Customer Churn Prediction

Customer churn is defined as users who have left within the last month.

Dataset Used - Kaggle Telco Customer Churn
[https://www.kaggle.com/datasets/blastchar/telco-customer-churn]

```
# Importing dependencies

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder # To handle categorical
values
from imblearn.over_sampling import SMOTE # To handle class imbalance
from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
import pickle

# loading the dataset
customer_data = pd.read_csv("telco.csv")

customer_data.shape
```

```
(7043, 21)
```

```
pd.set_option('display.max_columns', None) # To ensure all the
features are displayed.
customer_data.head()
```

{"type":"dataframe","variable_name":"customer_data"}

- We dont need the customerID column, can be removed.
- Tenure is in months.

```
customer_data.info()

# There are 19 independent features used to predict the target - Churn
```

```
<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
```

We observe that the TotalCharges is an Object, which should be converted to a float value.

```
# Dropping the customer ID column since it is not required.
df = customer_data.drop(columns = ['customerID'], axis = 1)

# Printing the unique values in all the catergorical columns
numerical_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']
for col in df.columns:
    if col not in numerical_cols:
        print(col,":", df[col].unique())
        print('----'*50)
```

```
gender : ['Female' 'Male']
----------------------------------------------------------------------
----------------------------------------------------------------------
------------------------------------------------------------
SeniorCitizen : [0 1]
----------------------------------------------------------------------
----------------------------------------------------------------------
------------------------------------------------------------
Partner : ['Yes' 'No']
----------------------------------------------------------------------
```

```
----------------------------------------------------------------------
------------------------------------------------------------
Dependents : ['No' 'Yes']
----------------------------------------------------------------------
----------------------------------------------------------------------
------------------------------------------------------------
PhoneService : ['No' 'Yes']
----------------------------------------------------------------------
----------------------------------------------------------------------
------------------------------------------------------------
MultipleLines : ['No phone service' 'No' 'Yes']
----------------------------------------------------------------------
----------------------------------------------------------------------
------------------------------------------------------------
InternetService : ['DSL' 'Fiber optic' 'No']
----------------------------------------------------------------------
----------------------------------------------------------------------
------------------------------------------------------------
OnlineSecurity : ['No' 'Yes' 'No internet service']
----------------------------------------------------------------------
----------------------------------------------------------------------
------------------------------------------------------------
OnlineBackup : ['Yes' 'No' 'No internet service']
----------------------------------------------------------------------
----------------------------------------------------------------------
------------------------------------------------------------
DeviceProtection : ['No' 'Yes' 'No internet service']
----------------------------------------------------------------------
----------------------------------------------------------------------
------------------------------------------------------------
TechSupport : ['No' 'Yes' 'No internet service']
----------------------------------------------------------------------
----------------------------------------------------------------------
------------------------------------------------------------
StreamingTV : ['No' 'Yes' 'No internet service']
----------------------------------------------------------------------
----------------------------------------------------------------------
------------------------------------------------------------
StreamingMovies : ['No' 'Yes' 'No internet service']
----------------------------------------------------------------------
----------------------------------------------------------------------
------------------------------------------------------------
Contract : ['Month-to-month' 'One year' 'Two year']
----------------------------------------------------------------------
----------------------------------------------------------------------
------------------------------------------------------------
PaperlessBilling : ['Yes' 'No']
----------------------------------------------------------------------
----------------------------------------------------------------------
```

```
------------------------------------------------------------
PaymentMethod : ['Electronic check' 'Mailed check' 'Bank transfer
(automatic)'
 'Credit card (automatic)']
----------------------------------------------------------------------
----------------------------------------------------------------------
------------------------------------------------------------
Churn : ['No' 'Yes']
----------------------------------------------------------------------
----------------------------------------------------------------------
------------------------------------------------------------
```

```
# Checking for null values in the dataset

print(customer_data.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["TotalCharges"] = df["TotalCharges"].astype(float)
```

```
----------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
<ipython-input-9-f2be9b02420a> in <cell line: 0>()
----> 1 df["TotalCharges"] = df["TotalCharges"].astype(float)

/usr/local/lib/python3.11/dist-packages/pandas/core/generic.py in
astype(self, dtype, copy, errors)
```

```
   6641         else:
   6642             # else, only a single dtype is given
-> 6643             new_data = self._mgr.astype(dtype=dtype,
copy=copy, errors=errors)
   6644             res = self._constructor_from_mgr(new_data,
axes=new_data.axes)
   6645             return res.__finalize__(self, method="astype")

/usr/local/lib/python3.11/dist-packages/pandas/core/internals/managers
.py in astype(self, dtype, copy, errors)
    428             copy = False
    429 
--> 430         return self.apply(
    431             "astype",
    432             dtype=dtype,

/usr/local/lib/python3.11/dist-packages/pandas/core/internals/managers
.py in apply(self, f, align_keys, **kwargs)
    361                 applied = b.apply(f, **kwargs)
    362             else:
--> 363                 applied = getattr(b, f)(**kwargs)
    364             result_blocks = extend_blocks(applied,
result_blocks)
    365 

/usr/local/lib/python3.11/dist-packages/pandas/core/internals/blocks.p
y in astype(self, dtype, copy, errors, using_cow, squeeze)
    756             values = values[0, :]  # type: ignore[call-
overload]
    757 
--> 758         new_values = astype_array_safe(values, dtype,
copy=copy, errors=errors)
    759 
    760         new_values = maybe_coerce_values(new_values)

/usr/local/lib/python3.11/dist-packages/pandas/core/dtypes/astype.py
in astype_array_safe(values, dtype, copy, errors)
    235 
    236     try:
--> 237         new_values = astype_array(values, dtype, copy=copy)
    238     except (ValueError, TypeError):
    239         # e.g. _astype_nansafe can fail on object-dtype of
strings

/usr/local/lib/python3.11/dist-packages/pandas/core/dtypes/astype.py
in astype_array(values, dtype, copy)
    180 
    181     else:
--> 182         values = _astype_nansafe(values, dtype, copy=copy)
    183 
```

```
    184     # in pandas we don't store numpy str dtypes, so convert to
object

/usr/local/lib/python3.11/dist-packages/pandas/core/dtypes/astype.py
in _astype_nansafe(arr, dtype, copy, skipna)
    131     if copy or arr.dtype == object or dtype == object:
    132         # Explicit copy, or required since NumPy can't view
from / to object.
--> 133         return arr.astype(dtype, copy=True)
    134
    135     return arr.astype(dtype, copy=copy)

ValueError: could not convert string to float: ' '
```

```
# Replacing the missing values by 0

df["TotalCharges"] = df["TotalCharges"].replace(" ", "0.0") # They
have not used the service for more than one month, hence charge is 0.0
df["TotalCharges"] = df["TotalCharges"].astype(float)

customer_data.duplicated().sum()
```

```
0
```

```
df.info()
```

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

```
dtypes: float64(2), int64(2), object(16)
memory usage: 1.1+ MB

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

Churn
No     5174
Yes    1869
Name: count, dtype: int64
```

- We can see that there is a imbalance in the samples. We find that there are more samples for the class 'no' than the 'yes' class. Only around 27% of the customers in the dataset have churned. We are dealing with an imbalanced classification problem through upsampling/ downsampling.

Exploratory Data Analysis

```
df.describe()
```

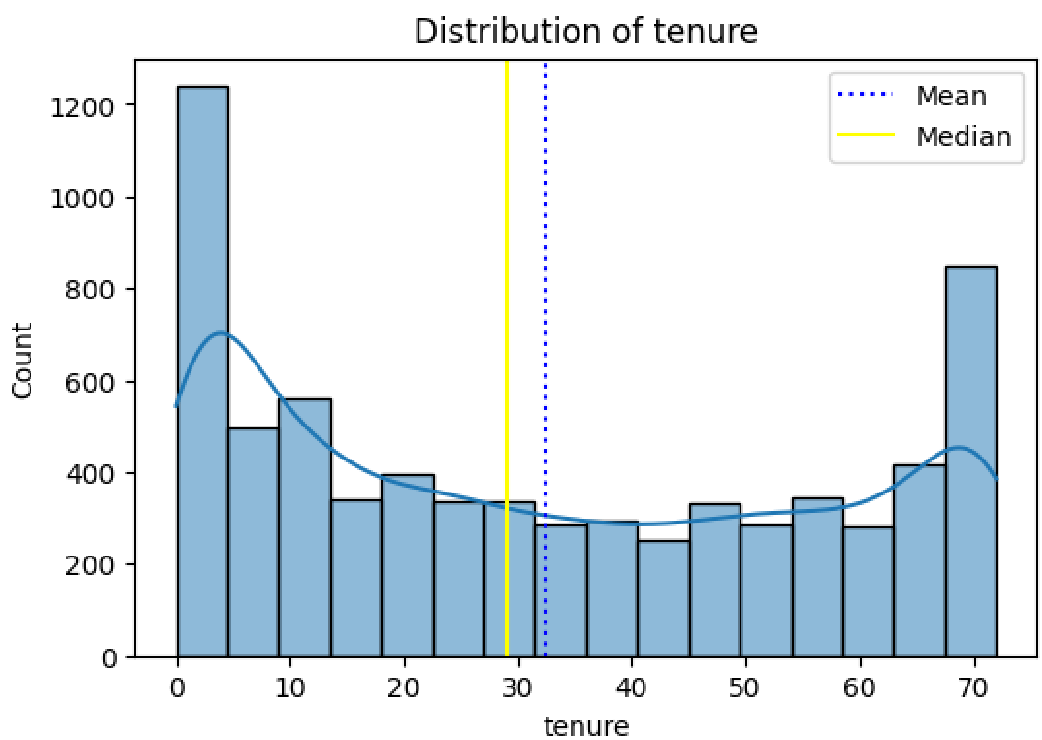

```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 8,\n  \"fields\": [\n    {\n      \"column\": \"SeniorCitizen\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 2489.9992387084,\n        \"min\": 0.0,\n        \"max\": 7043.0,\n        \"num_unique_values\": 5,\n        \"samples\": [\n          0.1621468124378816,\n          1.0,\n          0.3686116056100268 7\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"tenure\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 2478.9752758409018,\n        \"min\": 0.0,\n        \"max\": 7043.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n          32.37114865824223,\n          29.0,\n          7043.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"MonthlyCharges\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 2468.7047672837775,\n        \"min\": 18.25,\n        \"max\": 7043.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n          64.76169246059918,\n          70.35,\n          7043.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"TotalCharges\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 3122.5732655623974,\n        \"min\": 0.0,\n        \"max\": 8684.8,\n        \"num_unique_values\": 8,\n        \"samples\": [\n          2279.7343035638223,\n          1394.55,\n          7043.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

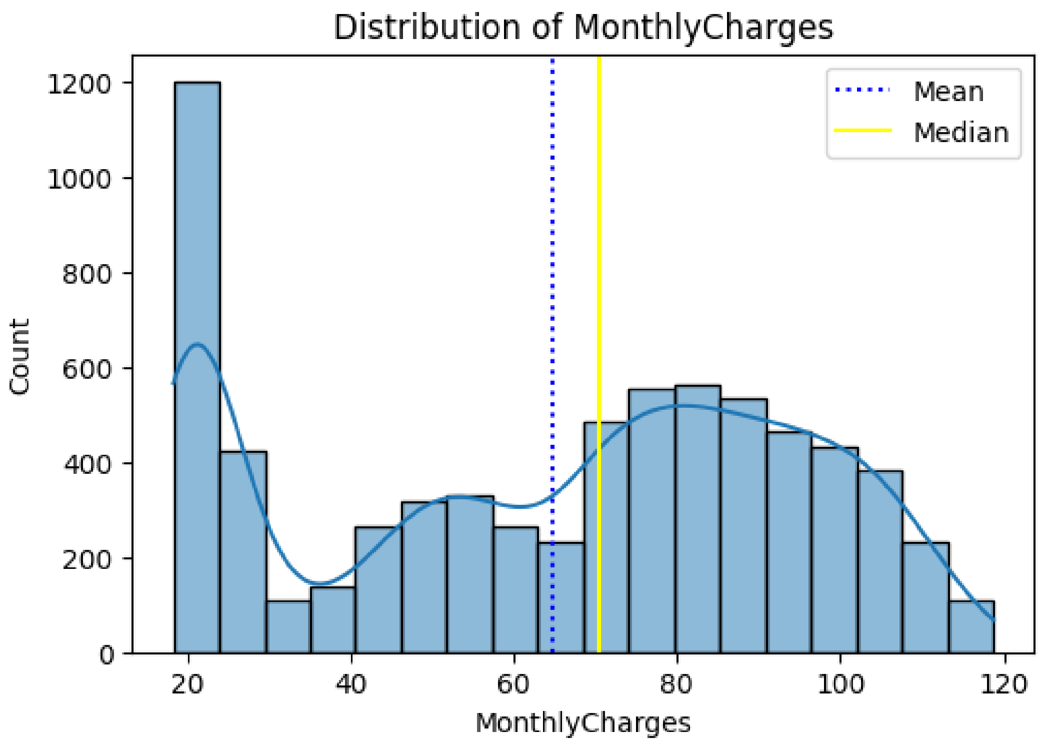

Observation:

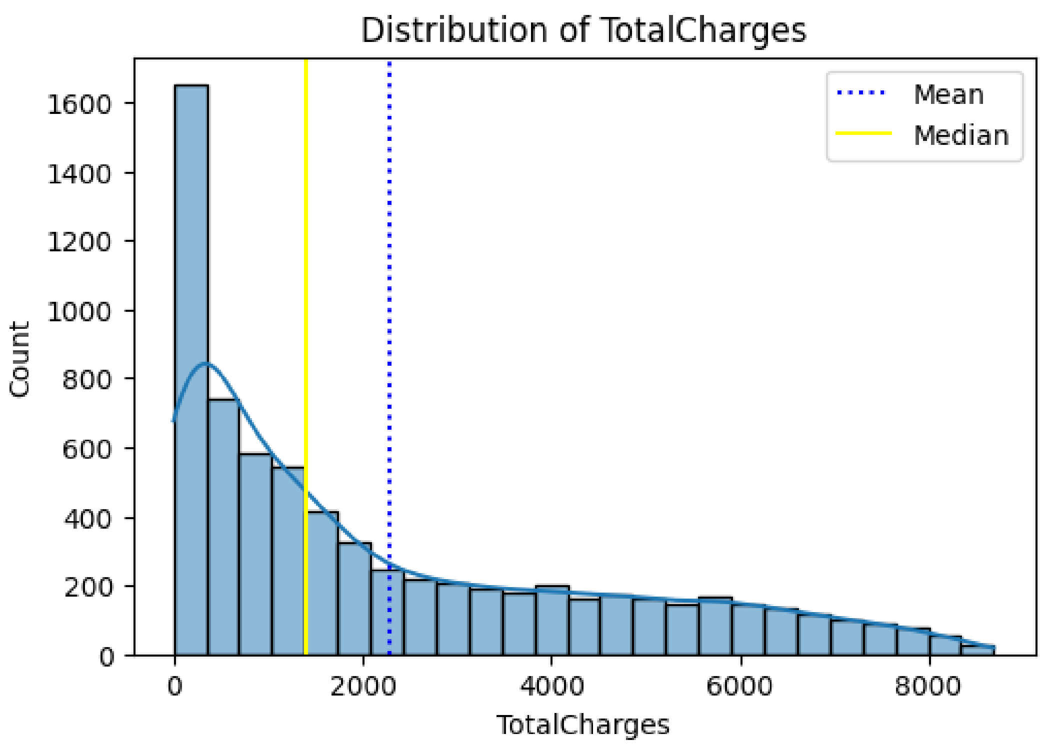
- SeniorCitizen - We can observe that the mean is 0.16, which means we dont have many samples where the customer is a senior citizen.

Analysis of the continuous numerical features

```
def plt_hist(dataset, column_name):
    plt.figure(figsize = (6, 4))
    sns.histplot( dataset[column_name], kde = True)
    plt.title(f"Distribution of {column_name}")

    # Calculating the mean and median of each feature
    feature_mean = dataset[column_name].mean()
    feature_median = dataset[column_name].median()

    # Plotting a line for the mean and median
    plt.axvline(feature_mean, color = 'blue', linestyle = 'dotted',
label = "Mean")
    plt.axvline(feature_median, color = 'yellow', linestyle = '-',
label = "Median")

    plt.legend()
    plt.show()


num_features = ['tenure', 'MonthlyCharges', 'TotalCharges']
for col in num_features:
    plt_hist(df, col)
```



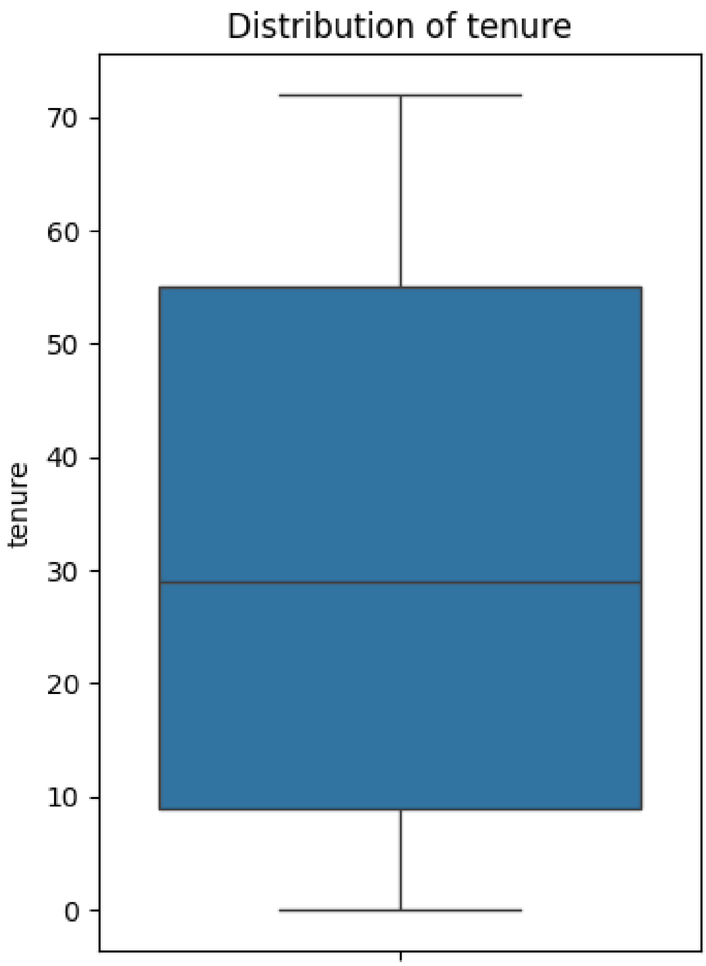


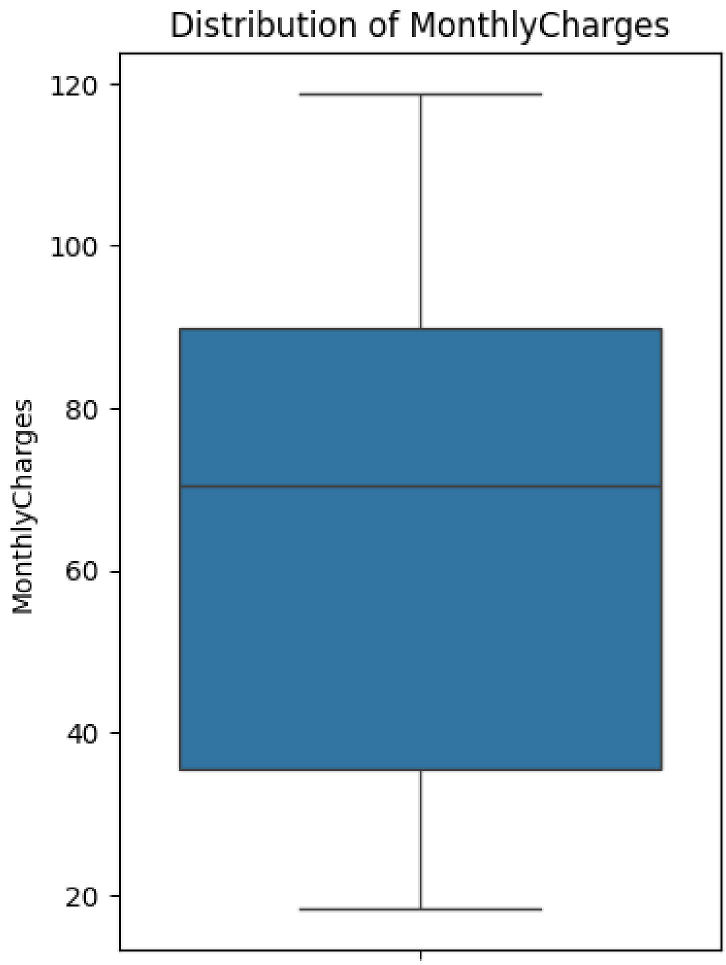


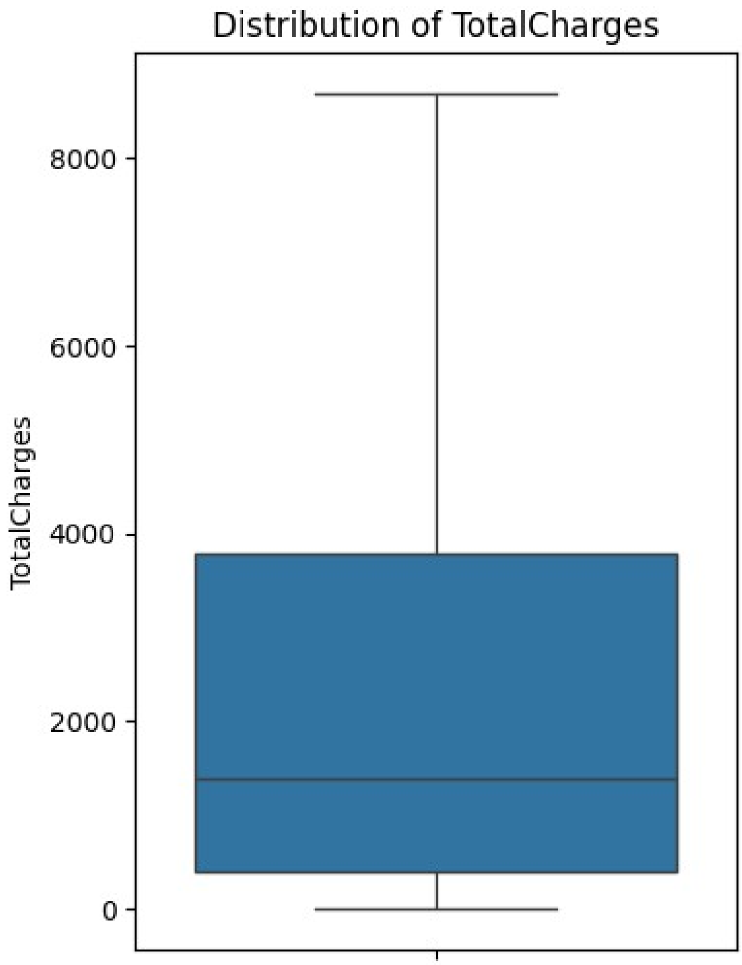
The feature 'TotalCharges' is skewed. Hence we have to scale the samples

```
def plt_box(dataset, column_name):
    plt.figure(figsize = (4, 6))
```

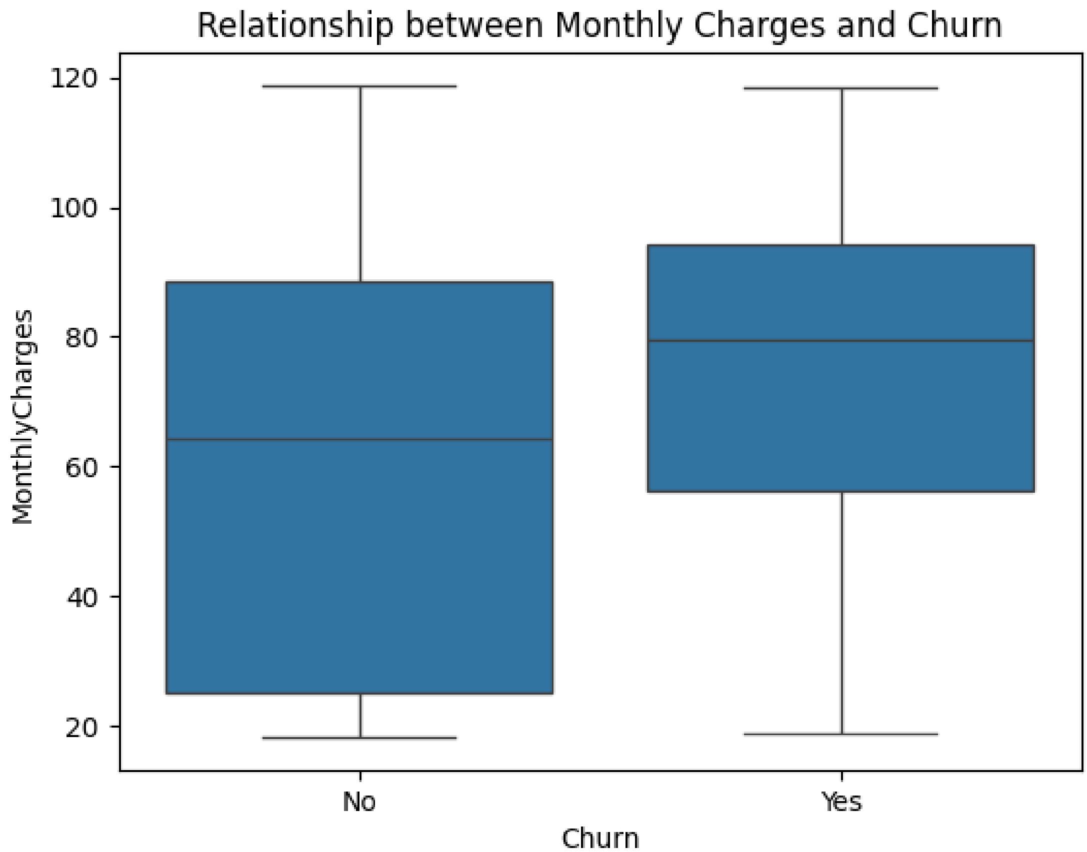
```
    sns.boxplot(y = dataset[column_name])
    plt.title(f"Distribution of {column_name}")
    plt.ylabel(column_name)
    plt.show()

for col in num_features:
    plt_box(df, col)
```




Distribution of MonthlyCharges
MonthlyCharges
120
100
80
60
40
20

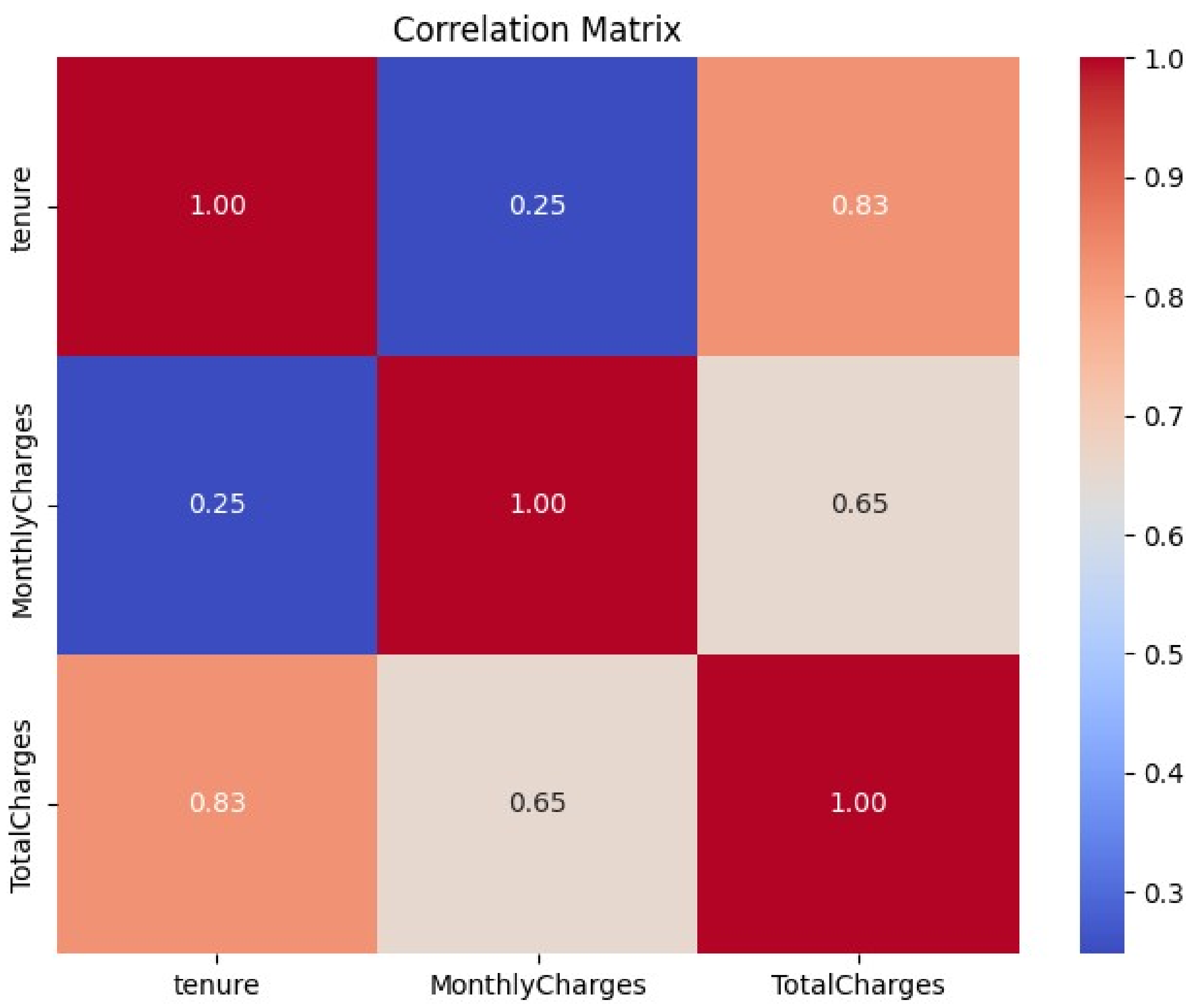



```
# Visualising the relationship between the monthly charges and churn

sns.boxplot(x = 'Churn', y = 'MonthlyCharges', data = customer_data)
plt.title('Relationship between Monthly Charges and Churn')
plt.show()
```



Insight: Customers with higher monthly charges are more likely to leave, indicating that the cost is a potential driver of churn.

```
# Correlation matrix for numerical features

plt.figure(figsize = (8, 6))
sns.heatmap(df[['tenure', 'MonthlyCharges', 'TotalCharges']].corr(),
annot = True, cmap = "coolwarm", fmt = ".2f")
plt.title("Correlation Matrix")
plt.show()
```



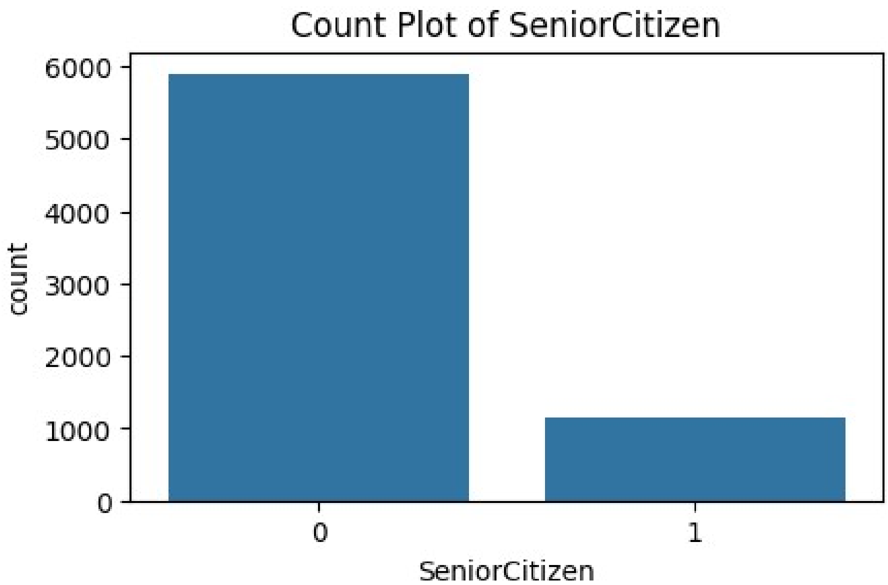
We can observe that the correlation between Tenure and Total Charges are high(0.83). We could consider drop one of the columns.

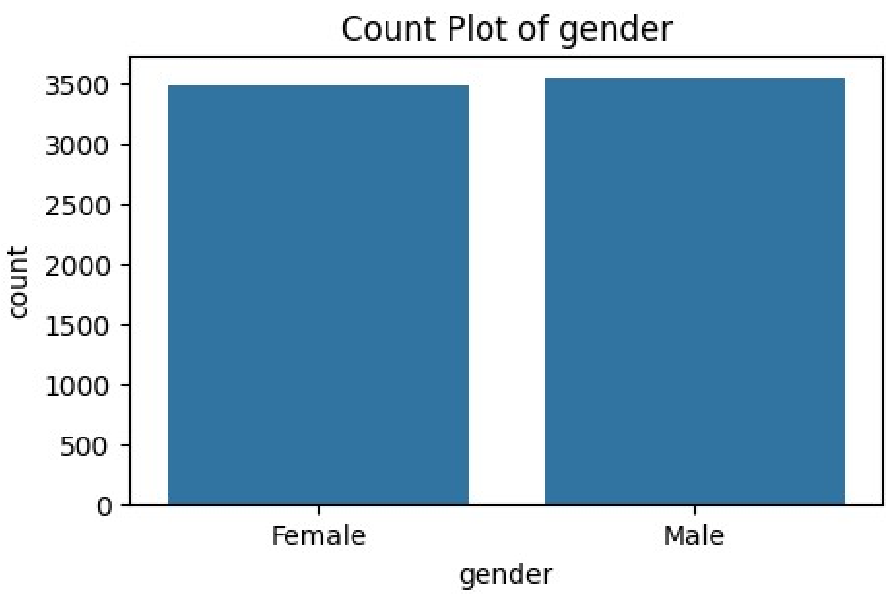
Analysing the categorical features

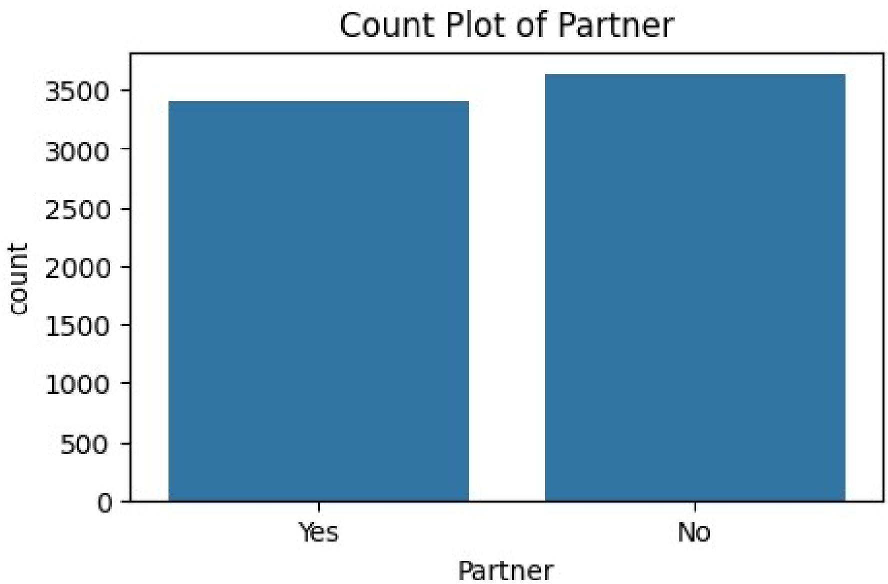
```
df.columns

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

df.info()
```

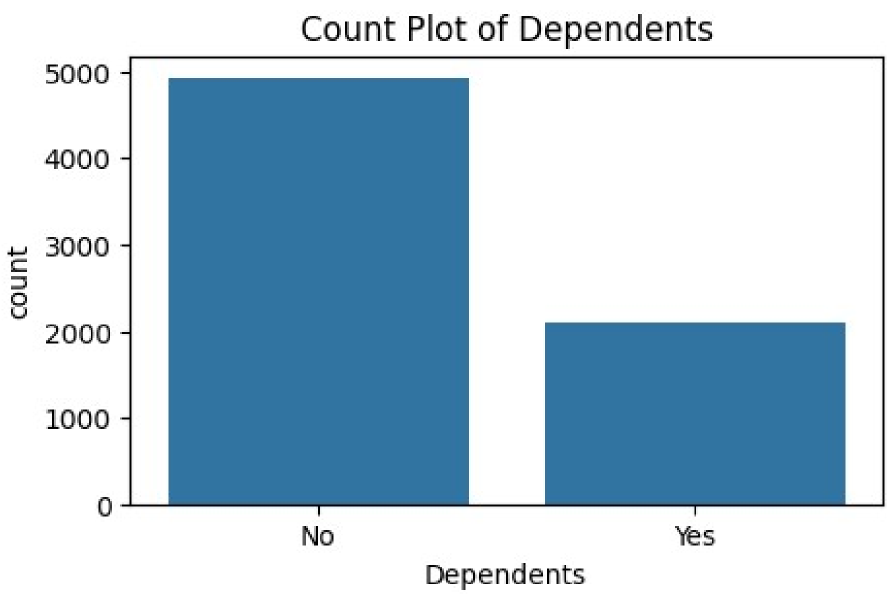
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   gender            7043 non-null   object 
 1   SeniorCitizen     7043 non-null   int64  
 2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

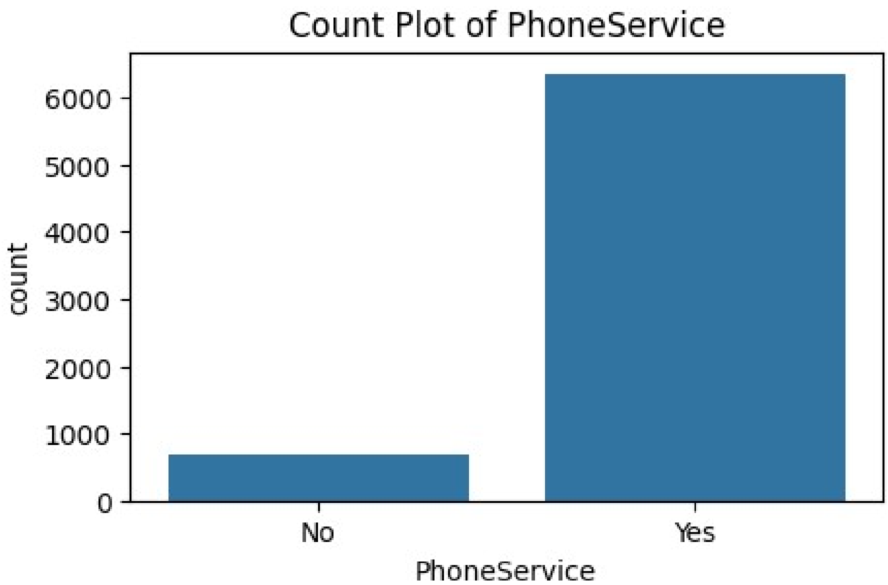
```
# Creating a list of categorical features
categorical_cols = df.select_dtypes(include =
"object").columns.to_list()

# Handling the 'SeniorCitizen' column
categorical_cols = ['SeniorCitizen'] + categorical_cols

# Plotting countplot for each feature
for col in categorical_cols:
    plt.figure(figsize = (5, 3))
    sns.countplot(x = col, data = df) # creating a count plot
    plt.title(f"Count Plot of {col}") # # Adding title to the plot
    plt.show()
```

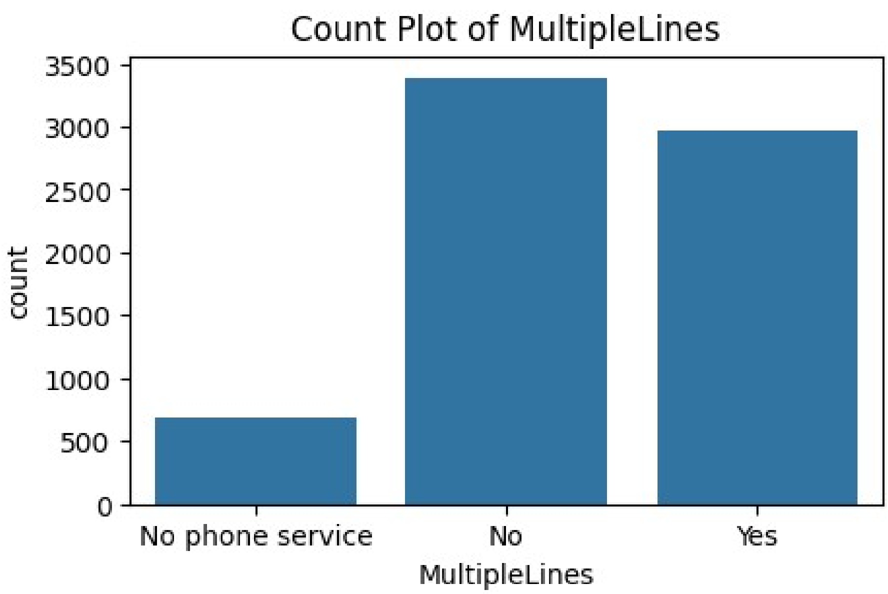

Count Plot of SeniorCitizen
6000
5000
4000
3000
2000
1000
0
count
0
1
SeniorCitizen

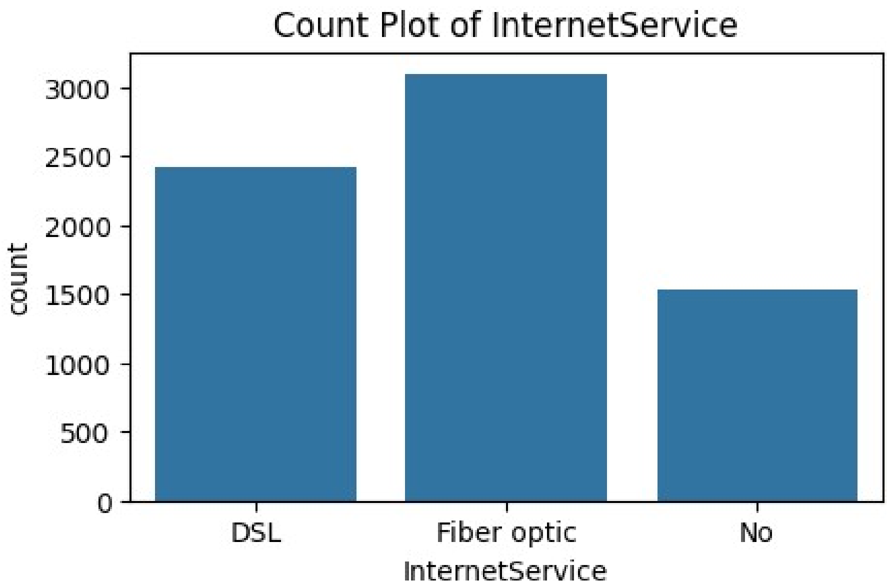

Count Plot of gender
3500
3000
2500
2000
1500
1000
500
0
count
Female
Male
gender

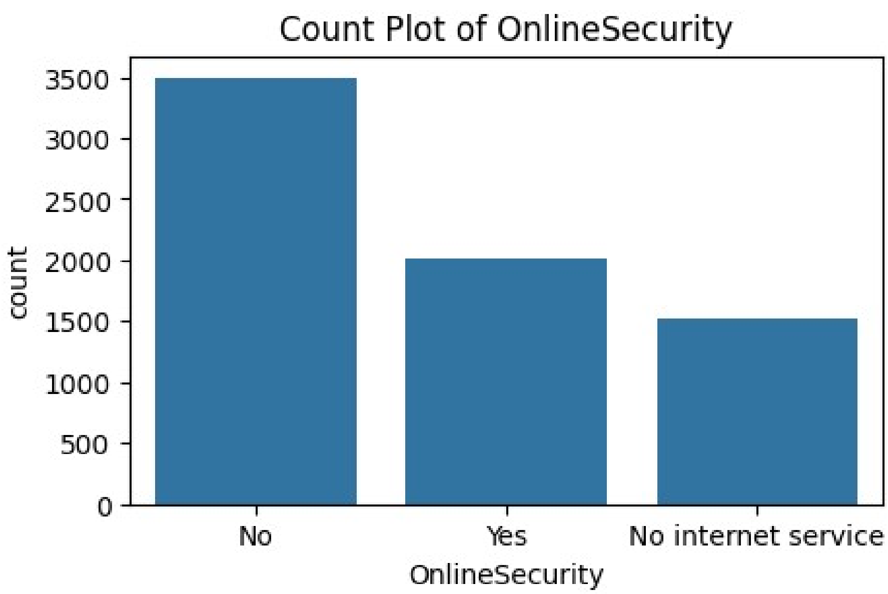

Count Plot of Partner
count
3500
3000
2500
2000
1500
1000
500
0
Yes
No
Partner

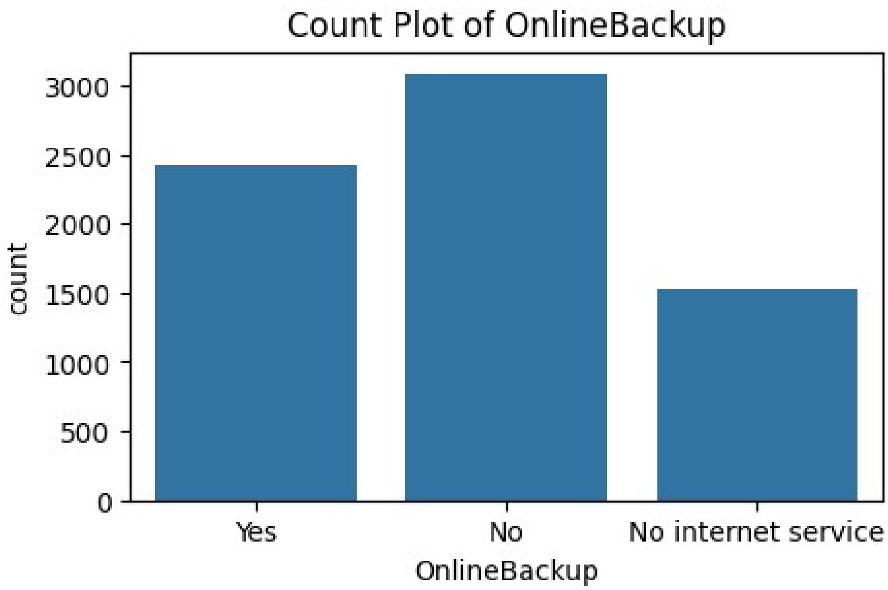

Count Plot of Dependents
count
5000
4000
3000
2000
1000
0
No
Yes
Dependents

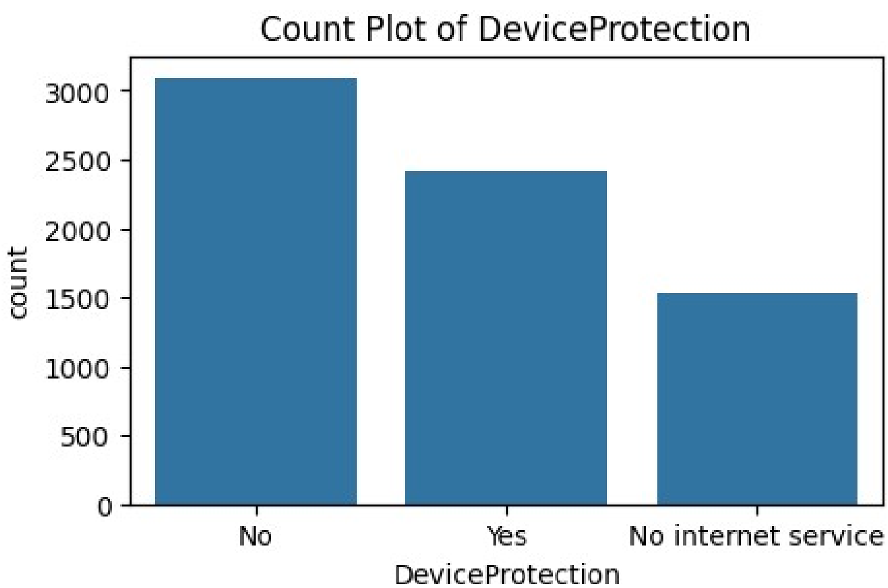

Count Plot of PhoneService
6000
5000
4000
3000
2000
1000
0
count
No
Yes
PhoneService

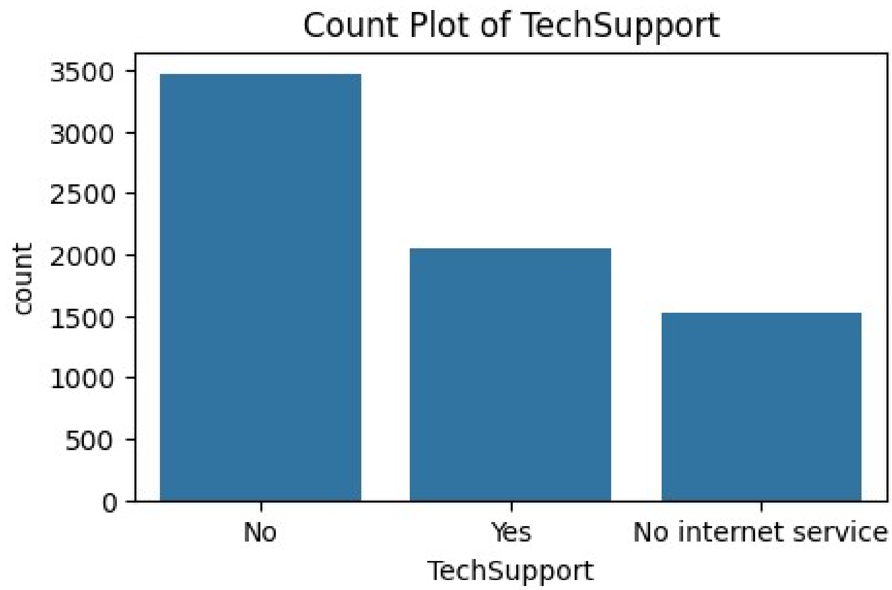

Count Plot of MultipleLines
3500
3000
2500
2000
1500
1000
500
0
count
No phone service
No
Yes
MultipleLines

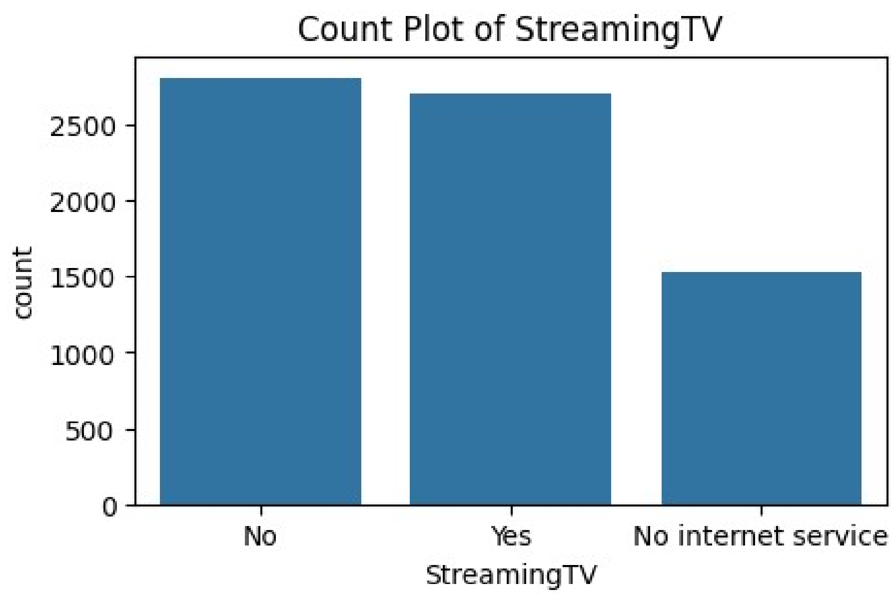

Count Plot of InternetService
3000
2500
2000
1500
1000
500
0
count
DSL
Fiber optic
No
InternetService

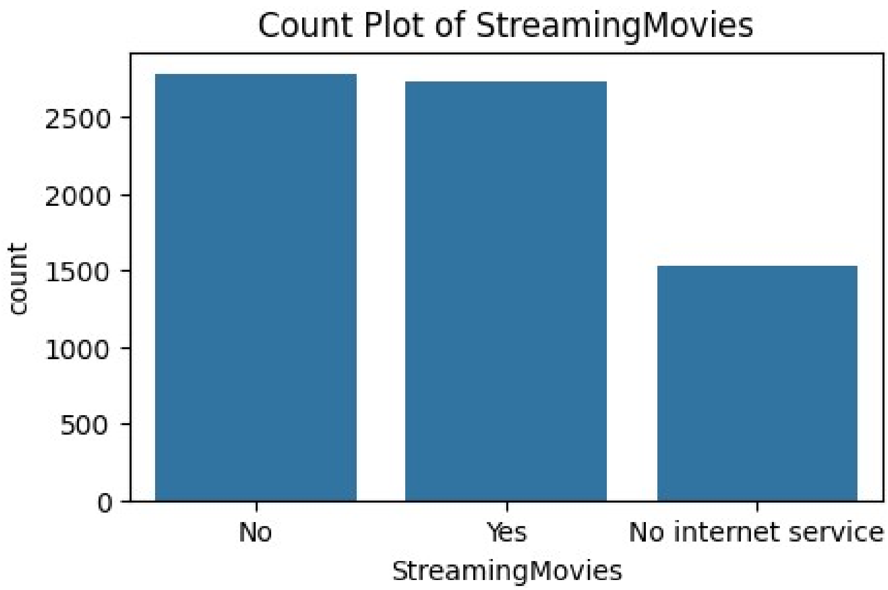

Count Plot of OnlineSecurity
3500
3000
2500
2000
1500
1000
500
0
count
No
Yes
No internet service
OnlineSecurity

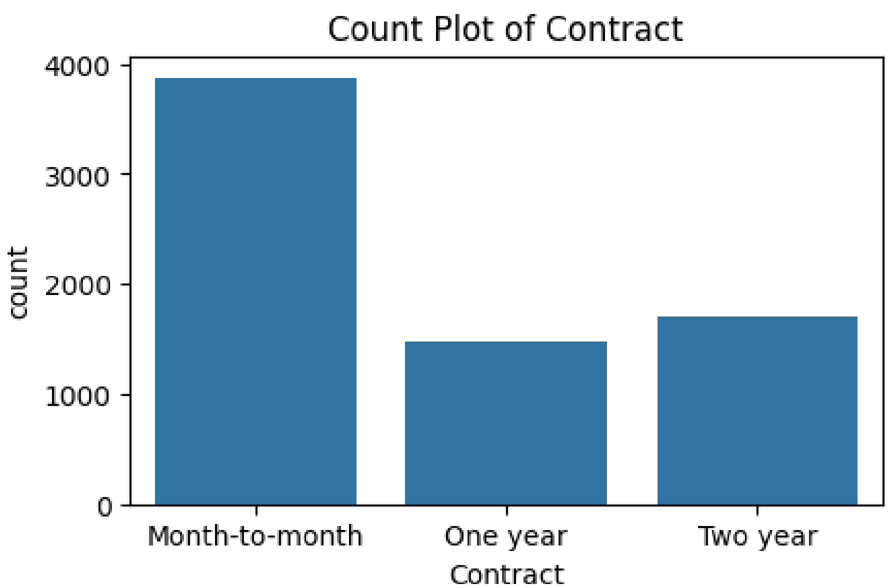

Count Plot of OnlineBackup
count
3000
2500
2000
1500
1000
500
0
Yes
No
No internet service
OnlineBackup

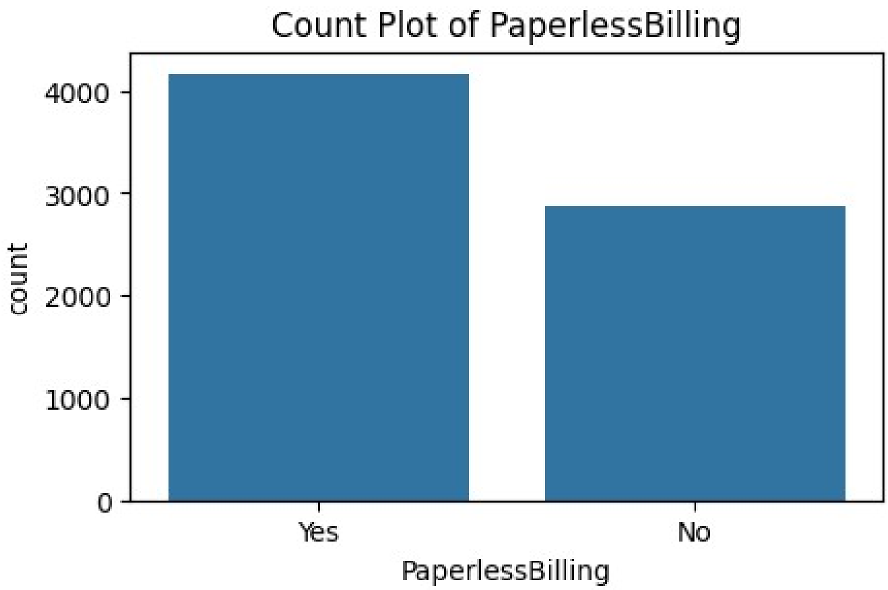

Count Plot of DeviceProtection
count
3000
2500
2000
1500
1000
500
0
No
Yes
No internet service
DeviceProtection

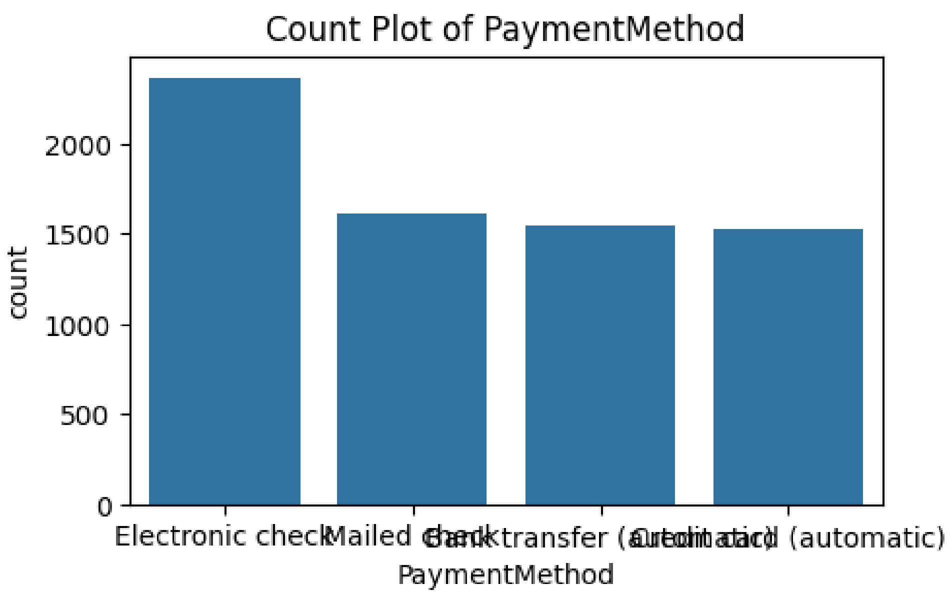

Count Plot of TechSupport
3500
3000
2500
2000
1500
1000
500
0
count
No
Yes
No internet service
TechSupport

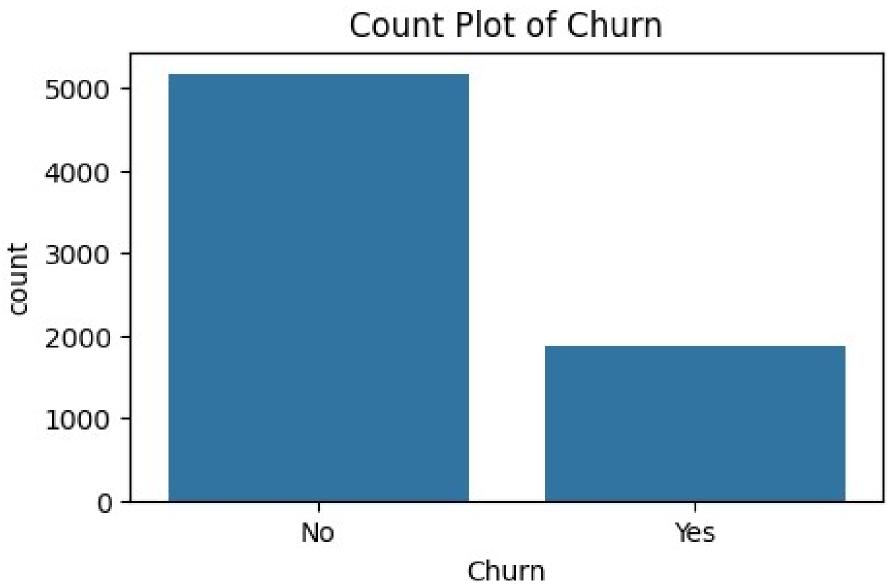

Count Plot of StreamingTV
2500
2000
1500
1000
500
0
count
No
Yes
No internet service
StreamingTV


Count Plot of StreamingMovies
count
2500
2000
1500
1000
500
0
No
Yes
No internet service
StreamingMovies


Count Plot of Contract
count
4000
3000
2000
1000
0
Month-to-month
One year
Two year
Contract


Count Plot of PaperlessBilling
4000
3000
2000
1000
0
count
Yes
No
PaperlessBilling


Count Plot of PaymentMethod
2000
1500
1000
500
0
count
Electronic check
Mailed
(automatic)
PaymentMethod



Data Preprocessing

```
# Label encoding of target

df['Churn'] = df['Churn'].replace({'Yes': 1, 'No': 0})
print(df['Churn'].value_counts())
```

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

<ipython-input-22-8bc686636493>:3: FutureWarning: Downcasting behavior
in `replace` is deprecated and will be removed in a future version. To
retain the old behavior, explicitly call
`result.infer_objects(copy=False)`. To opt-in to the future behavior,
set `pd.set_option('future.no_silent_downcasting', True)`
  df['Churn'] = df['Churn'].replace({'Yes': 1, 'No': 0})
```

Label encoding of other categorical features

```
# Identifying the feature with the Object datatype
obj_cols = df.select_dtypes(include = "object").columns
print(obj_cols)
```

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

```python
# Initialize a dictionary to save all the encoders as pickle file

encoder = {}

# Applying label encoding for each of the categorical feature
for col in obj_cols:
    label_encoder = LabelEncoder()
    df[col] = label_encoder.fit_transform(df[col])
    encoder[col] = label_encoder
    print(f"{col} has been label encoded")

# Saving the encoders to a pickle file
with open("encoder.pkl", "wb") as f:
    pickle.dump(encoder, f)
```

```
gender has been label encoded
Partner has been label encoded
Dependents has been label encoded
PhoneService has been label encoded
MultipleLines has been label encoded
InternetService has been label encoded
OnlineSecurity has been label encoded
OnlineBackup has been label encoded
DeviceProtection has been label encoded
TechSupport has been label encoded
StreamingTV has been label encoded
StreamingMovies has been label encoded
Contract has been label encoded
PaperlessBilling has been label encoded
PaymentMethod has been label encoded
```

```python
encoder
```

```
{'gender': LabelEncoder(),
 'Partner': LabelEncoder(),
 'Dependents': LabelEncoder(),
 'PhoneService': LabelEncoder(),
 'MultipleLines': LabelEncoder(),
 'InternetService': LabelEncoder(),
 'OnlineSecurity': LabelEncoder(),
 'OnlineBackup': LabelEncoder(),
 'DeviceProtection': LabelEncoder(),
 'TechSupport': LabelEncoder(),
 'StreamingTV': LabelEncoder(),
 'StreamingMovies': LabelEncoder(),
 'Contract': LabelEncoder(),
 'PaperlessBilling': LabelEncoder(),
 'PaymentMethod': LabelEncoder()}
```

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

{"summary":"{\n \"name\": \"df\",\n \"rows\": 7043,\n \"fields\":
[\n   {\n   \"column\": \"gender\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"SeniorCitizen\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Partner\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   0,\n   1\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Dependents\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"tenure\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 24,\n   \"min\": 0,\n
\"max\": 72,\n   \"num_unique_values\": 73,\n   \"samples\":
[\n   8,\n   40\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"PhoneService\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"MultipleLines\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 2,\n   \"num_unique_values\": 3,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"InternetService\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 2,\n   \"num_unique_values\": 3,\n   \"samples\":
[\n   0,\n   1\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"OnlineSecurity\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 2,\n   \"num_unique_values\": 3,\n   \"samples\":
[\n   0,\n   2\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"OnlineBackup\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 2,\n   \"num_unique_values\": 3,\n   \"samples\":
[\n   2,\n   0\n   ],\n   \"semantic_type\":

```
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"DeviceProtection\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 2,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          0,\n          2\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"TechSupport\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 2,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          0,\n          2\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"StreamingTV\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 2,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          0,\n          2\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"StreamingMovies\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 2,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          0,\n          2\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Contract\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 2,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          0,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"PaperlessBilling\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          0,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"PaymentMethod\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 0,\n
\"max\": 3,\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          3,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"MonthlyCharges\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 30.09004709767854,\n
\"min\": 18.25,\n        \"max\": 118.75,\n
\"num_unique_values\": 1585,\n        \"samples\": [\n
48.85,\n          20.05\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"TotalCharges\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 2266.7944696890195,\n
\"min\": 0.0,\n        \"max\": 8684.8,\n
\"num_unique_values\": 6531,\n        \"samples\": [\n
4600.7,\n          20.35\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Churn\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
```

```
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```


Splitting the dataset into Training and Testing sets

```
# Splitting the features and target
X = df.drop(columns = ['Churn'], axis = 1)
y = df['Churn']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state = 42)
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (5634, 19)
Shape of X_test: (1409, 19)
Shape of y_train: (5634,)
Shape of y_test: (1409,)
```

```
print("Imbalance in the Training data:", y_train.value_counts())
print("\nImbalance in the Testing data:", y_test.value_counts())
```

```
Imbalance in the Training data: Churn
0    4138
1    1496
Name: count, dtype: int64

Imbalance in the Testing data: Churn
0    1036
1     373
Name: count, dtype: int64
```

Performing SMOTE (Synthetic Minority Oversampling TEchnique)

```
# Using SMOTE to handle the imbalance in the dataset

smote = SMOTE(random_state = 42)

# Performing SMOTE on the training data
X_train_resampled, y_train_resampled = smote.fit_resample(X_train,
y_train)

print(y_train_resampled.value_counts())  # We have handled the
imbalance in the data
```

```
Churn
0    4138
1    4138
Name: count, dtype: int64
```

Model Training

```
!pip install scikeras
```

```
Collecting scikeras
  Downloading scikeras-0.13.0-py3-none-any.whl.metadata (3.1 kB)
Requirement already satisfied: keras>=3.2.0 in
/usr/local/lib/python3.11/dist-packages (from scikeras) (3.5.0)
Requirement already satisfied: scikit-learn>=1.4.2 in
/usr/local/lib/python3.11/dist-packages (from scikeras) (1.6.0)
Requirement already satisfied: absl-py in
/usr/local/lib/python3.11/dist-packages (from keras>=3.2.0->scikeras)
(1.4.0)
Requirement already satisfied: numpy in
/usr/local/lib/python3.11/dist-packages (from keras>=3.2.0->scikeras)
(1.26.4)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-
packages (from keras>=3.2.0->scikeras) (13.9.4)
Requirement already satisfied: namex in
/usr/local/lib/python3.11/dist-packages (from keras>=3.2.0->scikeras)
(0.0.8)
Requirement already satisfied: h5py in /usr/local/lib/python3.11/dist-
packages (from keras>=3.2.0->scikeras) (3.12.1)
Requirement already satisfied: optree in
/usr/local/lib/python3.11/dist-packages (from keras>=3.2.0->scikeras)
(0.13.1)
Requirement already satisfied: ml-dtypes in
/usr/local/lib/python3.11/dist-packages (from keras>=3.2.0->scikeras)
(0.4.1)
Requirement already satisfied: packaging in
/usr/local/lib/python3.11/dist-packages (from keras>=3.2.0->scikeras)
(24.2)
Requirement already satisfied: scipy>=1.6.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn>=1.4.2-
>scikeras) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn>=1.4.2-
>scikeras) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn>=1.4.2-
>scikeras) (3.5.0)
Requirement already satisfied: typing-extensions>=4.5.0 in
/usr/local/lib/python3.11/dist-packages (from optree->keras>=3.2.0-
>scikeras) (4.12.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in
```

```
/usr/local/lib/python3.11/dist-packages (from rich->keras>=3.2.0->scikeras) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
/usr/local/lib/python3.11/dist-packages (from rich->keras>=3.2.0->scikeras) (2.18.0)
Requirement already satisfied: mdurl~=0.1 in
/usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.2.0->scikeras) (0.1.2)
Downloading scikeras-0.13.0-py3-none-any.whl (26 kB)
Installing collected packages: scikeras
Successfully installed scikeras-0.13.0
```

```python
# Training with default hyperparameters
from sklearn.linear_model import LogisticRegression

# Dictionary of models
models = {
    "Decision Tree" : DecisionTreeClassifier(random_state = 42),
    "Random Forest" : RandomForestClassifier(random_state = 42),
    "KNN" : KNeighborsClassifier(),
    "Support Vector Machine" : SVC(random_state = 42),
    "Logistic Regression" : LogisticRegression(max_iter=10000,
random_state = 42),
}

for model_name, model in models.items():
    print(model_name)
    print(model)
    print("-"*50)
```

```
Decision Tree
DecisionTreeClassifier(random_state=42)
--------------------------------------------------
Random Forest
RandomForestClassifier(random_state=42)
--------------------------------------------------
KNN
KNeighborsClassifier()
--------------------------------------------------
Support Vector Machine
SVC(random_state=42)
--------------------------------------------------
Logistic Regression
LogisticRegression(max_iter=10000, random_state=42)
--------------------------------------------------
```

```python
# Dictionary to store all the cross validation results

cv_results = {}
```

```
# Performing 5 - fold cross validation for each model
for model_name, model in models.items():
    print(f"Training {model_name} with default hyperparameters")
    scores = cross_val_score(model, X_train_resampled,
y_train_resampled, cv = 5, scoring = "accuracy")
    cv_results[model_name] = scores
    print(f"{model_name} cross validation result:
{np.mean(scores):.2f}")
    print("***"*20)
```

```
Training Decision Tree with default hyperparameters
Decision Tree cross validation result: 0.78
************************************************************
Training Random Forest with default hyperparameters
Random Forest cross validation result: 0.84
************************************************************
Training KNN with default hyperparameters
KNN cross validation result: 0.77
************************************************************
Training Support Vector Machine with default hyperparameters
Support Vector Machine cross validation result: 0.64
************************************************************
Training Logistic Regression with default hyperparameters
Logistic Regression cross validation result: 0.79
************************************************************
```

```
cv_results
```

```
{'Decision Tree': array([0.68297101, 0.71299094, 0.82175227,
0.83564955, 0.83564955]),
 'Random Forest': array([0.72524155, 0.77824773, 0.90513595,
0.89425982, 0.90090634]),
 'KNN': array([0.75060386, 0.75951662, 0.78247734, 0.78731118,
0.77462236]),
 'Support Vector Machine': array([0.65519324, 0.65740181, 0.61510574,
0.61993958, 0.65015106]),
 'Logistic Regression': array([0.73007246, 0.74803625, 0.82779456,
0.81993958, 0.83625378])}
```

We observe that Random forest performs better than the other models on the dataset.

```
rfc = RandomForestClassifier(random_state = 42)
rfc.fit(X_train_resampled, y_train_resampled)
```

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

```
print(y_test.value_counts()) # We observe a imbalance in the target
classes.
```

```
Churn
0    1036
1     373
Name: count, dtype: int64
```

```
y_pred = rfc.predict(X_test)

print("Accuracy Score: ", accuracy_score(y_test, y_pred))

# Since there is imbalance in the test data, the accuracy is not highly reliable.
```

```
Accuracy Score:  0.7785663591199432
```

```
confusion_matrix(y_test, y_pred)
```

```
array([[878, 158],
       [154, 219]])
```

Out of 1036 samples from class 0, the model that predicted class 0 878 times while class 1 has been predicted correctly 219 times out of 373 times.

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

# We see a clear difference between the performance scores of class 1 and 0 because of the class imbalance.
```

```
              precision    recall  f1-score   support

           0       0.85      0.85      0.85      1036
           1       0.58      0.59      0.58       373

    accuracy                           0.78      1409
   macro avg       0.72      0.72      0.72      1409
weighted avg       0.78      0.78      0.78      1409
```

```
# Saving the model and feature names as a pickle file

model_data = {"model": rfc, "feature_names": X.columns.tolist()}

with open("customer_churn.pkl", "wb") as f:
    pickle.dump(model_data, f)
```

Building a predictive System

```
with open("customer_churn.pkl", "rb") as f:
    model_data = pickle.load(f)

loaded_model = model_data["model"]
feature_names = model_data["feature_names"]
```

```python
print(loaded_model)
```

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

```python
print(feature_names)
```

```
['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure',
'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity',
'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',
'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod',
'MonthlyCharges', 'TotalCharges']
```

```python
input_data = {
    "gender": "Female",
    "SeniorCitizen": 0,
    "Partner": "Yes",
    "Dependents": "No",
    "tenure": 1,
    "PhoneService": "No",
    "MultipleLines": "No phone service",
    "InternetService": "DSL",
    "OnlineSecurity": "No",
    "OnlineBackup": "Yes",
    "DeviceProtection": "No",
    "TechSupport": "No",
    "StreamingTV": "No",
    "StreamingMovies": "No",
    "Contract": "Month-to-month",
    "PaperlessBilling": "Yes",
    "PaymentMethod": "Electronic check",
    "MonthlyCharges": 29.85,
    "TotalCharges": 29.85
}

input_df = pd.DataFrame([input_data])

with open("encoder.pkl", "rb") as f:
    encoders = pickle.load(f)

print(input_df)
```

```
   gender  SeniorCitizen Partner Dependents  tenure PhoneService  \
0  Female              0     Yes         No       1           No

      MultipleLines InternetService OnlineSecurity OnlineBackup  \
0  No phone service             DSL             No          Yes

  DeviceProtection TechSupport StreamingTV StreamingMovies
Contract  \
0               No          No          No              No  Month-to-
month
```

```
  PaperlessBilling     PaymentMethod  MonthlyCharges  TotalCharges
0              Yes  Electronic check           29.85         29.85
```

```
# Encoding the input data
for col, encoder in encoders.items():
    input_df[col] = encoder.transform(input_df[col])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/sklearn/utils/_encode.py in _encode(values, uniques, check_unknown)
    234         try:
--> 235             return _map_to_integer(values, uniques)
    236         except KeyError as e:

/usr/local/lib/python3.11/dist-packages/sklearn/utils/_encode.py in _map_to_integer(values, uniques)
    173     table = _nandict({val: i for i, val in enumerate(uniques)})
--> 174     return xp.asarray([table[v] for v in values], device=device(values))
    175

/usr/local/lib/python3.11/dist-packages/sklearn/utils/_encode.py in <listcomp>(.0)
    173     table = _nandict({val: i for i, val in enumerate(uniques)})
--> 174     return xp.asarray([table[v] for v in values], device=device(values))
    175

/usr/local/lib/python3.11/dist-packages/sklearn/utils/_encode.py in __missing__(self, key)
    166             return self.nan_value
--> 167         raise KeyError(key)
    168

KeyError: 0

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
<ipython-input-103-77678f2917a6> in <cell line: 0>()
      1 # Encoding the input data
      2 for col, encoder in encoders.items():
----> 3     input_df[col] = encoder.transform(input_df[col])
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_label.p
y in transform(self, y)
    132             return xp.asarray([])
    133 
--> 134         return _encode(y, uniques=self.classes_)
    135 
    136     def inverse_transform(self, y):

/usr/local/lib/python3.11/dist-packages/sklearn/utils/_encode.py in
_encode(values, uniques, check_unknown)
    235             return _map_to_integer(values, uniques)
    236         except KeyError as e:
--> 237             raise ValueError(f"y contains previously unseen
labels: {str(e)}")
    238     else:
    239         if check_unknown:

ValueError: y contains previously unseen labels: 0
```

```
input_df.head()
```

{"summary":"{\n \"name\": \"input_df\",\n \"rows\": 1,\n
\"fields\": [\n   {\n   \"column\": \"gender\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
null,\n   \"min\": 0,\n   \"max\": 0,\n
\"num_unique_values\": 1,\n   \"samples\": [\n   0\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"SeniorCitizen\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
null,\n   \"min\": 0,\n   \"max\": 0,\n
\"num_unique_values\": 1,\n   \"samples\": [\n   0\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"Partner\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
null,\n   \"min\": 1,\n   \"max\": 1,\n
\"num_unique_values\": 1,\n   \"samples\": [\n   1\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"Dependents\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
null,\n   \"min\": 0,\n   \"max\": 0,\n
\"num_unique_values\": 1,\n   \"samples\": [\n   0\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"tenure\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\": null,\n
\"min\": 1,\n   \"max\": 1,\n   \"num_unique_values\": 1,\n
\"samples\": [\n   1\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"PhoneService\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 0,\n

```
\"max\": 0,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"MultipleLines\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": null,\n        \"min\": 1,\n
\"max\": 1,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"InternetService\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": null,\n        \"min\": 0,\n
\"max\": 0,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"OnlineSecurity\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": null,\n        \"min\": 0,\n
\"max\": 0,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"OnlineBackup\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": null,\n        \"min\": 2,\n
\"max\": 2,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          2\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"DeviceProtection\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": null,\n        \"min\": 0,\n
\"max\": 0,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"TechSupport\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": null,\n        \"min\": 0,\n
\"max\": 0,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"StreamingTV\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": null,\n        \"min\": 0,\n
\"max\": 0,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"StreamingMovies\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": null,\n        \"min\": 0,\n
\"max\": 0,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Contract\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": null,\n        \"min\": 0,\n
\"max\": 0,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PaperlessBilling\",\n      \"properties\": {\n        \"dtype\":
```

```
\"number\",\n        \"std\": null,\n        \"min\": 1,\n
\"max\": 1,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PaymentMethod\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": null,\n        \"min\": 2,\n
\"max\": 2,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          2\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"MonthlyCharges\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": null,\n        \"min\": 29.85,\n
\"max\": 29.85,\n        \"num_unique_values\": 1,\n
\"samples\": [\n          29.85\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"TotalCharges\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
null,\n        \"min\": 29.85,\n        \"max\": 29.85,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          29.85\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"input_df"}
```


```
encoders
```

```
{'gender': LabelEncoder(),
 'Partner': LabelEncoder(),
 'Dependents': LabelEncoder(),
 'PhoneService': LabelEncoder(),
 'MultipleLines': LabelEncoder(),
 'InternetService': LabelEncoder(),
 'OnlineSecurity': LabelEncoder(),
 'OnlineBackup': LabelEncoder(),
 'DeviceProtection': LabelEncoder(),
 'TechSupport': LabelEncoder(),
 'StreamingTV': LabelEncoder(),
 'StreamingMovies': LabelEncoder(),
 'Contract': LabelEncoder(),
 'PaperlessBilling': LabelEncoder(),
 'PaymentMethod': LabelEncoder()}
```

```python
# Making a prediction using the new input data
predicted_result = loaded_model.predict(input_df)
print("Predicted output:", predicted_result)

predicted_prob = loaded_model.predict_proba(input_df)
print("Prediction Probablility:", predicted_prob)

if predicted_result[0] == 1:
    print("Customer will churn")
else:
    print("Customer will not churn")
```

```
Predicted output: [0]
Prediction Probablility: [[0.78 0.22]]
Customer will not churn
```