1. Importing the dependencies

```
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_scor
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classi
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
```

2. Data Loading and Understanding

```
#load the csv data to a pandas dataframe
df = pd.read_csv("/content/WA_Fn-UseC_-Telco-Customer-Churn.csv.xls"
```

```
df.shape
```

```
(7043, 21)
```

```
df.head()      #may not show some column... some columns might get t
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	Pho
0	7590-VHVEG	Female	0	Yes	No	1	
1	5575-GNVDE	Male	0	No	No	34	
2	3668-QPYBK	Male	0	No	No	2	
3	7795-CFOCW	Male	0	No	No	45	
4	9237-HQITU	Female	0	No	No	2	

```
pd.set_option("display.max_columns", None)     #will display every
```

```
df.head(2)       #total charges = monthly charges * tenure(in month
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	Pho
0	7590-VHVEG	Female	0	Yes	No	1	
1	5575-GNVDE	Male	0	No	No	34	

```
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
```

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

```
df.head(2)        #customerID is removed
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	M
0	Female	0	Yes	No	1	No	N
1	Male	0	No	No	34	Yes	

Next steps: Generate code with df | New interactive sheet

```
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')
```

```
print(df["gender"].unique())      #categorical
```

```
['Female' 'Male']
```

```
print(df["SeniorCitizen"].unique())     #categorical
```

```
[0 1]
```

```
#printing the unique values in all the columns / printing only the

numerical_features_list = ["tenure", "MonthlyCharges", "TotalCharge

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 (aut
 'Credit card (automatic)']
--------------------------------------------------
Churn ['No' 'Yes']
--------------------------------------------------
```

```
print(df.isnull().sum())           #will count the number of missin
```

```
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[df["TotalCharges"]==" "]
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneServic
488	Female	0	Yes	Yes	0	N
753	Male	0	No	Yes	0	Ye
936	Female	0	Yes	Yes	0	Ye
1082	Male	0	Yes	Yes	0	Ye
1340	Female	0	Yes	Yes	0	N
3331	Male	0	Yes	Yes	0	Ye
3826	Male	0	Yes	Yes	0	Ye
4380	Female	0	Yes	Yes	0	Ye
5218	Male	0	Yes	Yes	0	Ye
6670	Female	0	Yes	Yes	0	Ye
6754	Male	0	No	Yes	0	Ye

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

```
11
```

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

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

```
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
```

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

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

Insights:

1. Customer ID removed as it is not required for modelling
2. No missing values in the dataset
3. Missing values in the TotalCharges column were repleced with 0
4. Class imbalance identified in the target

3. Exploratory Data Analysis (EDA)

```
df.shape
```

```
(7043, 20)
```

```
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.head(2)
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	M
0	Female	0	Yes	No	1	No	N
1	Male	0	No	No	34	Yes	

Next steps: Generate code with df | New interactive sheet

```
df.describe()
```

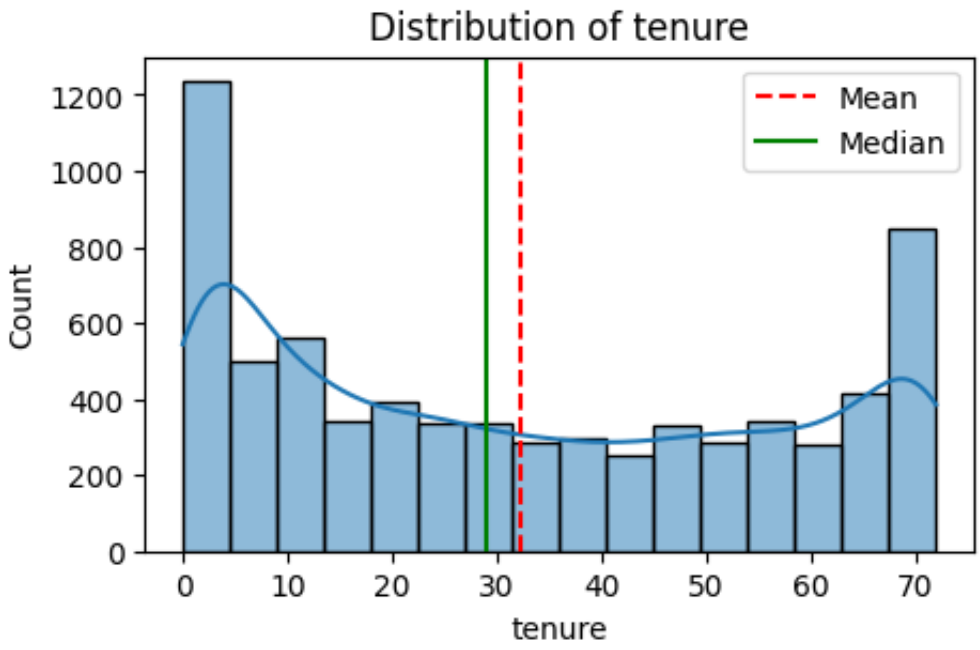
	SeniorCitizen	tenure	MonthlyCharges	TotalCharges
count	7043.000000	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692	2279.734304
std	0.368612	24.559481	30.090047	2266.794470
min	0.000000	0.000000	18.250000	0.000000
25%	0.000000	9.000000	35.500000	398.550000
50%	0.000000	29.000000	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

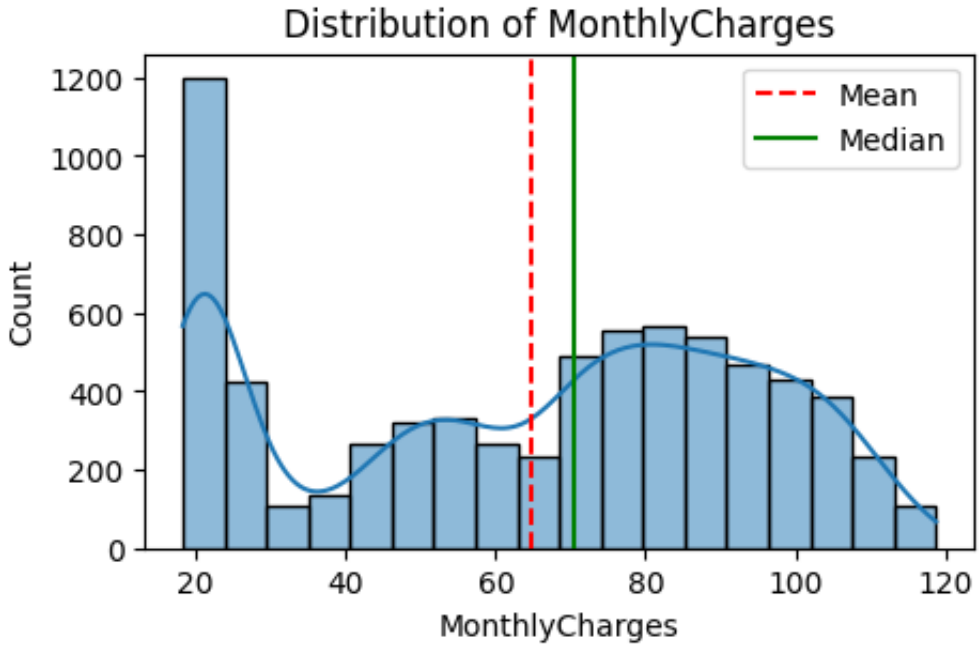
Understand the distribution of the numerical features

```
from seaborn.palettes import color_palette
def plot_histogram(df, column_name):

  plt.figure(figsize=(5,3))
  sns.histplot(data=df, x=column_name, kde=True)
  plt.title(f"Distribution of {column_name}")

  # calculate the mean and median values for the columns
  col_mean = df[column_name].mean()
  col_median = df[column_name].median()

  # add vertical lines for mean and median
  plt.axvline(col_mean, color="red", linestyle="--", label="Mean")
  plt.axvline(col_median, color="green", linestyle="-", label="Medi

  plt.legend()

  plt.show()
```

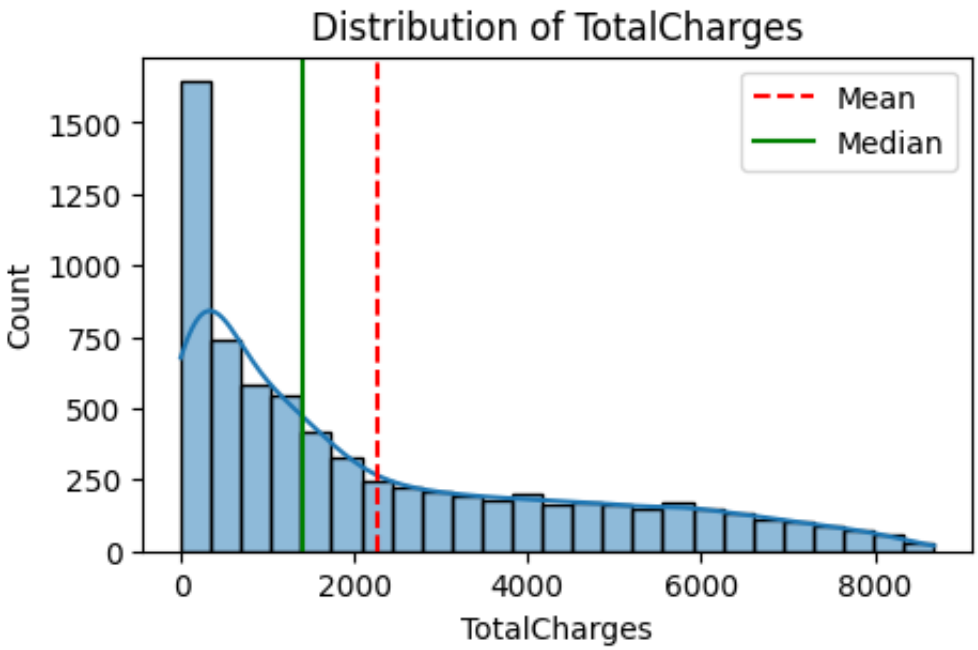
```
plot_histogram(df, "tenure")
```




```
plot_histogram(df, "MonthlyCharges")
```



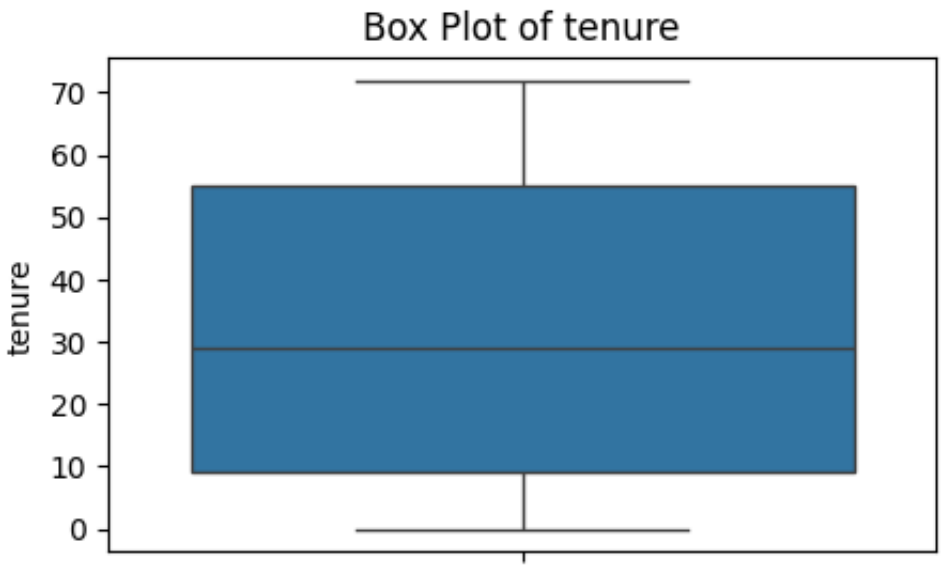
```
plot_histogram(df, "TotalCharges")
```



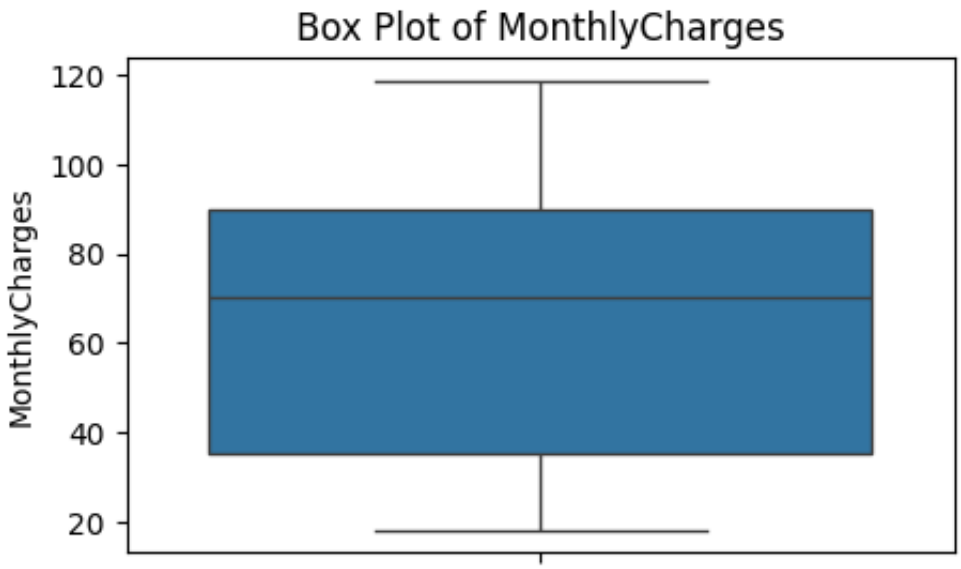
Box plot for numerical features

```
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
```

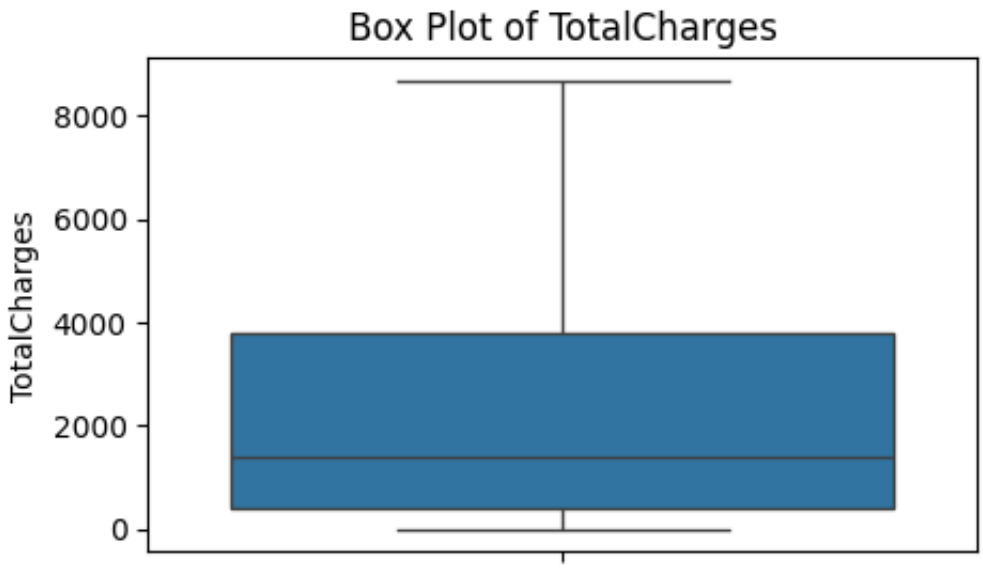
```
plot_boxplot(df, "tenure")
```




```
plot_boxplot(df, "MonthlyCharges")
```

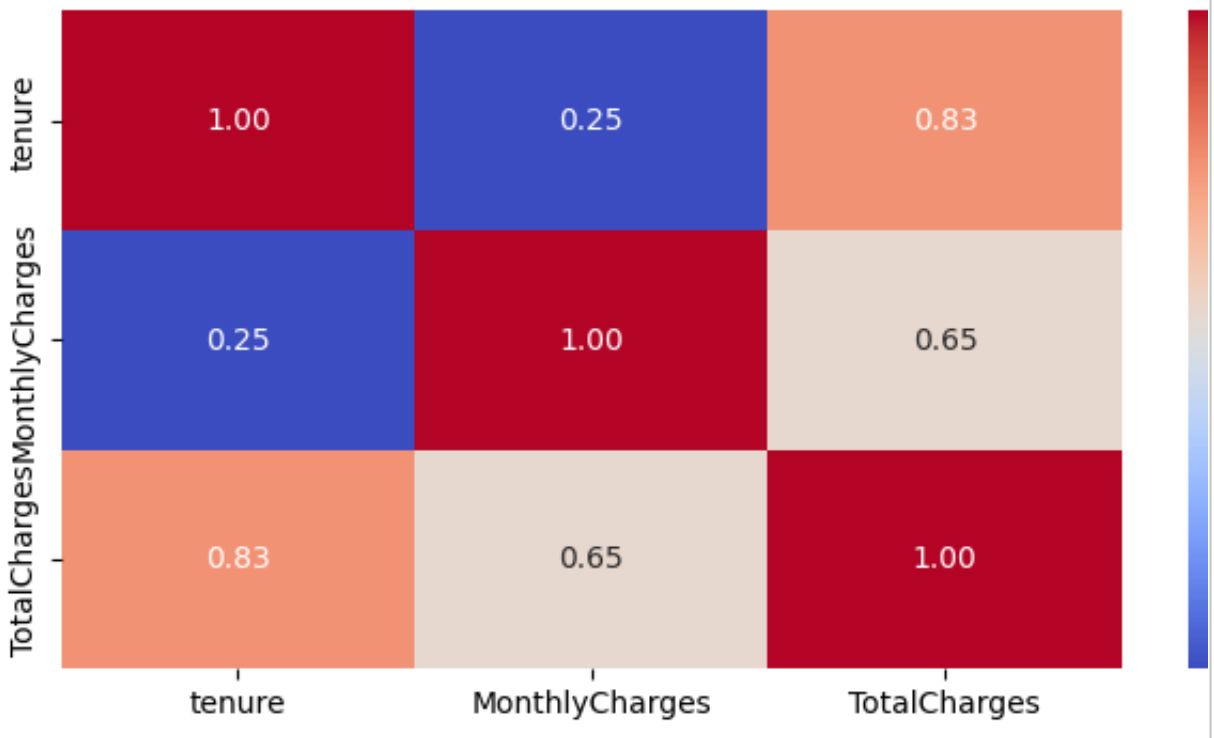


```
plot_boxplot(df, "TotalCharges")
```



Correlation Heatmap for numerical columns

```
# correlation matrix - heatmap
plt.figure(figsize=(8, 4))
sns.heatmap(df[["tenure", "MonthlyCharges", "TotalCharges"]].corr()
plt.show()
```



Categorical features - Analysis

```
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')
```

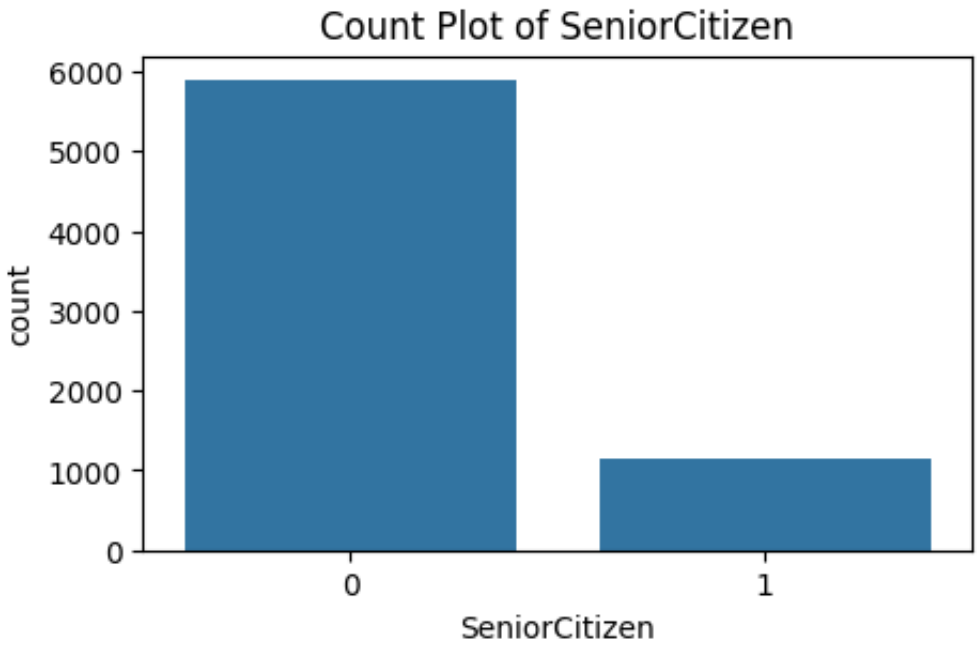
```
df.info()
```

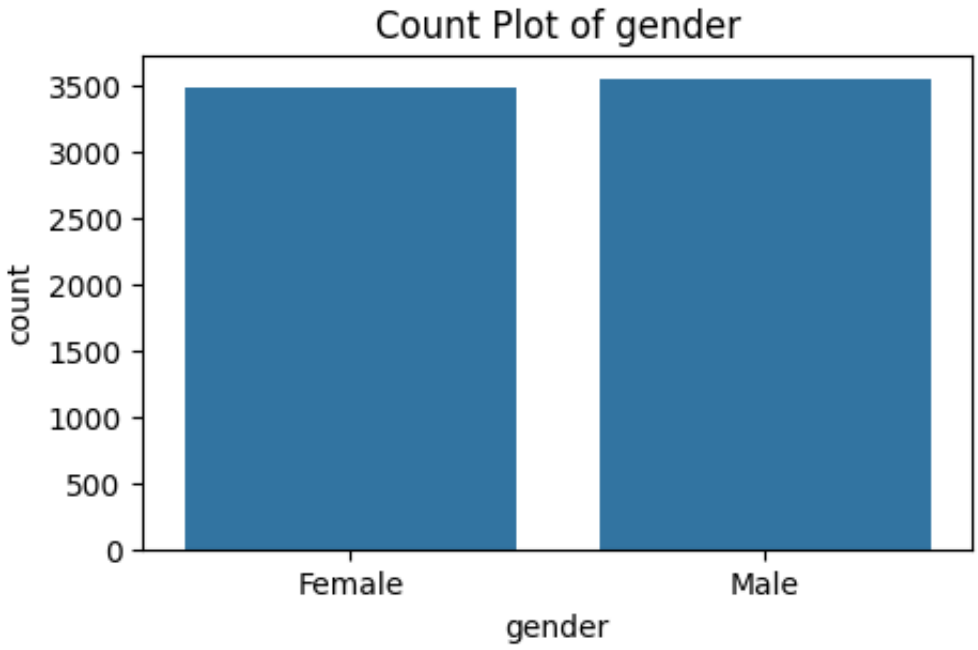
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   gender            7043 non-null   object
 1   SeniorCitizen     7043 non-null   int64
 2   Partner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

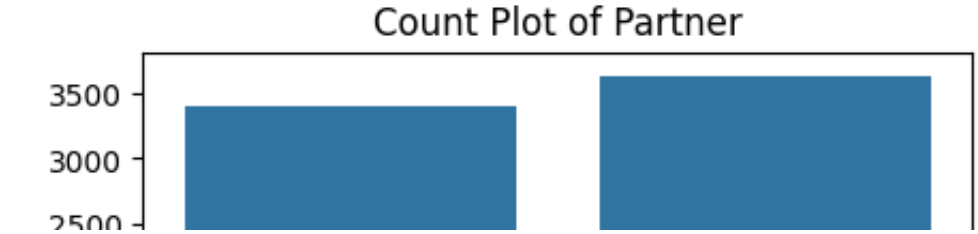
Countplot for categorical columns

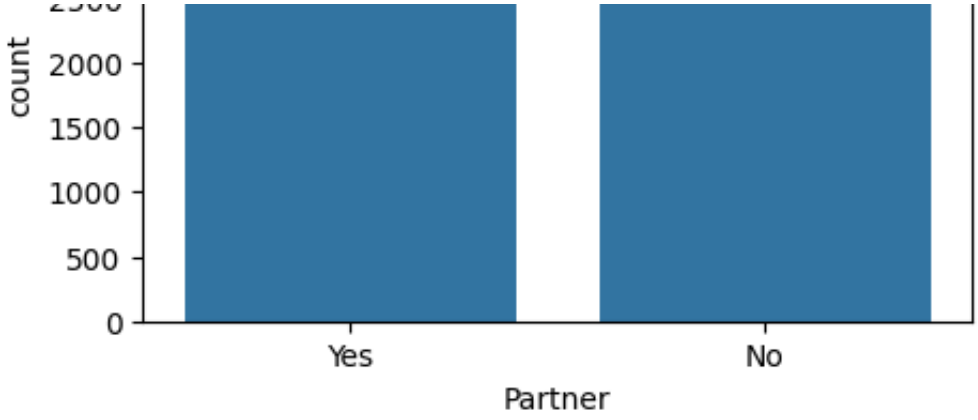
```
object_cols = df.select_dtypes(include="object").columns.to_list()
```

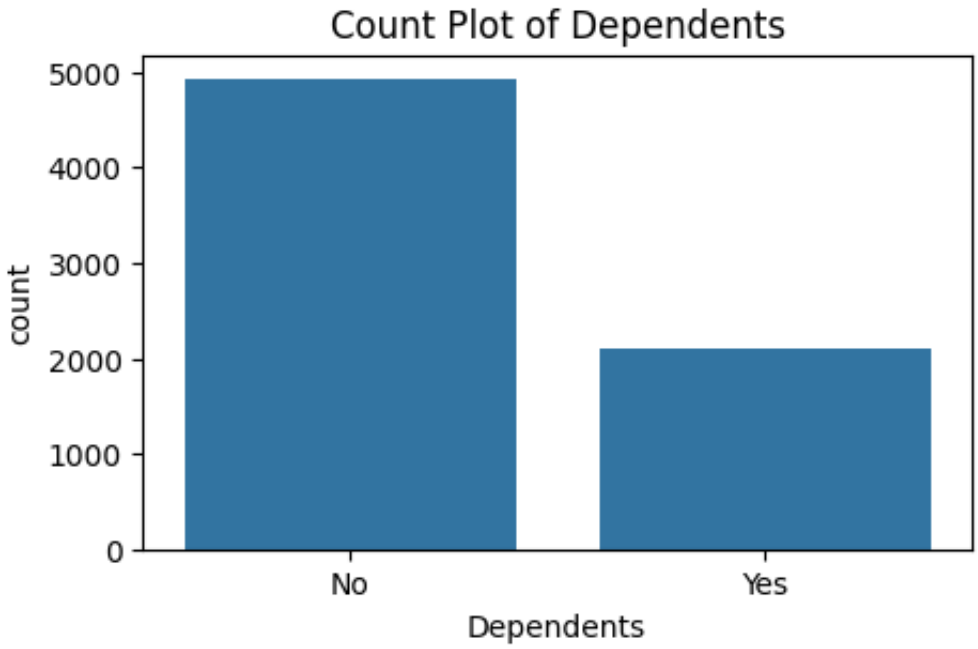
```
object_cols = ["SeniorCitizen"] + object_cols

for col in object_cols:
  plt.figure(figsize=(5,3))
  sns.countplot(x=df[col])
  plt.title(f"Count Plot of {col}")
  plt.show()
```

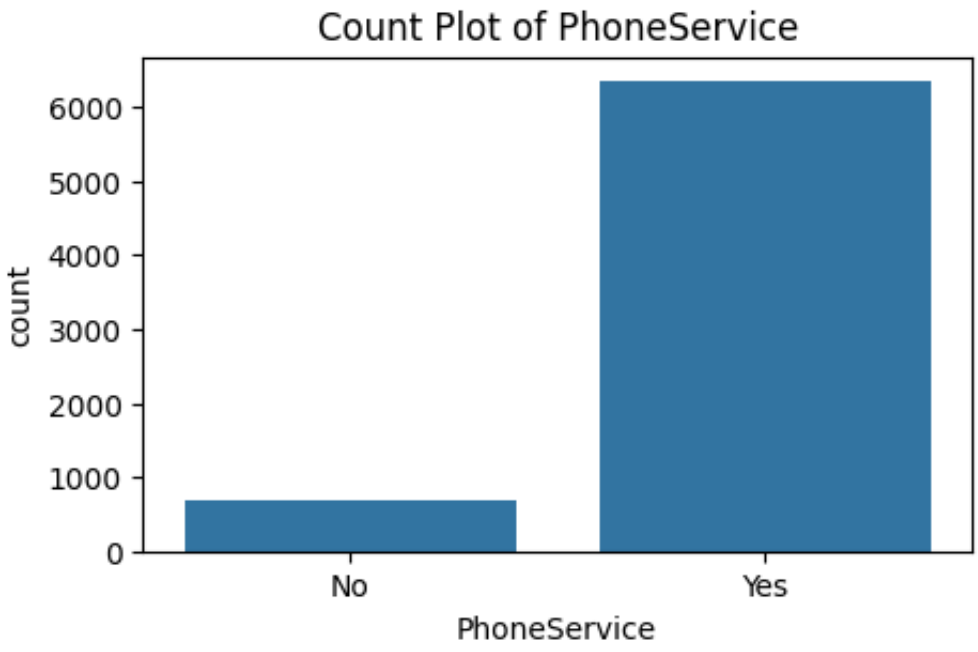


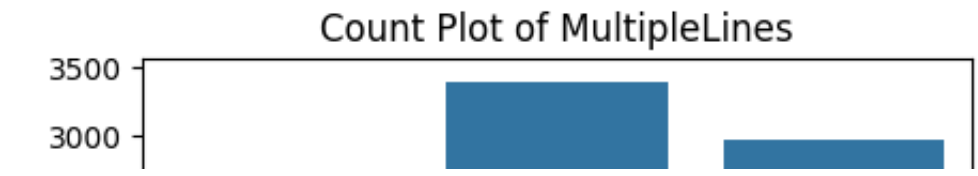


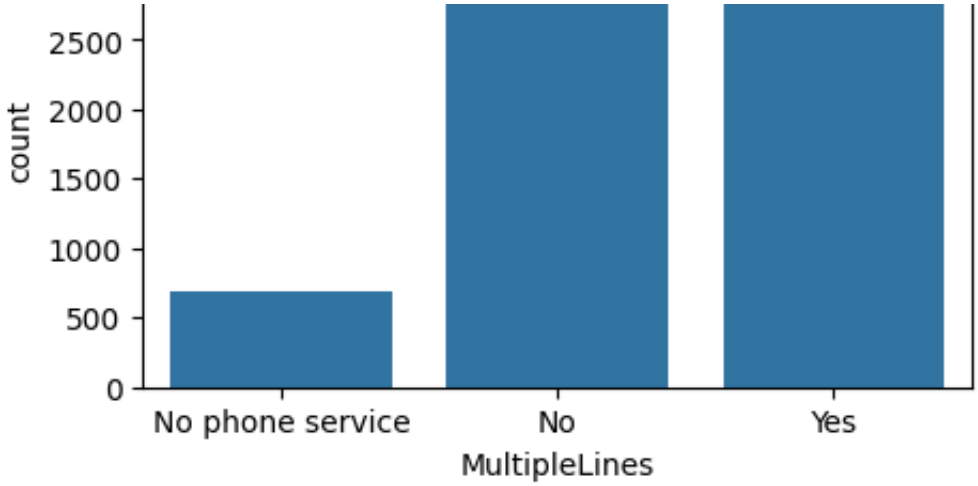


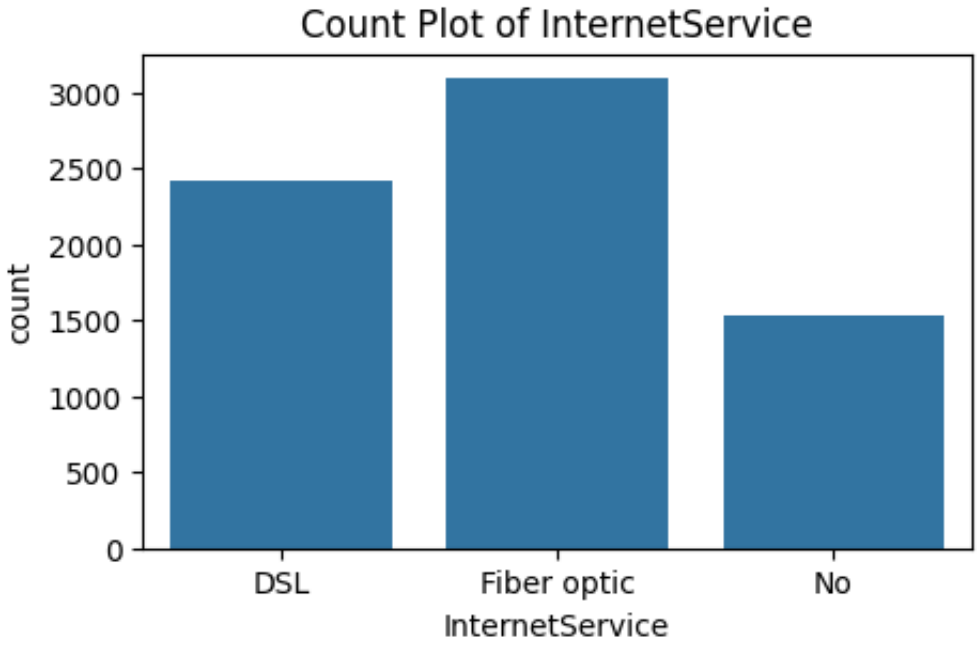

count
2000
1500
1000
500
0
Yes
No
Partner

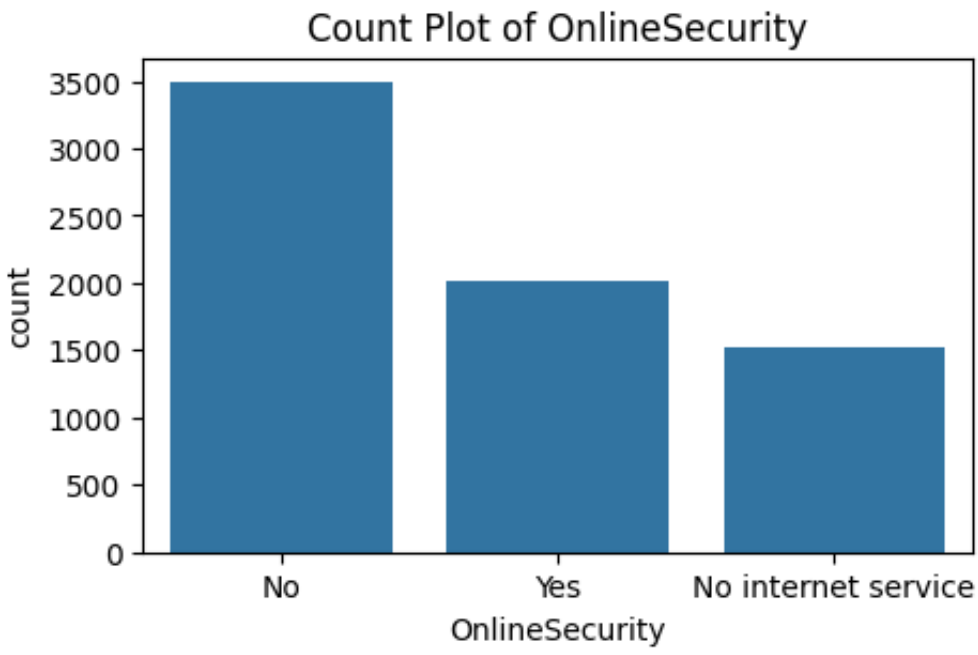

Count Plot of Dependents
count
5000
4000
3000
2000
1000
0
No
Yes
Dependents


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

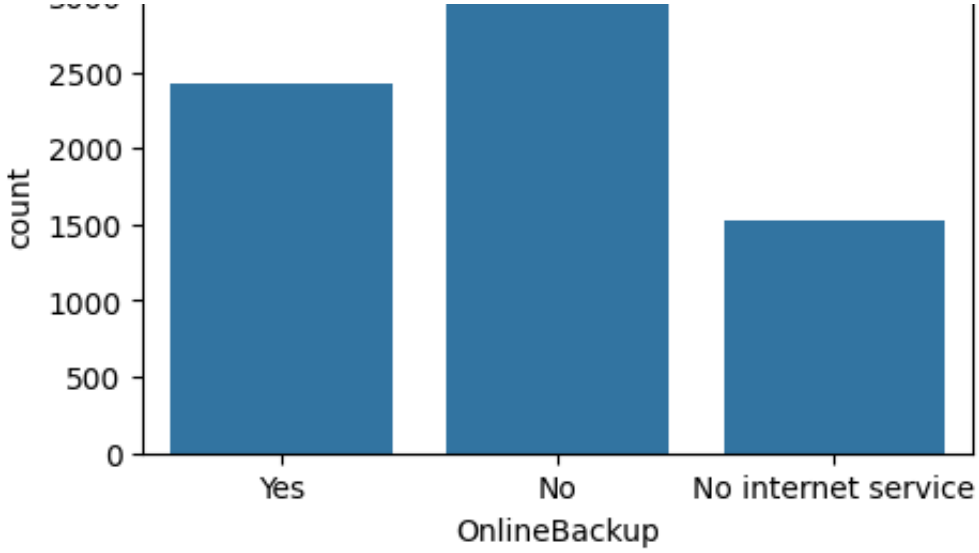

Count Plot of MultipleLines
3500
3000

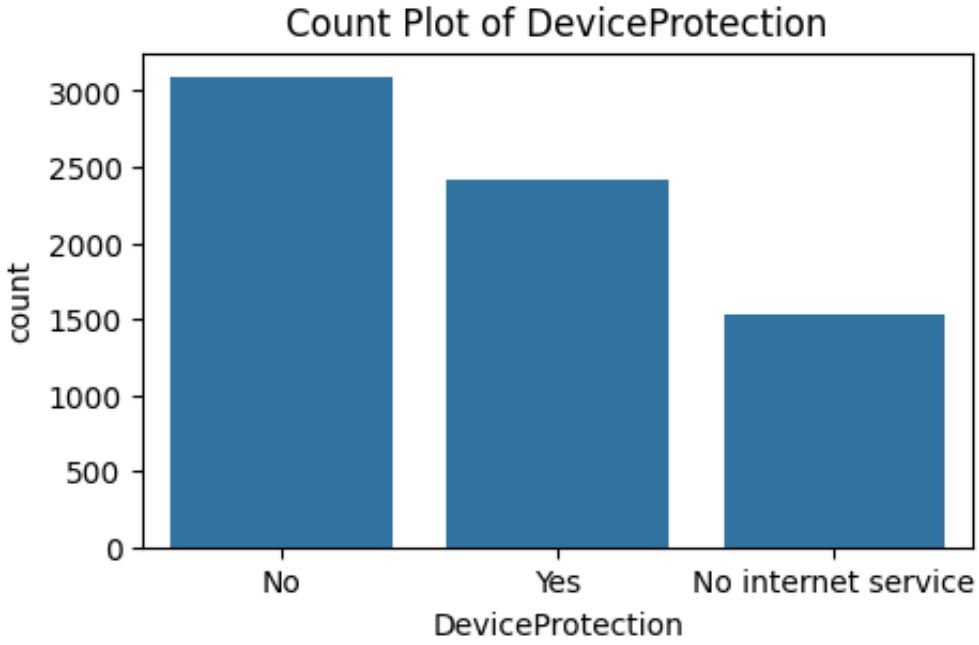

2500
2000
1500
1000
500
0
count
No phone service
No
Yes
MultipleLines

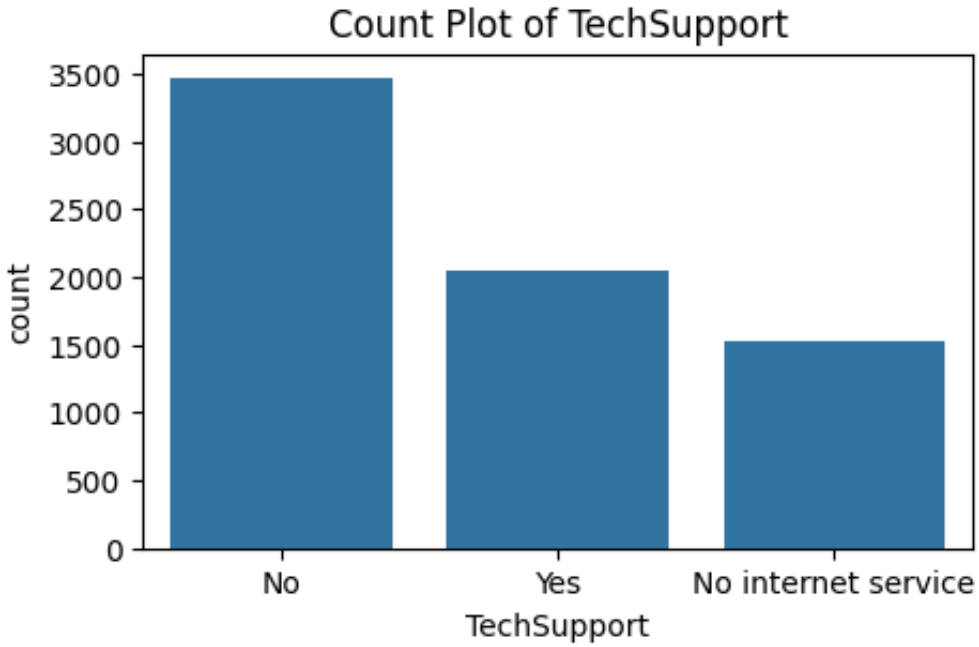

Count Plot of InternetService
3000
2500
2000
1500
1000
500
0
count
DSL
Fiber optic
No
InternetService


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

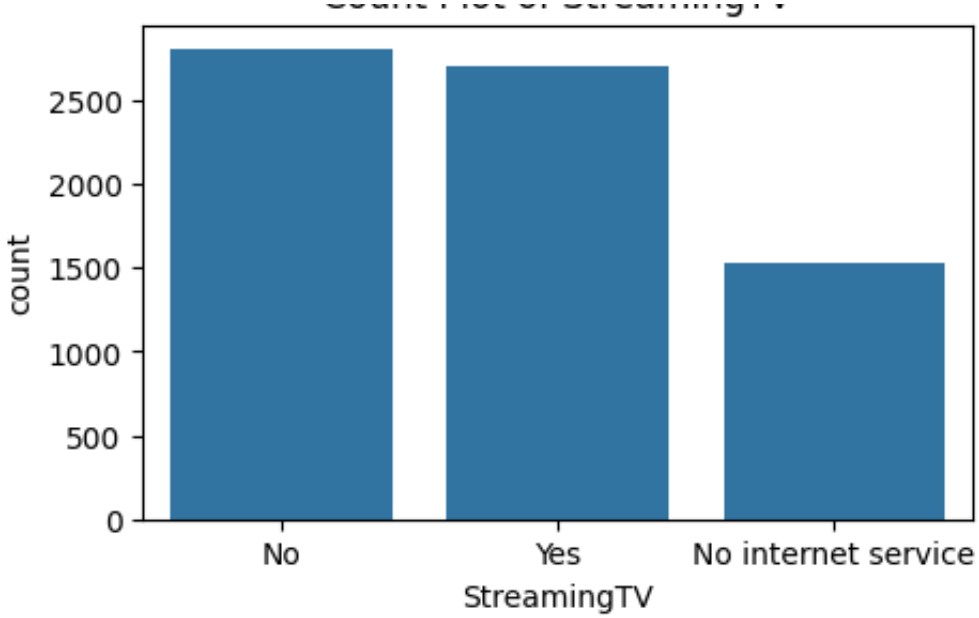

Count Plot of OnlineBackup
3000

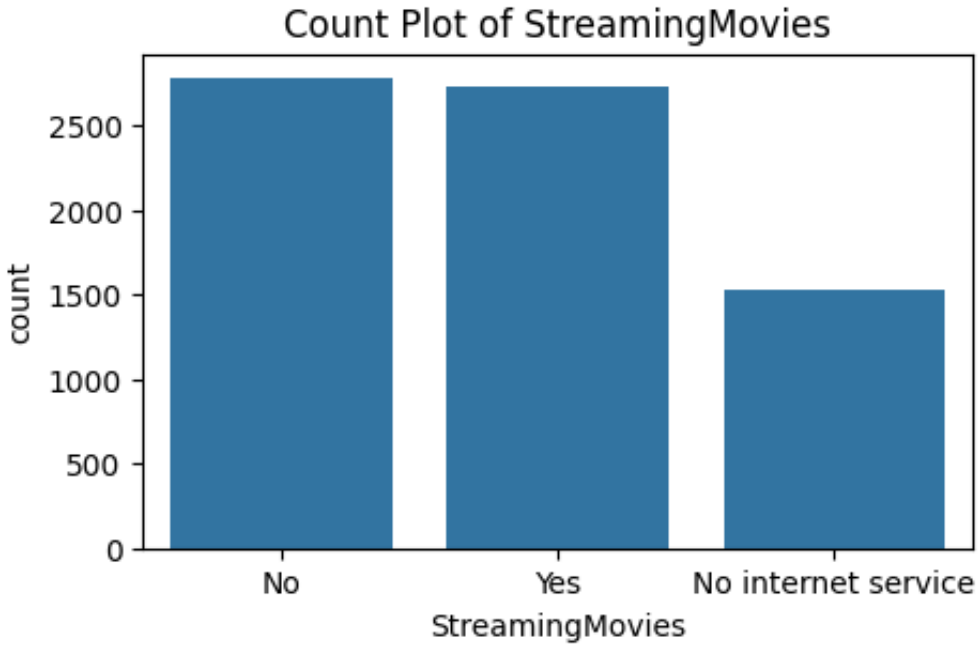


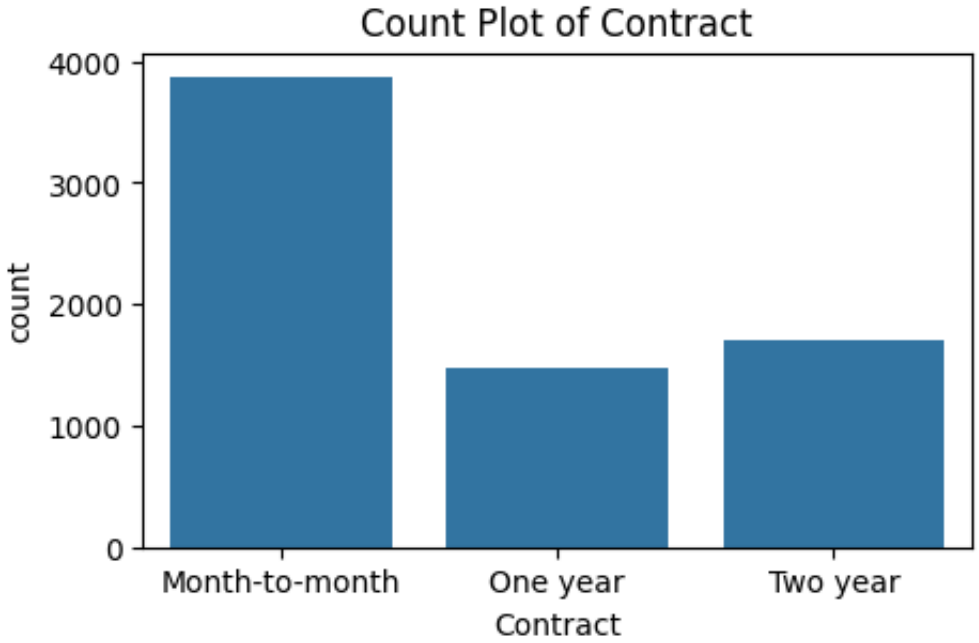


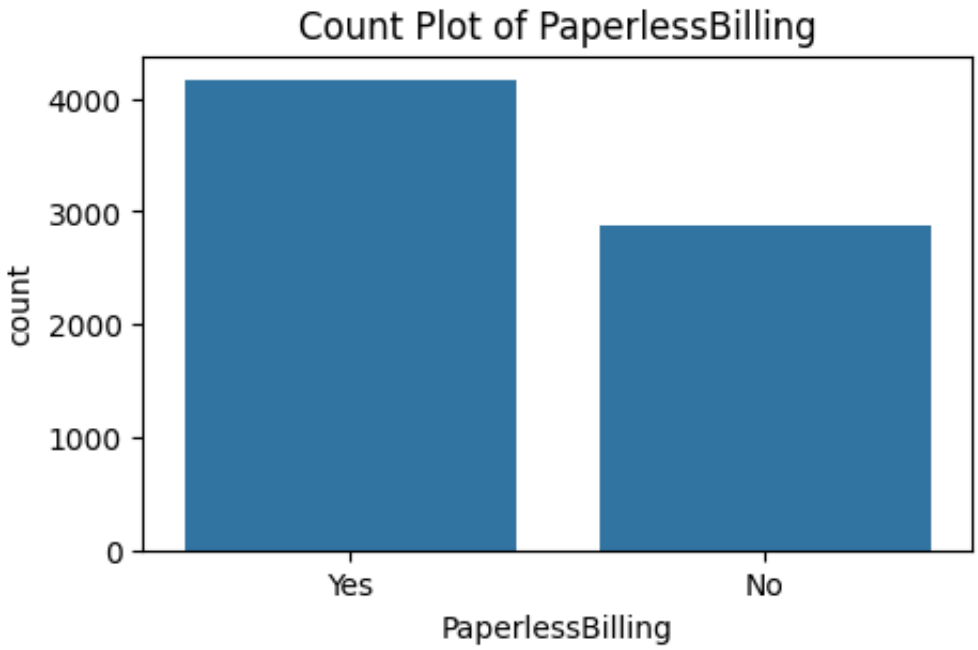


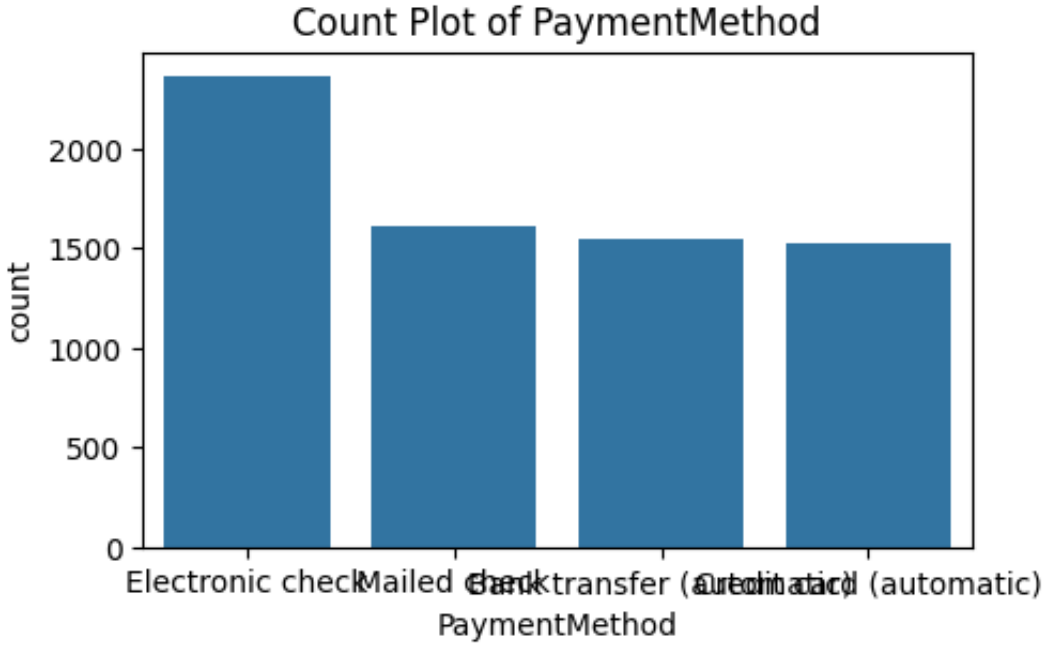
Count Plot of StreamingTV

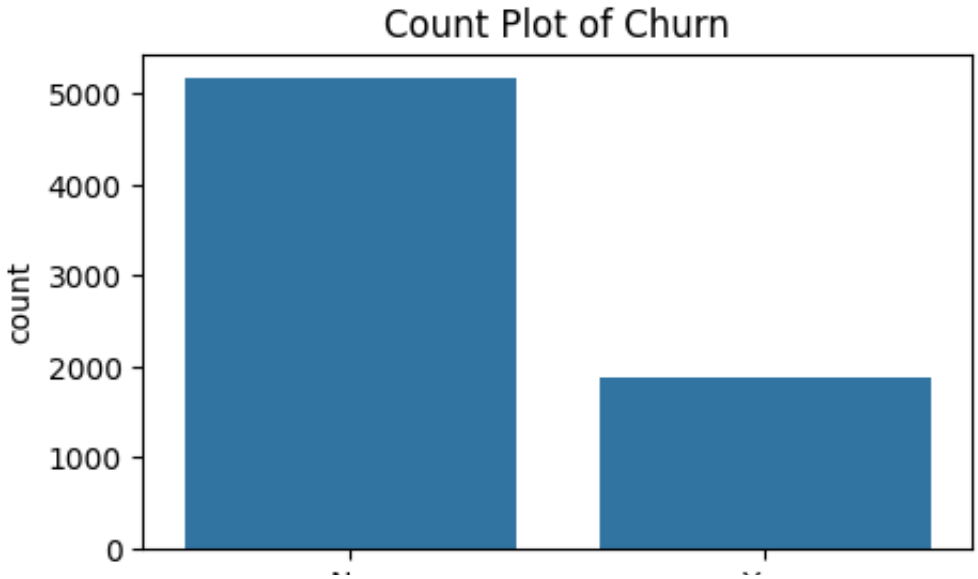

2500
2000
1500
1000
500
0
count
No
Yes
No internet service
StreamingTV

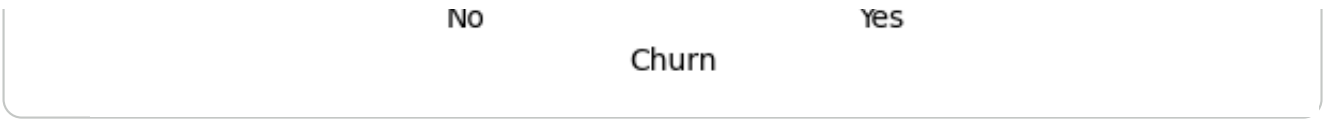

Count Plot of StreamingMovies
2500
2000
1500
1000
500
0
count
No
Yes
No internet service
StreamingMovies


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


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


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


Count Plot of Churn
count
5000
4000
3000
2000
1000
0



4. Data Preprocessing

```
df.head()
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	M
0	Female	0	Yes	No	1	No	N
1	Male	0	No	No	34	Yes	
2	Male	0	No	No	2	Yes	
3	Male	0	No	No	45	No	N
4	Female	0	No	No	2	Yes	

Next steps: Generate code with df | New interactive sheet

Label encoding of target column

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

```
/tmp/ipython-input-2364848822.py:1: FutureWarning: Downcasting behav
  df["Churn"] = df["Churn"].replace({"Yes": 1, "No": 0})
```

```
df.head(3)
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	M
0	Female	0	Yes	No	1	No	N
1	Male	0	No	No	34	Yes	
2	Male	0	No	No	2	Yes	

Next steps: Generate code with df New interactive sheet

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

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

Label encoding of categorical features

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

```
print(object_columns)
```

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

```
# 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)
```

```
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()}
```

```
df.head()
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	M
0	0	0	1	0	1	0	
1	1	0	0	0	34	1	
2	1	0	0	0	2	1	
3	1	0	0	0	45	0	
4	0	0	0	0	2	1	

Next steps: Generate code with df | New interactive sheet

Training and test data split

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

```
# split training and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
```

```
print(y_train.shape)
```

```
(5634,)
```

```
print(y_train.value_counts())
```

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

Synthetic Minority Oversampling TEchnique (SMOTE)

```
smote = SMOTE(random_state=42)
```

```
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

```
print(y_train_smote.shape)
```

```
(8276,)
```

```
print(y_train_smote.value_counts())
```

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

5. Model Training

Training with default hyperparameters

```
# dictionary of models
models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "XGBoost": XGBClassifier(random_state=42, use_label_encoder=Fal
    "Logistic Regression": LogisticRegression(random_state=42, max_
    "SVC": SVC(random_state=42, probability=True)   # probability=T
}
```

```
# dictionary to store the cross validation results
cv_scores = {}

print(" Performing 5-Fold Cross Validation for All Models \n")

# 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, c
    cv_scores[model_name] = scores
    print(f"{model_name} cross-validation accuracy: {np.mean(scores
    print("-" * 70)
```

```
 Performing 5-Fold Cross Validation for All Models

Training Decision Tree with default parameters...
Decision Tree cross-validation accuracy: 0.7778
----------------------------------------------------------------------
Training Random Forest with default parameters...
Random Forest cross-validation accuracy: 0.8408
----------------------------------------------------------------------
Training XGBoost with default parameters...
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: Use
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: Use
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: Use
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: Use
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: Use
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
XGBoost cross-validation accuracy: 0.8310
----------------------------------------------------------------------
Training Logistic Regression with default parameters...
Logistic Regression cross-validation accuracy: 0.7927
----------------------------------------------------------------------
Training SVC with default parameters...
SVC cross-validation accuracy: 0.6396
----------------------------------------------------------------------
```

```
cv_scores
```

```
{'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]),
 'XGBoost': array([0.70048309, 0.75649547, 0.90271903, 0.89486405,
0.90030211]),
 'Logistic Regression': array([0.73188406, 0.74803625, 0.8265861 ,
0.81993958, 0.83685801]),
 'SVC': array([0.65519324, 0.65740181, 0.61510574, 0.61993958,
0.65015106])}
```

6. Model Evaluation

```
from sklearn.metrics import accuracy_score, confusion_matrix, classi

evaluation_results = {}

print("\n Evaluating Models on Test Data \n")

# Loop through all 5 models and evaluate each
for model_name, model in models.items():
    print(f"Evaluating {model_name}...")
    model.fit(X_train_smote, y_train_smote)   # train each model
    y_pred = model.predict(X_test)            # test predictions

    acc = accuracy_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)
    cr = classification_report(y_test, y_pred, output_dict=True)

    evaluation_results[model_name] = {
        "Accuracy": acc,
        "Precision": cr['1']['precision'],
        "Recall": cr['1']['recall'],
        "F1-Score": cr['1']['f1-score']
    }

    print(f"Accuracy: {acc:.4f}")
    print("Confusion Matrix:\n", cm)
    print("Classification Report:\n", classification_report(y_test,
    print("-" * 70)

# Create a summary DataFrame to compare model performance
eval_df = pd.DataFrame(evaluation_results).T
print("\n Model Performance Comparison:")
display(eval_df.sort_values(by="Accuracy", ascending=False))
```

```
 Evaluating Models on Test Data

Evaluating Decision Tree...
Accuracy: 0.7317
Confusion Matrix:
 [[824 212]
 [166 207]]
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.80      0.81      1036
           1       0.49      0.55      0.52       373

    accuracy                           0.73      1409
   macro avg       0.66      0.68      0.67      1409
weighted avg       0.74      0.73      0.74      1409

----------------------------------------------------------------------
Evaluating Random Forest...
Accuracy: 0.7786
Confusion Matrix:
 [[878 158]
 [154 219]]
Classification Report:
              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

----------------------------------------------------------------------
Evaluating XGBoost...
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: Use
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
Accuracy: 0.7807
Confusion Matrix:
 [[885 151]
 [158 215]]
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.85      0.85      1036
           1       0.59      0.58      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

--------------------------------------------------------------------
Evaluating Logistic Regression...
Accuracy: 0.7644
Confusion Matrix:
 [[784 252]
 [ 80 293]]
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.76      0.83      1036
           1       0.54      0.79      0.64       373

    accuracy                           0.76      1409
   macro avg       0.72      0.77      0.73      1409
weighted avg       0.81      0.76      0.78      1409

--------------------------------------------------------------------
Evaluating SVC...
Accuracy: 0.6884
Confusion Matrix:
 [[733 303]
 [136 237]]
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.71      0.77      1036
           1       0.44      0.64      0.52       373

    accuracy                           0.69      1409
   macro avg       0.64      0.67      0.64      1409
weighted avg       0.74      0.69      0.70      1409

--------------------------------------------------------------------
```

Model Performance Comparison:

	Accuracy	Precision	Recall	F1-Score
XGBoost	0.780696	0.587432	0.576408	0.581867
Random Forest	0.778566	0.580902	0.587131	0.584000
Logistic Regression	0.764372	0.537615	0.785523	0.638344
Decision Tree	0.731725	0.494033	0.554960	0.522727
SVC	0.688432	0.438889	0.635389	0.519168

7.Selecting and Saving the Best Model

Save the best model

```
best_model_name = eval_df["Accuracy"].idxmax()
best_model = models[best_model_name]

print(f"\n Best Model: {best_model_name} with Accuracy = {eval_df.l

best_model.fit(X_train_smote, y_train_smote)

model_data = {"model": best_model, "feature_names": X.columns.to_li

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

print(" Best model saved as 'customer_churn_best_model.pkl'")
```

```
 Best Model: XGBoost with Accuracy = 0.7807
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: Use
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
 Best model saved as 'customer_churn_best_model.pkl'
```

Load the saved best model

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

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

print(" Loaded model:", loaded_model)
```

```
 Loaded model: XGBClassifier(base_score=None, booster=None, callback
               colsample_bylevel=None, colsample_bynode=None,
               colsample_bytree=None, device=None, early_stopping_rou
               enable_categorical=False, eval_metric='logloss',
               feature_types=None, feature_weights=None, gamma=None,
               grow_policy=None, importance_type=None,
               interaction_constraints=None, learning_rate=None, max_
               max_cat_threshold=None, max_cat_to_onehot=None,
               max_delta_step=None, max_depth=None, max_leaves=None,
               min_child_weight=None, missing=nan, monotone_constrair
               multi_strategy=None, n_estimators=None, n_jobs=None,
               num_parallel_tree=None, ...)
```

Predict Churn for a Sample Customer (Quick Test)... Static Datas

```
import pandas as pd

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
}

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

input_df = pd.DataFrame([input_data])

# Encode categorical features
for col, encoder in encoders.items():
    input_df[col] = encoder.transform(input_df[col])

# Predict
pred = loaded_model.predict(input_df)
prob = loaded_model.predict_proba(input_df)[0][1]

print(" Prediction Result:")
print(f" Prediction: {'Churn' if pred[0]==1 else 'No Churn'}")
print(f" Churn Probability: {prob:.2f}")
```