Customer Churn Prediction

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_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

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

```
df.shape
```

```
(7043, 21)
```

```
df.head()
```

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

```
pd.set_option("display.max_columns", None)
```

```
df.head(2)
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	Onl
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	

```
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)
```

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

```
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())
```

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

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

```
[0 1]
```

```
# 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']
--------------------------------------------------
```

```
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
```

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

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

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

```
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. Empty strings in the TotalCharges column were replaced 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	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	D
0	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes	
1	Male	0	No	No	34	Yes	No	DSL	Yes	No	

```
df.describe()
```

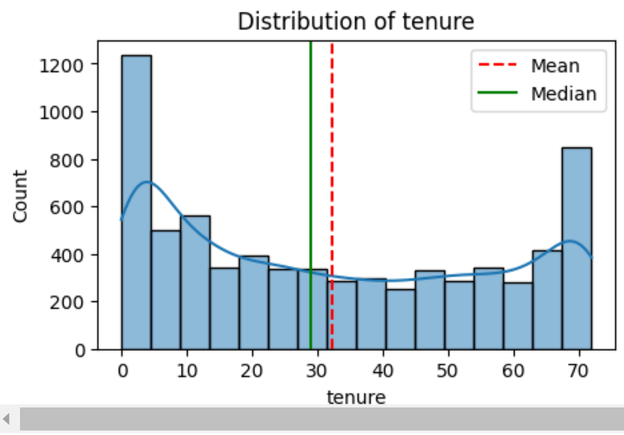
```
       SeniorCitizen       tenure  MonthlyCharges  TotalCharges
count    7043.000000  7043.000000     7043.000000   7043.000000
mean        0.162147    32.371149       64.761692   2279.734304
std         0.368612    24.559481       30.090047   2266.794470
min         0.000000     0.000000       18.250000      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

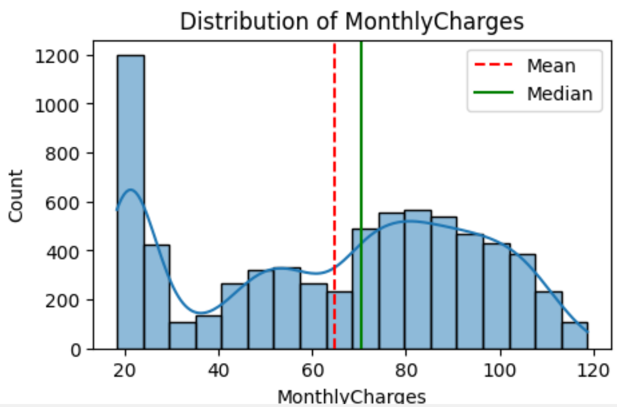
Understand the distribution of the numerical features

```
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 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="Median")

  plt.legend()

  plt.show()
```

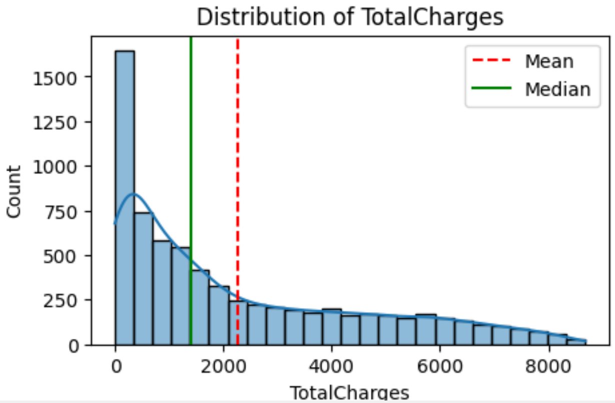
```
plot_histogram(df, "tenure")
```



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



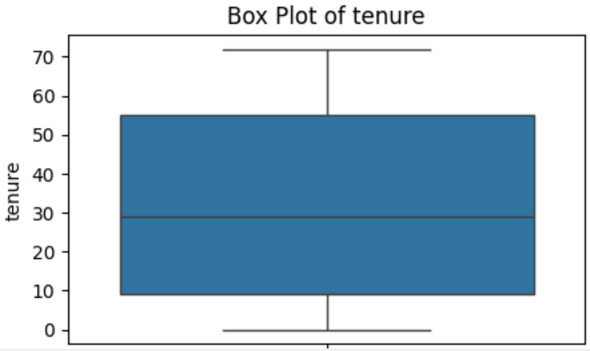
```
plot_histogram(df, "TotalCharges")
```



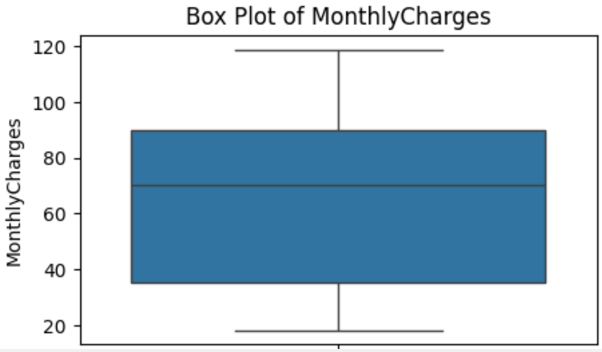
Box plot for numerical features

```
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
```

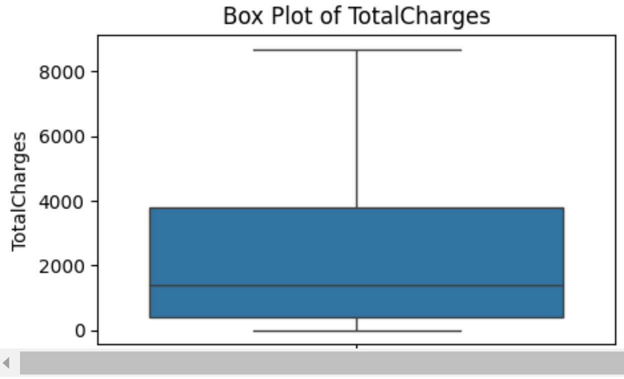
```
plot_boxplot(df, "tenure")
```



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

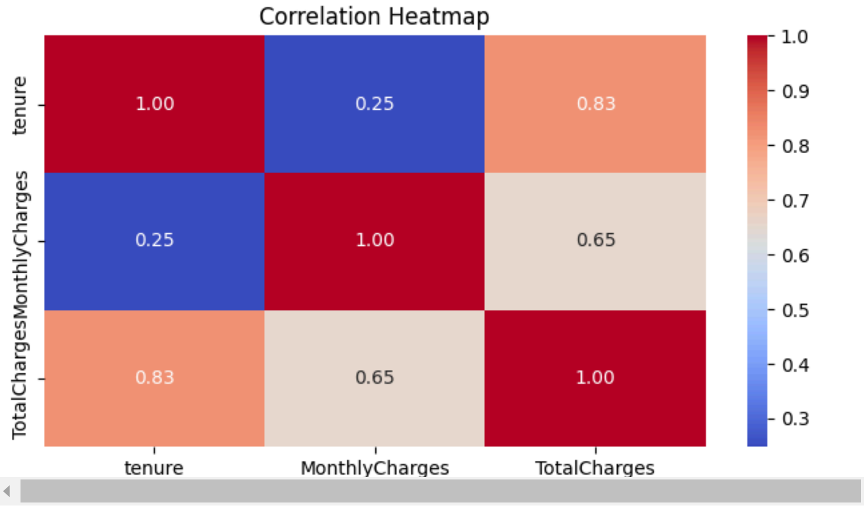


```
plot_boxplot(df, "TotalCharges")
```



Correlation Heatmap for numerical columns

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

Show hidden output

4. Data Preprocessing

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

	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

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

```
<ipython-input-57-b6eb27bc3ee0>:1: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future ve
  df["Churn"] = df["Churn"].replace({"Yes": 1, "No": 0})
```

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

	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

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

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

Label encoding of categorical fetaures

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

```
print(object_columns)
```

```
Index(['gender', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines',
       'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
       '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	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

Traianing 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=0.2, random_state=42)
```

```
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)
}
```

```
# 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
----------------------------------------------------------------------
```

```
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])}
```

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

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

```
rfc.fit(X_train_smote, y_train_smote)
```

```
▾       RandomForestClassifier      i ?

RandomForestClassifier(random_state=42)
```

```
print(y_test.value_counts())
```

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

6. Model Evaluation

```
# evaluate on test data
y_test_pred = rfc.predict(X_test)

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

```
Accuracy Score:
 0.7785663591199432
Confsuion 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
```

```
# 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

```
# load teh 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"]
```

```
print(loaded_model)
```

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

```
print(feature_names)
```

```
['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure', 'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity',
```

```
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 featires using teh 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)
```