Yes',
       'TechSupport_No internet service', 'TechSupport_Yes',
       'StreamingMovies_No internet service', 'StreamingMovies_Yes'],
      dtype='object')
```

```
#convert boolean into numeric
df[bool_cols] = df[bool_cols].astype(int)
```

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

| | CustomerID | Gender | SeniorCitizen | Partner | Dependents | Tenure | PhoneService | Multipl |
|---|---|---|---|---|---|---|---|---|
| **0** | CUST1000 | 1 | 0 | 0 | 0 | 30 | 1 | |
| **1** | CUST1001 | 0 | 0 | 0 | 1 | 11 | 1 | |
| **2** | CUST1002 | 0 | 1 | 0 | 0 | 17 | 0 | |
| **3** | CUST1003 | 0 | 0 | 1 | 0 | 26 | 1 | |
| **4** | CUST1004 | 1 | 0 | 1 | 1 | 23 | 1 | |

5 rows × 29 columns

## Feature Engineering

```
# Total spend = monthly charges * Tensure
df['TotalSpend'] = df['MonthlyCharges'] * df['Tenure']
```

```
#Tensure in years
df['TenureYears'] = df['Tenure'] / 12


# average monthly charges over tensure
df['AvgMonthlyCharges'] = df['TotalCharges'] / df['Tenure']
```

## Model Selection And Training

```
# dropping customer ID because we do need it in prediction
# which will throwing error in the model because of it unnumeric state.
df = df.drop('CustomerID', axis=1)



# Split the features in to (X) and (y)
# we going to put our features in to (x)
# we gping to puyt the target in to (y)
x=df.drop('Churn',axis=1) # input
y=df['Churn'] # target


# we Spilit the data into Train Test model and we training our model by 80 percent of data
# remaining the data for the test the model
#x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)


#from operator import mod
# choosing the logistic regression model for the absic and fast result
#model=LogisticRegression(max_iter=1000)


#from sklearn.ensemble import GradientBoostingClassifier

#odel = GradientBoostingClassifier(random_state=42)



#model.fit(x_train, y_train)
```

```
GradientBoostingClassifier
GradientBoostingClassifier(random_state=42)
```

```
# make a pridiction
#y_pred = model.predict(x_test)
```

```
#from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

#print("Accuracy:", accuracy_score(y_test, y_pred))
#print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
#print("Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.6565656565656566
Confusion Matrix:
 [[242  31]
 [105  18]]
Report:
              precision    recall  f1-score   support

           0       0.70      0.89      0.78       273
           1       0.37      0.15      0.21       123

    accuracy                           0.66       396
   macro avg       0.53      0.52      0.49       396
weighted avg       0.59      0.66      0.60       396
```

```
#from sklearn.ensemble import RandomForestClassifier

#model = RandomForestClassifier(random_state=42)
#model.fit(x_train, y_train)
```

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

```
#from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

#print("Accuracy:", accuracy_score(y_test, y_pred))
#print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
#print("Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.6540404040404041
Confusion Matrix:
 [[248  25]
 [112  11]]
Report:
```