

MACHINE LEARNING WITH PYTHON







1.Importing the dependencies

```
In [104... import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import pickle
```

2. Data Loading and Understanding

```
In [107... # load thecsv data to a pandas dataframe
df = pd.read_csv("WA_Fn-UseC_-Telco-Customer-Churn.csv")
```

```
In [109... df.shape
```

```
Out[109... (7043, 21)
```

```
In [110... df.head()
```

```
Out[110...  customerID  gender  SeniorCitizen  Partner  Dependents  tenure  PhoneService  MultipleLines  InternetService  OnlineSecurity  C
0      7590-  Female            0      Yes           No         1           No      No phone  DSL              No
   VHVVEG
1      5575-  Male             0      No           No        34          Yes       No      DSL              Yes
   GNVDE
2      3668-  Male             0      No           No         2          Yes       No      DSL              Yes
   QPYBK
3      7795-  Male             0      No           No        45          No      No phone  DSL              Yes
   CFOCW
4      9237-  Female            0      No           No         2          Yes       No      Fiber optic No
   HQTU
```

```
In [111... df.tail()
```

```
Out[111...  customerID  gender  SeniorCitizen  Partner  Dependents  tenure  PhoneService  MultipleLines  InternetService  OnlineSecurity
7038    6840-  Male             0      Yes           Yes        24          Yes       Yes      DSL              Yes
   RESVB
7039    2234-  Female            0      Yes           Yes        72          Yes       Yes      Fiber optic No
   XADUH
7040    4801-  Female            0      Yes           Yes        11          No      No phone  DSL              Yes
   JZAZL
7041    8361-  Male             1      Yes           No         4          Yes       Yes      Fiber optic No
   LTMKD
7042   3186-AJIEK  Male             0      No           No        66          Yes       No      Fiber optic Yes
```

```
In [112... pd.set_option("display.max_columns", None)
```

```
In [113... df.head(2)
```

```
Out[113...  customerID  gender  SeniorCitizen  Partner  Dependents  tenure  PhoneService  MultipleLines  InternetService  OnlineSecurity  C
0      7590-  Female            0      Yes           No         1           No      No phone  DSL              No
   VHVVEG
1      5575-  Male             0      No           No        34          Yes       No      DSL              Yes
   GNVDE
```

```
In [114... 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 [115.. # dropping customerID column as this is not required for modelling
df = df.drop(columns=["customerID"])

```

```

In [116.. df.head(2)

```

```

Out[116..
   gender  SeniorCitizen  Partner  Dependents  tenure  PhoneService  MultipleLines  InternetService  OnlineSecurity  OnlineBackup
0  Female              0     Yes         No         1           No      No phone service         DSL              No              Yes
1   Male              0     No         No        34          Yes           No         DSL              Yes              No

```

```

In [117.. df.columns

```

```

Out[117.. Index(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure',
              'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity',
              'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',
              'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod',
              'MonthlyCharges', 'TotalCharges', 'Churn'],
              dtype='object')

```

```

In [118.. print(df["gender"].unique())

['Female' 'Male']

```

```

In [119.. print(df["SeniorCitizen"].unique())

[0 1]

```

```

In [120.. # printing the unique values in all the columns

numerical_features_list = ["tenure", "MonthlyCharges", "TotalCharges"]

for col in df.columns:
    if col not in numerical_features_list:
        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']
-----

```

```
In [121.. print(df.isnull().sum())
```

```

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

```

```
In [122.. #df["TotalCharges"] = df["TotalCharges"].astype(float)
```

```
In [123.. df[df["TotalCharges"]==" "]
```

Out [123...

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBack
488	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	
753	Male	0	No	Yes	0	Yes	No	No	No internet service	No inter serv
936	Female	0	Yes	Yes	0	Yes	No	DSL	Yes	Y
1082	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service	No inter serv
1340	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	Y
3331	Male	0	Yes	Yes	0	Yes	No	No	No internet service	No inter serv
3826	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service	No inter serv
4380	Female	0	Yes	Yes	0	Yes	No	No	No internet service	No inter serv
5218	Male	0	Yes	Yes	0	Yes	No	No	No internet service	No inter serv
6670	Female	0	Yes	Yes	0	Yes	Yes	DSL	No	Y
6754	Male	0	No	Yes	0	Yes	Yes	DSL	Yes	Y

In [124...

len(df[df["TotalCharges"]== " "])

Out[124...

11

In [125...

df["TotalCharges"] = df["TotalCharges"].replace({" ": "0.0"})

In [126...

df["TotalCharges"] = df["TotalCharges"].astype(float)

In [127...

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

In [128...

checking the class distribution of target column
print(df["Churn"].value_counts())

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

Insights:

Customer ID removed as it is not required for modelling

No missing values in the dataset

Missing values in the TotalCharges column were replaced with 0

Class imbalance identified in the target

3.. Exploratory Data Analysis (EDA)

```
In [129.. df.shape
```

```
Out[129.. (7043, 20)
```

```
In [130.. df.columns
```

```
Out[130.. Index(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure',  
        'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity',  
        'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',  
        'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod',  
        'MonthlyCharges', 'TotalCharges', 'Churn'],  
        dtype='object')
```

```
In [131.. df.head(2)
```

```
Out[131..
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup
0	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes
1	Male	0	No	No	34	Yes	No	DSL	Yes	No

```
In [133.. df.describe()
```

```
Out[133..
```

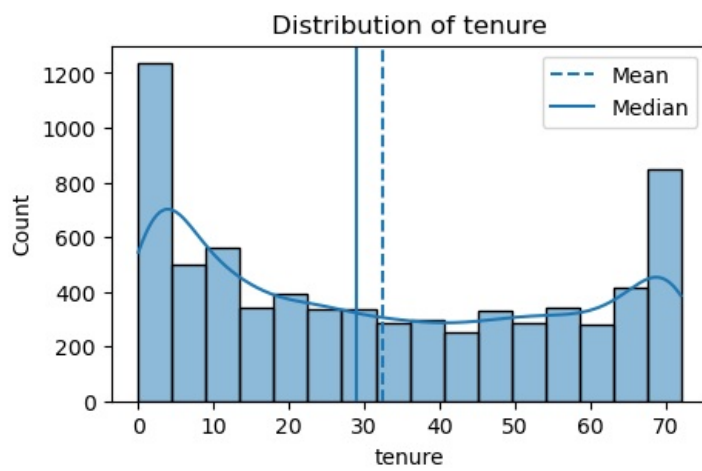
	SeniorCitizen	tenure	MonthlyCharges	TotalCharges
count	7043.000000	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692	2279.734304
std	0.368612	24.559481	30.090047	2266.794470
min	0.000000	0.000000	18.250000	0.000000
25%	0.000000	9.000000	35.500000	398.550000
50%	0.000000	29.000000	70.350000	1394.550000
75%	0.000000	55.000000	89.850000	3786.600000
max	1.000000	72.000000	118.750000	8684.800000

Numerical Features - Analysis

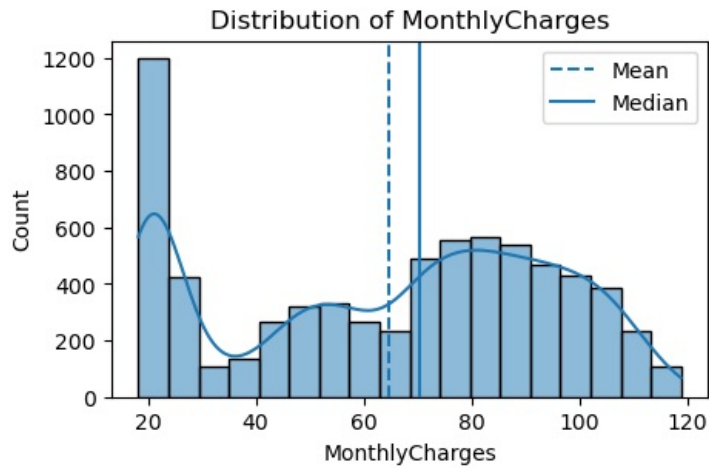
Understand the distribution of teh numerical features

```
In [135.. def plot_histogram(df, column_name):  
  
    plt.figure(figsize=(5, 3))  
    sns.histplot(df[column_name], kde=True)  
    plt.title(f"Distribution of {column_name}")  
  
    # calculate mean and median  
    col_mean = df[column_name].mean()  
    col_median = df[column_name].median()  
  
    # add vertical lines  
    plt.axvline(col_mean, linestyle="--", label="Mean")  
    plt.axvline(col_median, linestyle="-", label="Median")  
  
    plt.legend()  
    plt.show()
```

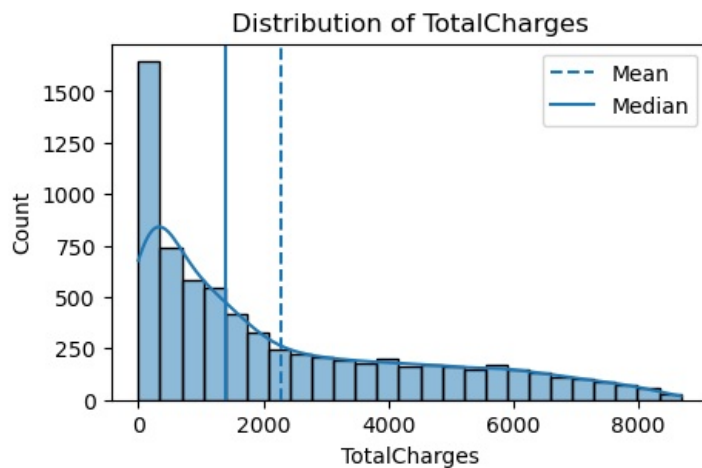
```
In [136.. plot_histogram(df, "tenure")
```

```
In [137.. plot_histogram(df, "MonthlyCharges")
```



```
In [138.. plot_histogram(df, "TotalCharges")
```



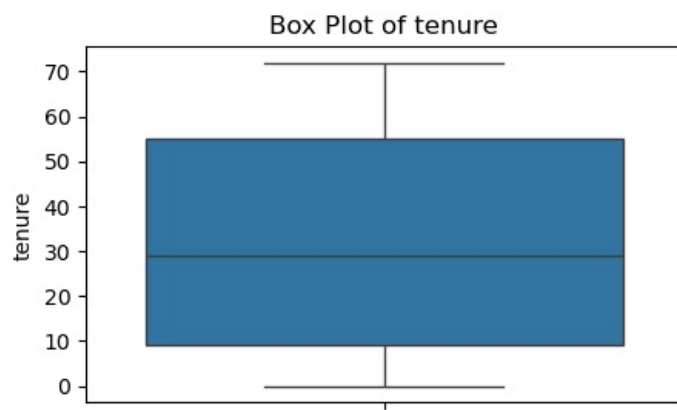
Box plot for numerical features

```
In [141.. import matplotlib.pyplot as plt

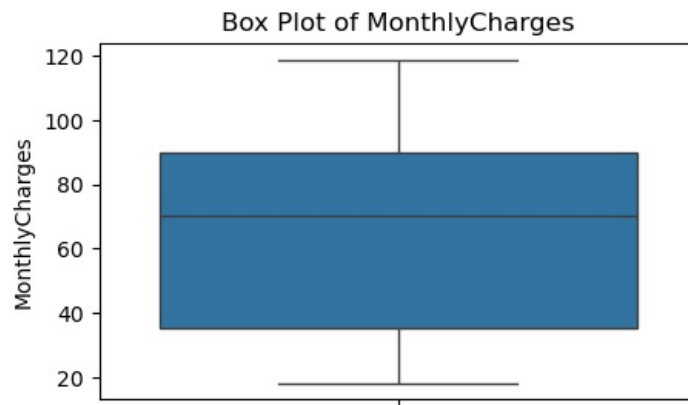
def plot_boxplot(df, column_name):

    plt.figure(figsize=(5, 3))
    sns.boxplot(y=df[column_name])
    plt.title(f"Box Plot of {column_name}")
    plt.ylabel(column_name)
    plt.show()
```

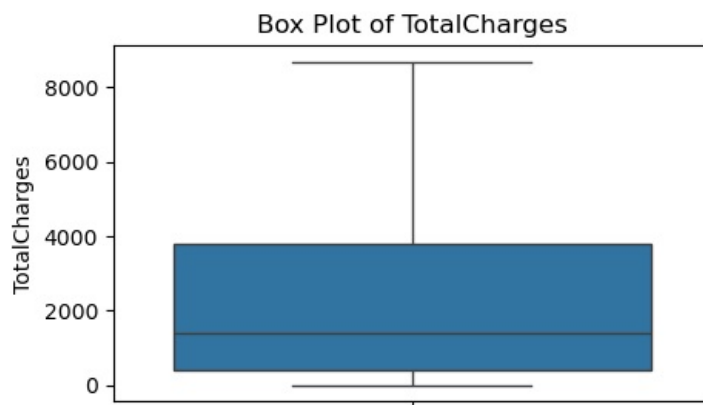
```
In [142.. plot_boxplot(df, "tenure")
```



```
In [143... plot_boxplot(df, "MonthlyCharges")
```

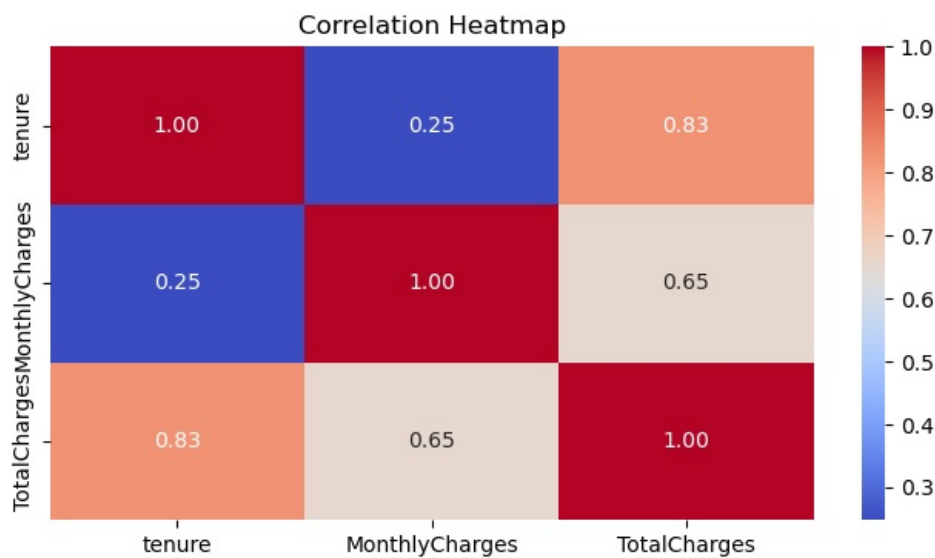


```
In [144... plot_boxplot(df, "TotalCharges")
```



Correlation Heatmap for numerical columns

```
In [145... # correlation matrix - heatmap
plt.figure(figsize=(8, 4))
sns.heatmap(df[["tenure", "MonthlyCharges", "TotalCharges"]].corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```



Categorical features - Analysis

```
In [147...] df.columns
```

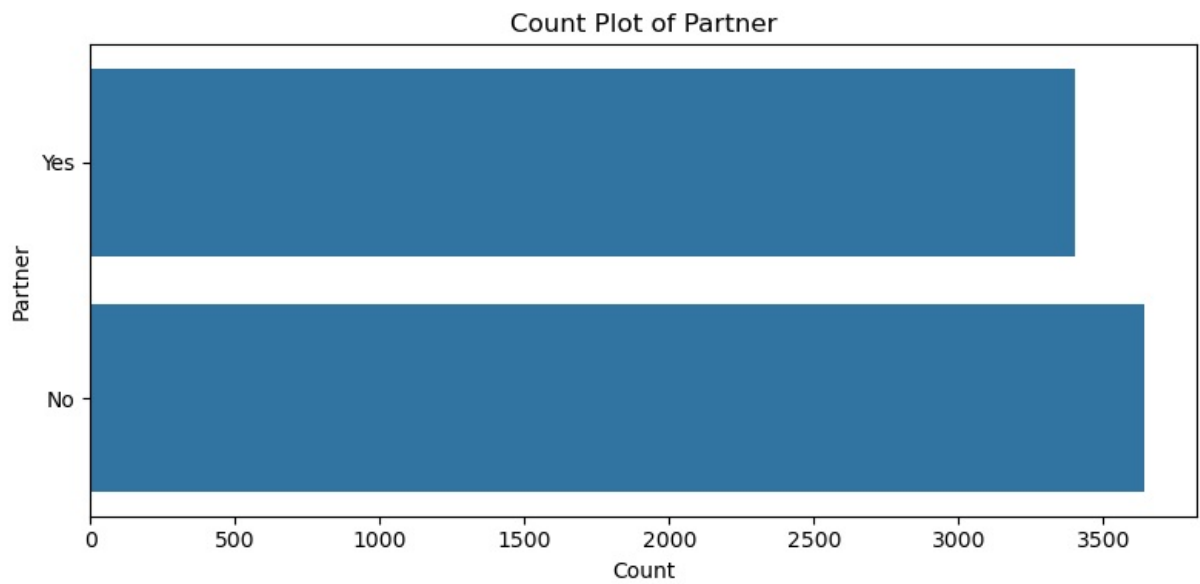
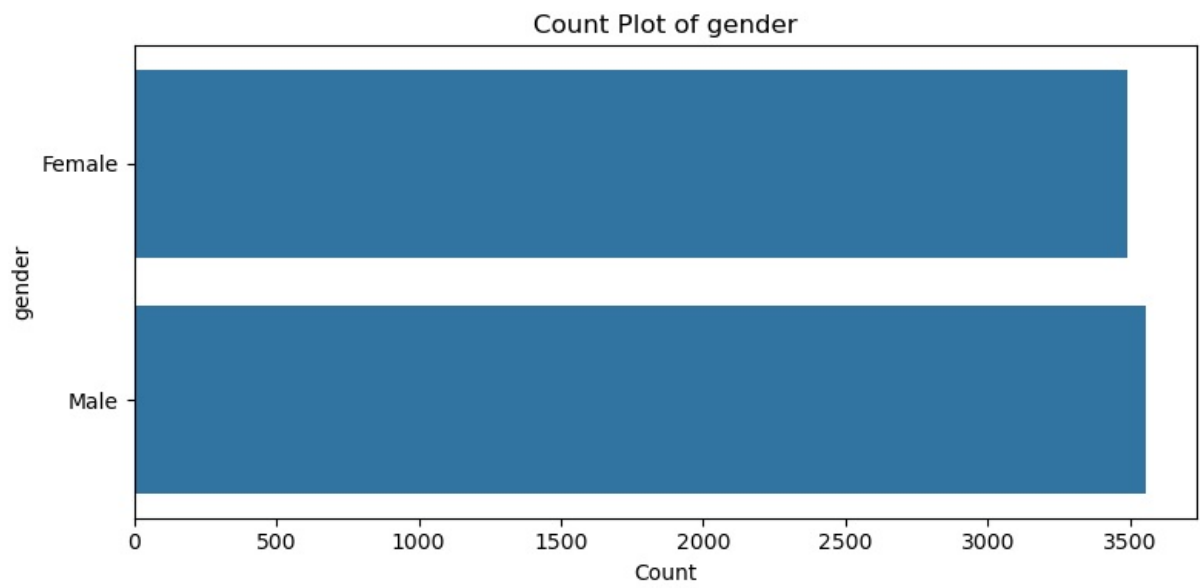
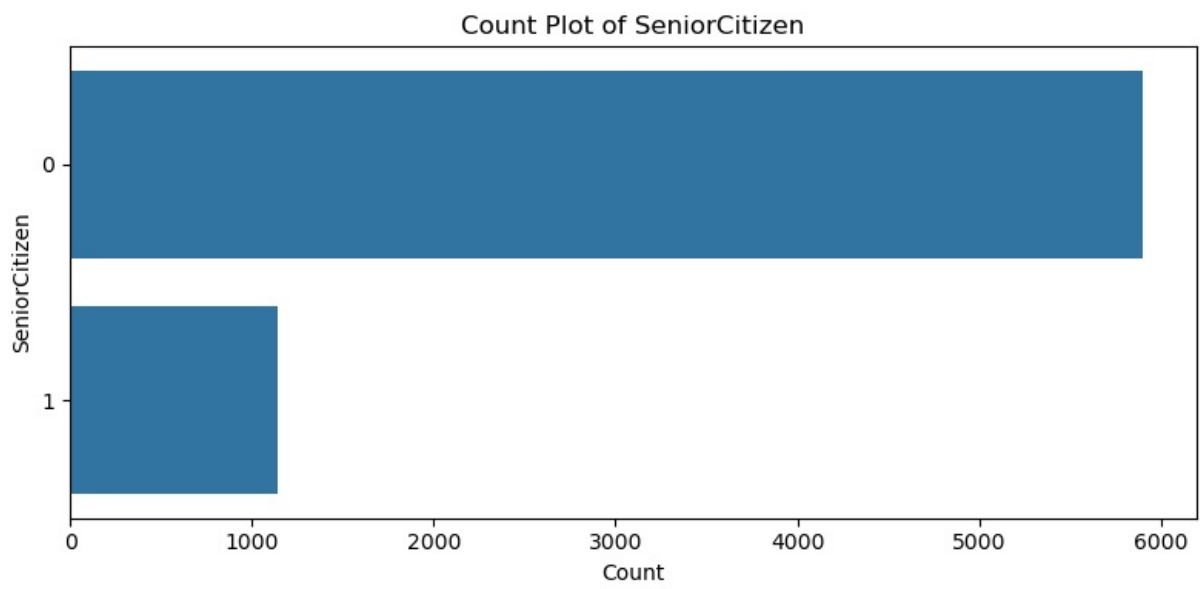
```
Out[147...] Index(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure',
                  'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity',
                  'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',
                  'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod',
                  'MonthlyCharges', 'TotalCharges', 'Churn'],
                  dtype='object')
```

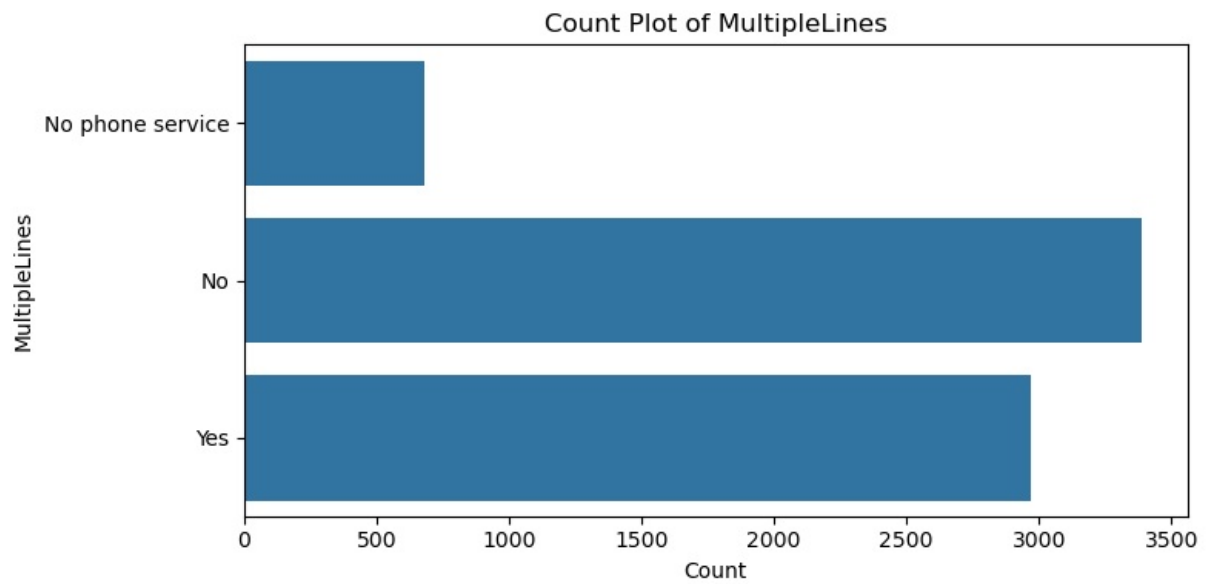
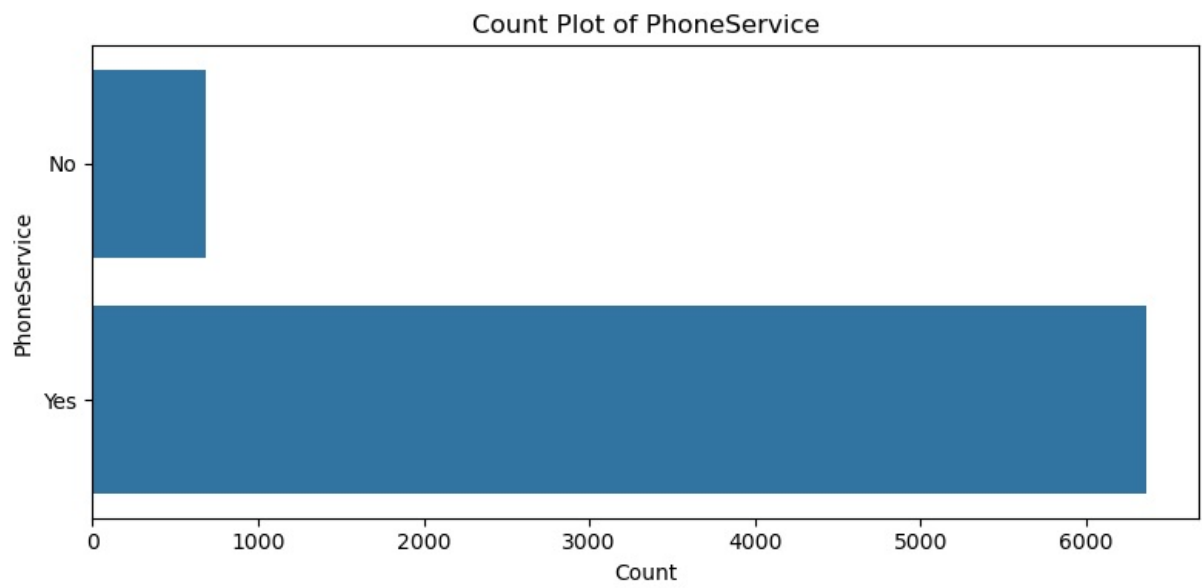
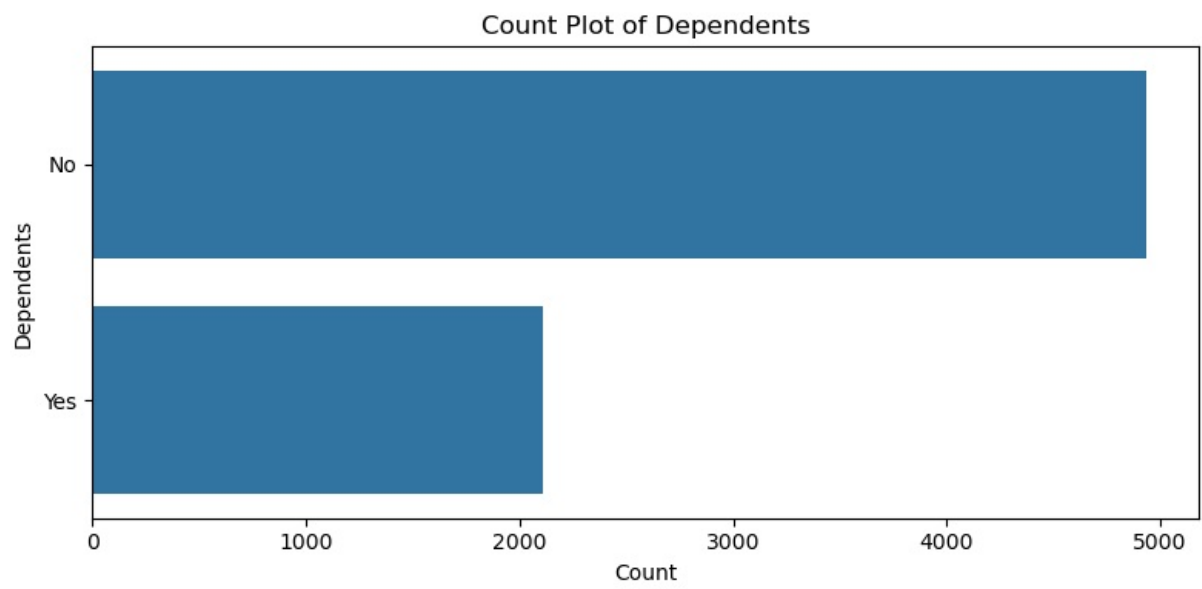
```
In [148...] df.info()
```

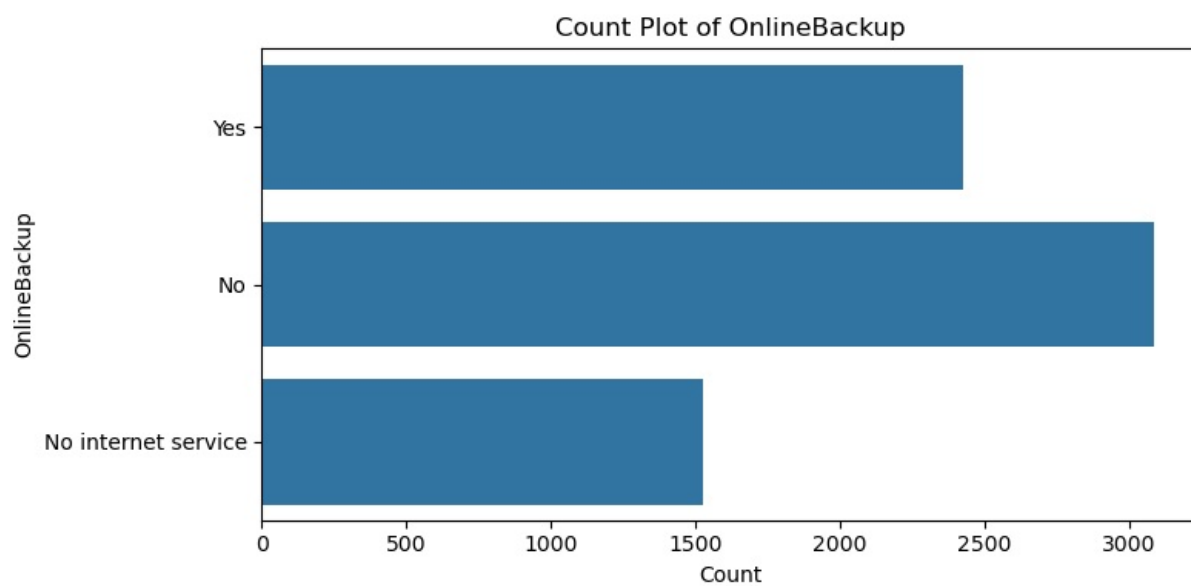
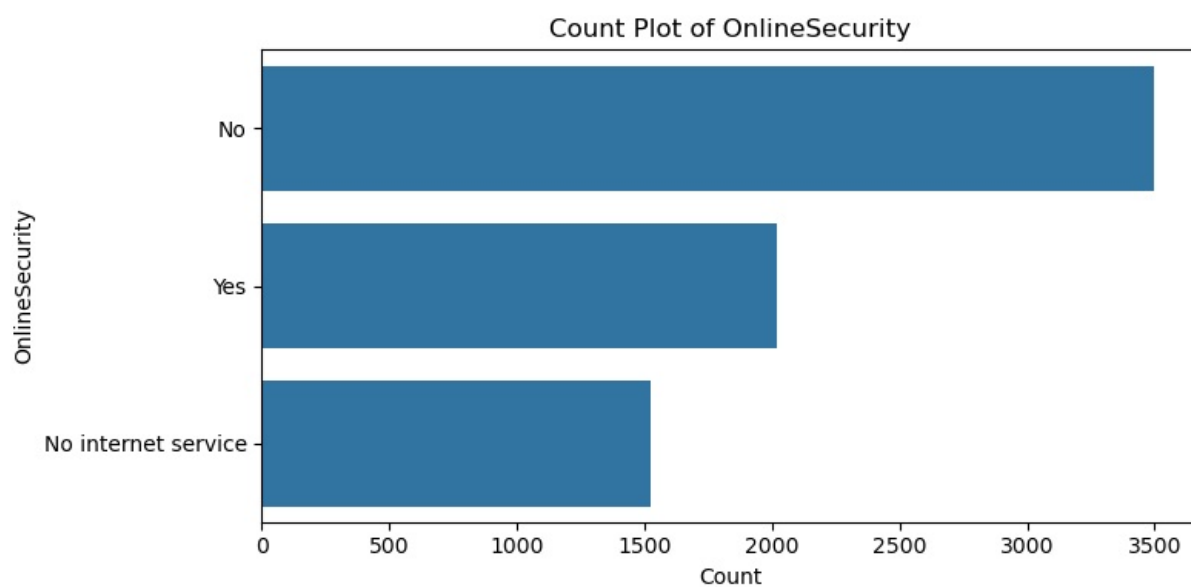
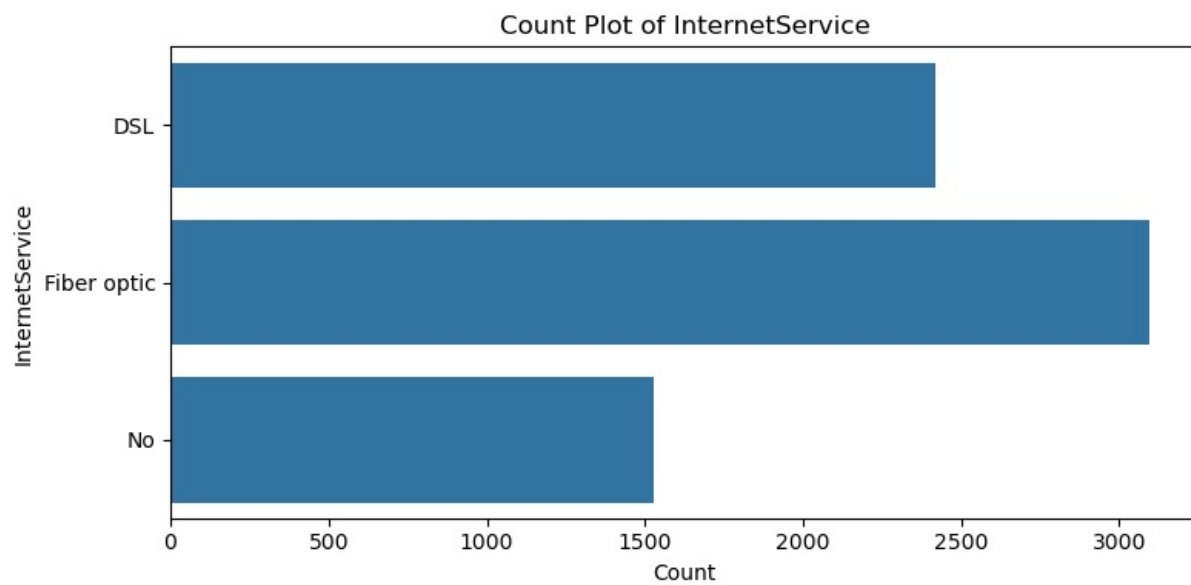
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
#   Column                Non-Null Count  Dtype
---  -
0   gender                 7043 non-null  object
1   SeniorCitizen          7043 non-null  int64
2   Partner                7043 non-null  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
```

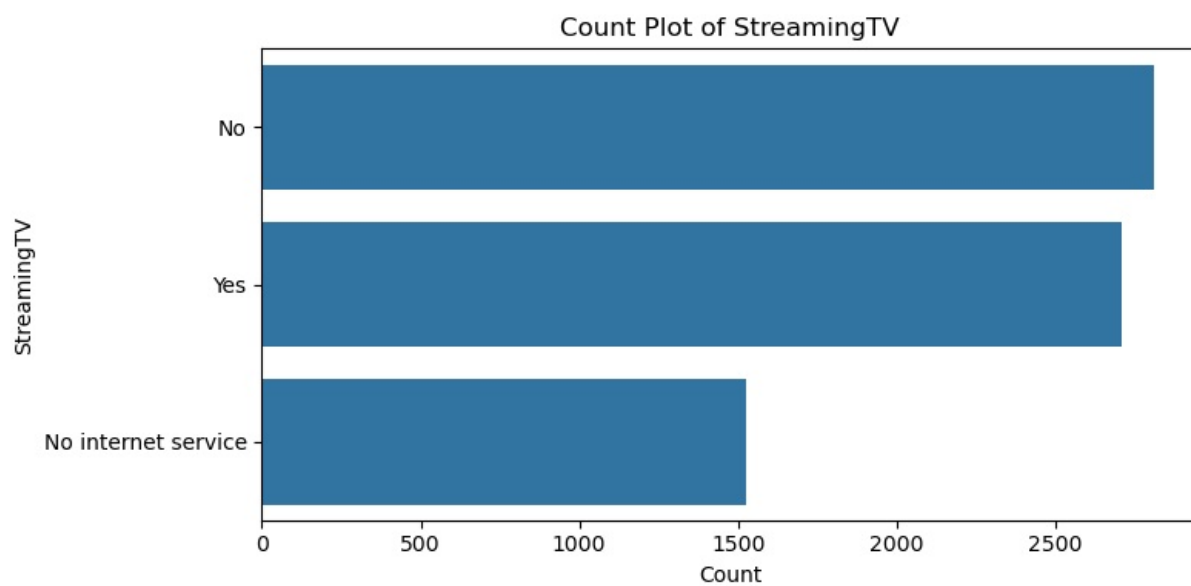
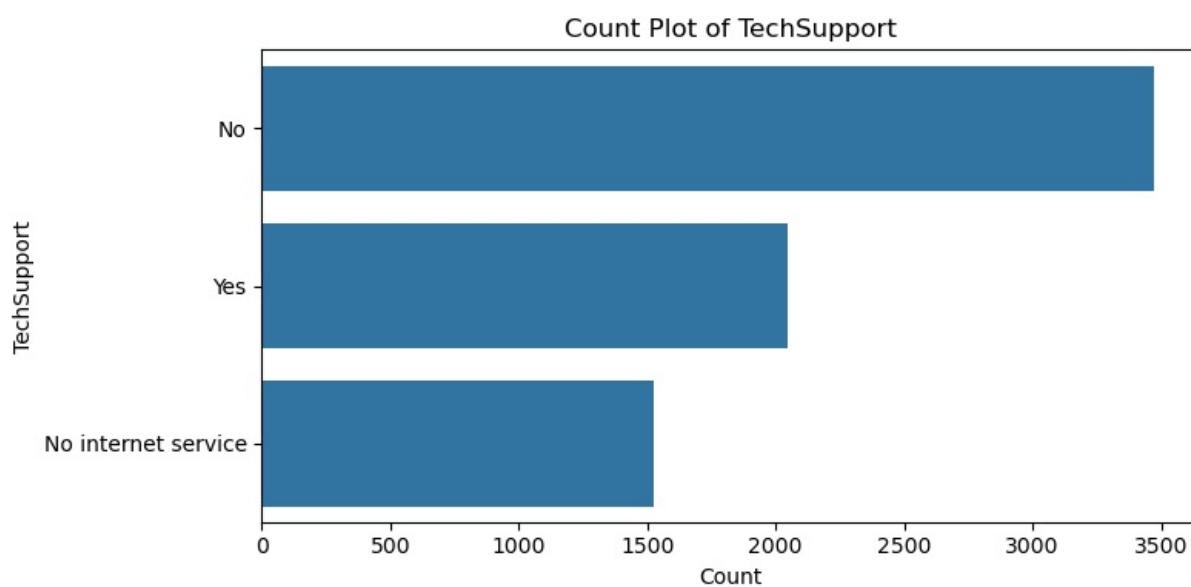
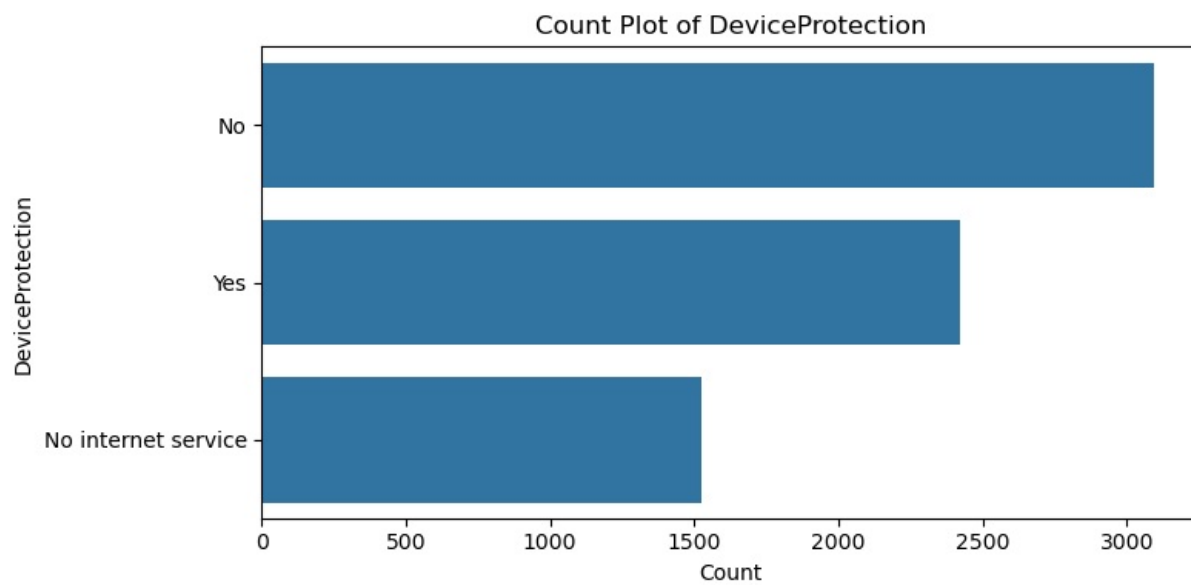
Countplot for categorical columns

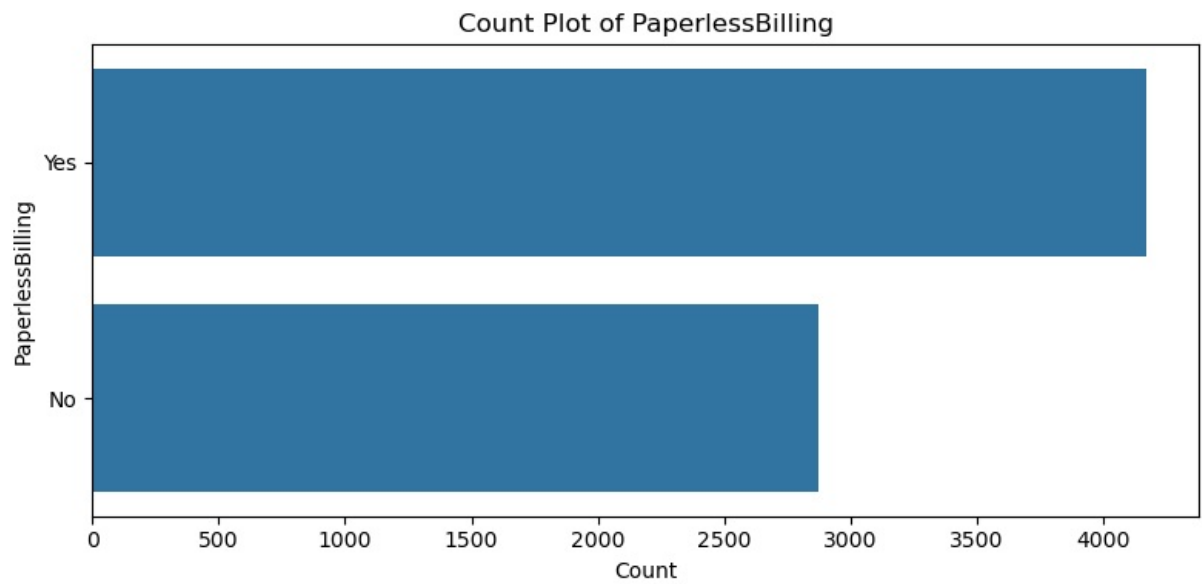
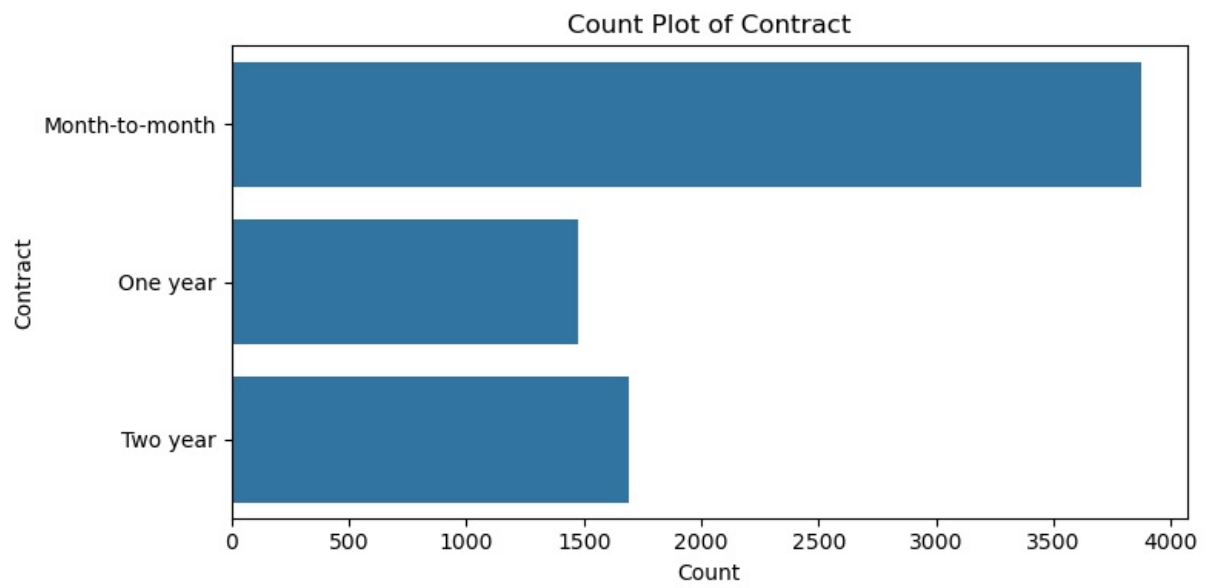
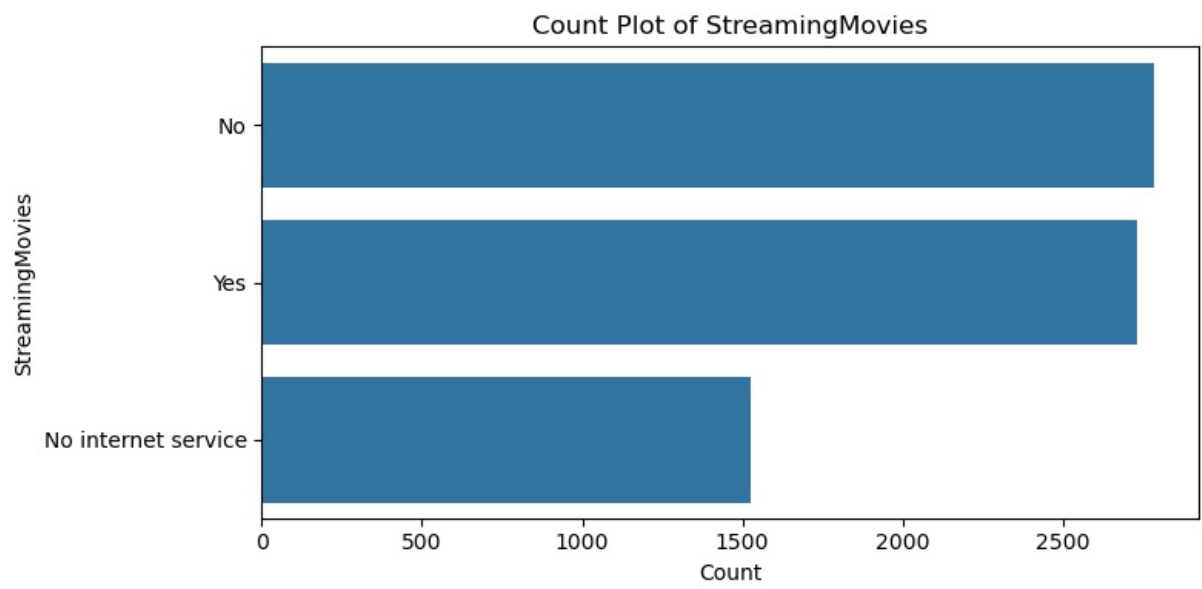
```
In [151...] for col in object_cols:
    plt.figure(figsize=(8, 4))
    sns.countplot(y=df[col])
    plt.title(f"Count Plot of {col}")
    plt.xlabel("Count")
    plt.ylabel(col)
    plt.tight_layout()
    plt.show()
```

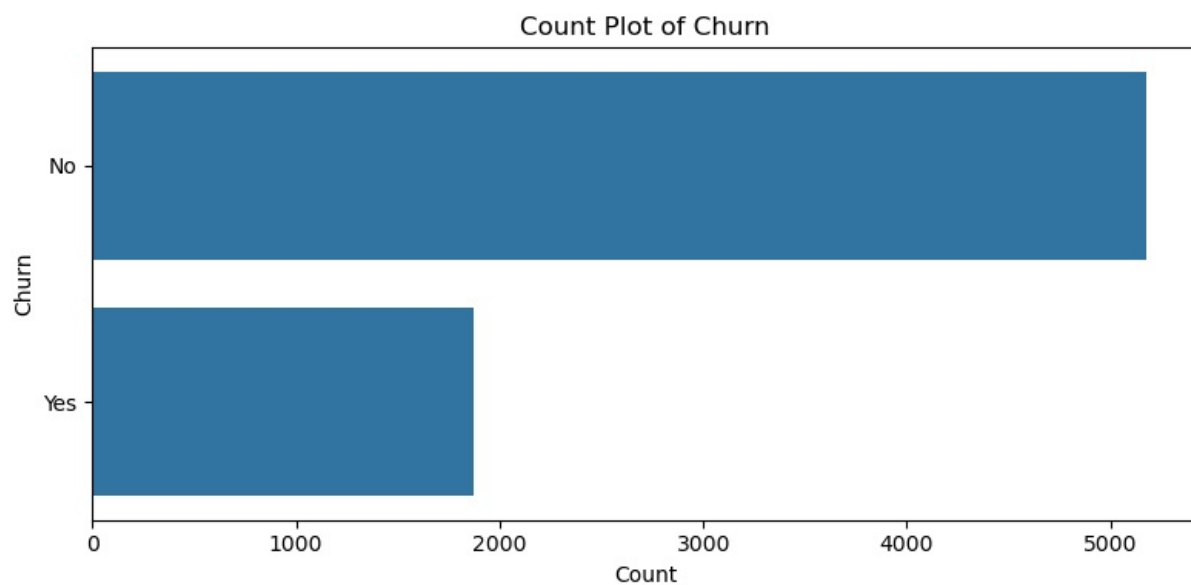
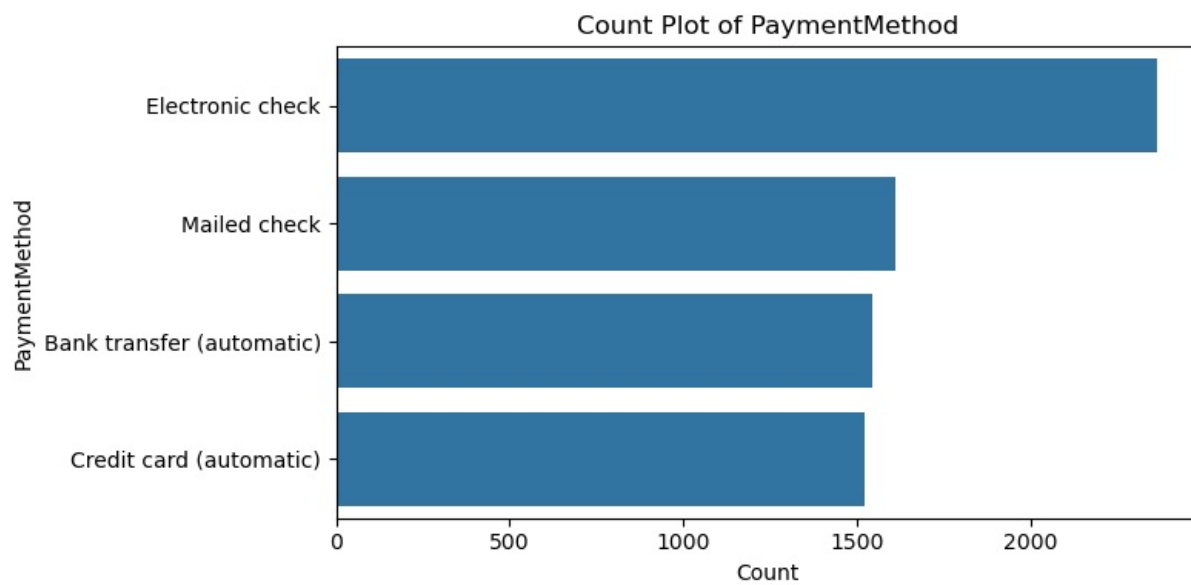












4. Data Preprocessing

In [152...] `df.head(3)`

Out[152...]

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup
0	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes
1	Male	0	No	No	34	Yes	No	DSL	Yes	No
2	Male	0	No	No	2	Yes	No	DSL	Yes	Yes

Label encoding of target column

In [153...] `df["Churn"] = df["Churn"].replace({"Yes": 1, "No": 0})`

C:\Users\asus\AppData\Local\Temp\ipykernel_27480\2364848822.py:1: 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})
```

In [154...] `df.head()`

Out[154...

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

In [155...

```
print(df["Churn"].value_counts())
```

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

Label encoding of categorical fetaures

In [156...

```
# identifying columns with object data type
object_columns = df.select_dtypes(include="object").columns
```

In [157...

```
print(object_columns)
```

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

In [158...

```
# initialize a dictionary to save the encoders
encoders = {}

# apply label encoding and store the encoders
for column in object_columns:
    label_encoder = LabelEncoder()
    df[column] = label_encoder.fit_transform(df[column])
    encoders[column] = label_encoder

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

In [159...

```
encoders
```

Out[159...

```
{'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()}
```

In [160...

```
df.head()
```

Out[160...

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

In [161...	df.tail()										
Out[161...		gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBack
	7038	1	0	1	1	24	1	2	0	2	
	7039	0	0	1	1	72	1	2	1	0	
	7040	0	0	1	1	11	0	1	0	2	
	7041	1	1	1	0	4	1	2	1	0	
	7042	1	0	0	0	66	1	0	1	2	

Training and test data split

```
In [162... # splitting the features and target
X = df.drop(columns=["Churn"])
y = df["Churn"]

In [163... # split training and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

In [164... print(y_train.shape)

(5634,)

In [165... print(y_train.value_counts())

Churn
0    4138
1    1496
Name: count, dtype: int64
```

Synthetic Minority Oversampling TEchnique (SMOTE)

```
In [166... smote = SMOTE(random_state=42)

In [167... X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

In [168... print(y_train_smote.shape)

(8276,)

In [169... print(y_train_smote.value_counts())

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

5. Model Training

Training with default hyperparameters

```
In [170... # dictionary of models
models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "XGBoost": XGBClassifier(random_state=42)
}

In [171... # dictionary to store the cross validation results
cv_scores = {}

# perform 5-fold cross validation for each model
for model_name, model in models.items():
    print(f"Training {model_name} with default parameters")
    scores = cross_val_score(model, X_train_smote, y_train_smote, cv=5, scoring="accuracy")
    cv_scores[model_name] = scores
    print(f"{model_name} cross-validation accuracy: {np.mean(scores):.2f}")
    print("-"*70)
```

Training Decision Tree with default parameters
Decision Tree cross-validation accuracy: 0.78

Training Random Forest with default parameters
Random Forest cross-validation accuracy: 0.84

Training XGBoost with default parameters
XGBoost cross-validation accuracy: 0.83

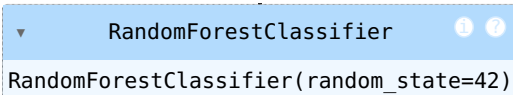
In [172...] cv_scores

Out[172...] {'Decision Tree': array([0.68115942, 0.71903323, 0.81752266, 0.84350453, 0.84350453]),
'Random Forest': array([0.72705314, 0.76676737, 0.90453172, 0.89244713, 0.89848943]),
'XGBoost': array([0.71074879, 0.75226586, 0.90271903, 0.89123867, 0.89909366])}

Random Forest gives the highest accuracy compared to other models with default parameters

In [173...] rfc = RandomForestClassifier(random_state=42)

In [174...] rfc.fit(X_train_smote, y_train_smote)

Out[174...] 
RandomForestClassifier(random_state=42)

In [175...] print(y_test.value_counts())

Churn
0 1036
1 373
Name: churn, dtype: int64

6. Model Evaluation

In [176...] *# evaluate on test data*
y_test_pred = rfc.predict(X_test)

print("Accuracy Score:\n", accuracy_score(y_test, y_test_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred))
print("Classification Report:\n", classification_report(y_test, y_test_pred))

Accuracy Score:
0.7771469127040455
Confusion Matrix:
[[880 156]
 [158 215]]
Classification Report:

	precision	recall	f1-score	support
0	0.85	0.85	0.85	1036
1	0.58	0.58	0.58	373
accuracy			0.78	1409
macro avg	0.71	0.71	0.71	1409
weighted avg	0.78	0.78	0.78	1409

In [177...] *# save the trained model as a pickle file*
model_data = {"model": rfc, "features_names": X.columns.tolist()}

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

7. Load the saved model and build a Predictive System

In [178...] *# load the saved model and the feature names*

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

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

In [179...] print(loaded_model)

RandomForestClassifier(random_state=42)

```

In [180.. print(feature_names)

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

In [181.. 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_data_df = pd.DataFrame([input_data])

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

# encode categorical features using the saved encoders
for column, encoder in encoders.items():
    input_data_df[column] = encoder.transform(input_data_df[column])

# make a prediction
prediction = loaded_model.predict(input_data_df)
pred_prob = loaded_model.predict_proba(input_data_df)

print(prediction)

# results
print(f"Prediction: {'Churn' if prediction[0] == 1 else 'No Churn'}")
print(f"Predicton Probability: {pred_prob}")

[0]
Prediction: No Churn
Predicton Probability: [[0.83 0.17]]

```

Conclusion

This project demonstrates an end-to-end machine learning workflow for predicting customer churn.

Through effective data analysis, preprocessing, and model building, meaningful churn patterns were identified.

The final model provides actionable insights that can help businesses improve customer retention.

Overall, this project reflects practical application of machine learning in a real-world business scenario.

In []:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



THE END

THANK YOU!

We appreciate your time and attention!

