Introduction to Churn Prediction Using ML

Churn prediction using machine learning (ML) models is an essential technique for businesses to identify and retain customers who are likely to stop using their products or services. This process involves several key steps, starting with data collection and preparation, where businesses gather comprehensive data on customer behaviour, demographics, transaction history, and other relevant factors. This data forms the foundation for building an effective churn prediction model. Next, exploratory data analysis (EDA) is conducted to uncover patterns and trends within the data that may indicate potential churn. Understanding these patterns helps in identifying the key factors contributing to churn and informs the subsequent steps in the process.

Model selection and training involve choosing appropriate ML models such as logistic regression, decision trees, random forests, or neural networks, and training them on the prepared data. This training process enables the models to learn and recognize the patterns associated with customer churn. Fine-tuning the models through optimization and hyperparameter tuning is crucial to improving their accuracy and performance. By adjusting various model parameters and testing different configurations, businesses can achieve the best results in predicting churn.

Once the models are trained and optimized, they are deployed and integrated into the business workflow. This implementation allows companies to start making predictions about which customers are at risk of churning and take proactive measures to retain them. Monitoring and maintenance are vital to ensure the model's continued effectiveness. Regular updates and adjustments are necessary to adapt to changing customer behaviour and trends, ensuring that the churn prediction model remains accurate over time.

The benefits of churn prediction are significant. Enhanced customer retention is a primary advantage, as identifying at-risk customers early allows businesses to take proactive steps to retain them. This can involve offering personalized incentives, improving customer support, or addressing specific pain points that may lead to churn. Improved customer satisfaction is another benefit, as targeted interventions can address customers' needs and concerns, enhancing their overall experience with the company. Reduced revenue loss is a direct result of retaining customers who might otherwise have churned, leading to increased profitability. Additionally, long-term profitability is achieved by building and maintaining long-term relationships with customers, resulting in sustained business growth.

Churn prediction using ML models is a strategic approach that leverages data-driven insights to understand customer behaviour and take informed actions to enhance retention and overall business performance. By investing in churn prediction, businesses can create a more personalized and responsive customer experience, ultimately leading to higher customer satisfaction and loyalty.

Methodology-

1. Data Collection and Preparation: Gather data on customer behaviour, demographics, transaction history, and other relevant factors. This data serves as the foundation for building the churn prediction model.

2. Exploratory Data Analysis (EDA): Conduct an initial analysis of the collected data to identify patterns and trends that may indicate potential churn. This helps in understanding the key factors contributing to churn.

3. Model Selection and Training: Choose appropriate ML models, such as logistic regression, decision trees, random forests, or neural networks. Train these models on the prepared data to learn the patterns associated with churn.

4. Model Optimization and Hyperparameter Tuning: Fine-tune the models to improve their accuracy and performance. This involves adjusting the model's parameters and testing different configurations to achieve the best results.

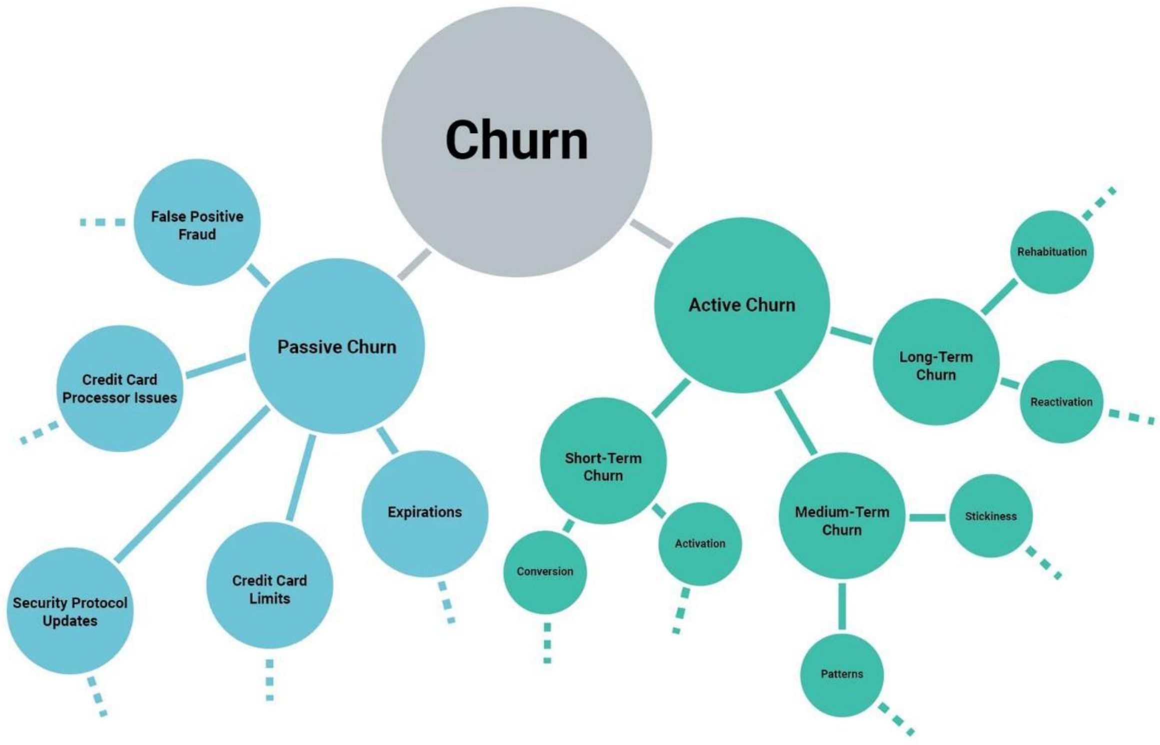
5. Model Deployment and Integration: Implement the trained churn prediction model into the business workflow. This enables the company to start making predictions about which customers are at risk of churning.

6. Monitoring and Maintenance: Continuously monitor the model's performance and update it as needed to ensure it remains effective. Regular maintenance is crucial to adapt to changing customer behaviour and trends.

WHAT IS CHURN?

Churn, in a business context, refers to the phenomenon of customers discontinuing their relationship with a company or ceasing to use its products or services. It is a critical metric for businesses as it directly impacts their revenue and growth. Customer churn can be measured as the percentage of customers who leave over a specific period.

Understanding and predicting churn is essential for businesses to retain customers and maintain a competitive edge. By identifying factors that contribute to churn, companies can implement strategies to improve customer satisfaction, enhance loyalty, and reduce attrition rates.



LIBRARIES USED

➢ TensorFlow

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google. TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open-source license on November 9, 2015.

➢ Numpy

Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows Numpy to seamlessly and speedily integrate with a wide variety of databases.

➢ Pandas

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with Pandas is used in a wide range of fields including academic and commercial domains
including finance, economics, Statistics, analytics, etc.

➢ Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

➢ Scikit – learn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.

Machine Learning Models: Decision Tree, Random Forest, and XGBoost

Decision Tree

A Decision Tree is a versatile and interpretable model used for both classification and regression tasks. It splits the dataset into subsets based on the values of input features. Each node represents a feature, each branch represents a decision rule, and each leaf node represents an outcome.

Features:

- **Nodes and Branches:** The structure of the tree, with nodes representing features and branches representing decision rules.
- **Root Node:** The topmost node, representing the most significant feature.
- **Internal Nodes:** Nodes that represent features used for further splitting the data.
- **Leaf Nodes:** Terminal nodes representing the final prediction (classification or regression).
- **Splitting:** The process of dividing nodes into sub-nodes based on decision rules.
- **Pruning:** The technique used to remove parts of the tree that do not contribute significantly, reducing complexity and preventing overfitting.
- **Impurity Measures:** Metrics like Gini Index, Information Gain, and Chi-square to determine the best feature for splitting.

Advantages:

- Easy to understand and interpret.
- Can handle both numerical and categorical data.
- Requires little data preprocessing.

Disadvantages:

- Prone to overfitting, especially with complex datasets.
- Can be unstable with small changes in data leading to different trees.

Random Forest

Random Forest is an ensemble learning method that creates multiple decision trees and merges their predictions. It uses the concept of bagging to improve accuracy and robustness.

Features:

- **Ensemble Learning:** Combines multiple decision trees to enhance model performance.
- **Bagging (Bootstrap Aggregating):** Generates multiple samples from the training data by sampling with replacement.
- **Feature Randomness:** Each tree is built using a random subset of features, increasing diversity and reducing correlation.
- **Majority Voting:** For classification, the final prediction is based on the majority vote of the individual trees.
- **Averaging:** For regression, the final prediction is the average of the individual trees' predictions.
- **Feature Importance:** Measures the contribution of each feature to the model's predictive power.

Advantages:

- Reduces overfitting by averaging multiple trees.
- Handles large datasets with high dimensionality well.
- Provides feature importance.

Disadvantages:

- Can be computationally intensive and slower to predict than individual trees.
- Requires careful tuning of hyperparameters for optimal performance.

XGBoost (Extreme Gradient Boosting)

XGBoost is an optimized distributed gradient boosting library designed for efficiency, flexibility, and high performance. It builds decision trees sequentially, with each new tree aiming to correct errors made by previous ones.

Features:

- **Gradient Boosting:** Sequentially builds trees to correct errors from previous ones.
- **Regularization:** Incorporates L1 (Lasso) and L2 (Ridge) regularization to prevent overfitting.
- **Parallel Processing:** Utilizes parallel computing for faster training and prediction.
- **Tree Pruning:** Uses advanced pruning based on maximum delta step for efficient tree construction.
- **Handling Missing Data:** Automatically learns the best way to handle missing data.
- **Cross-validation:** Supports built-in cross-validation for model evaluation and selection.
- **Scalability:** Highly scalable and can handle large datasets efficiently.

Advantages:

- Highly efficient and fast due to parallel processing.
- Offers regularization to reduce overfitting.
- Provides feature importance and handles missing data well.

Disadvantages:

- Can be more complex to tune and requires careful selection of hyperparameters.
- May be less interpretable than simpler models like decision trees.

Summary

Churn prediction using machine learning models such as Decision Trees, Random Forests, and XGBoost involves identifying customers who are likely to stop using a company's products or services. By analyzing customer data, these models help businesses take proactive measures to retain at-risk customers.

By leveraging these models, businesses can enhance customer retention, improve satisfaction, reduce revenue loss, and achieve long-term profitability. Each model has unique strengths, and the choice depends on the specific requirements of the task and dataset.

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 teh 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	Multipl
0	7590-VHVEG	Female	0	Yes	No	1	No	N
1	5575-GNVDE	Male	0	No	No	34	Yes	
2	3668-QPYBK	Male	0	No	No	2	Yes	
3	7795-CFOCW	Male	0	No	No	45	No	N
4	9237-HQITU	Female	0	No	No	2	Yes	

5 rows × 21 columns

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

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

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	Multipl
0	7590-VHVEG	Female	0	Yes	No	1	No	Nc
1	5575-GNVDE	Male	0	No	No	34	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	Inte
0	Female	0	Yes	No	1	No	No phone service	
1	Male	0	No	No	34	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	I
488	Female	0	Yes	Yes	0	No	No phone service	
753	Male	0	No	Yes	0	Yes	No	
936	Female	0	Yes	Yes	0	Yes	No	
1082	Male	0	Yes	Yes	0	Yes	Yes	
1340	Female	0	Yes	Yes	0	No	No phone service	
3331	Male	0	Yes	Yes	0	Yes	No	
3826	Male	0	Yes	Yes	0	Yes	Yes	
4380	Female	0	Yes	Yes	0	Yes	No	
5218	Male	0	Yes	Yes	0	Yes	No	
6670	Female	0	Yes	Yes	0	Yes	Yes	
6754	Male	0	No	Yes	0	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 mmissing values in the dataset
3. Missing values 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	Inte
0	Female	0	Yes	No	1	No	No phone service	
1	Male	0	No	No	34	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

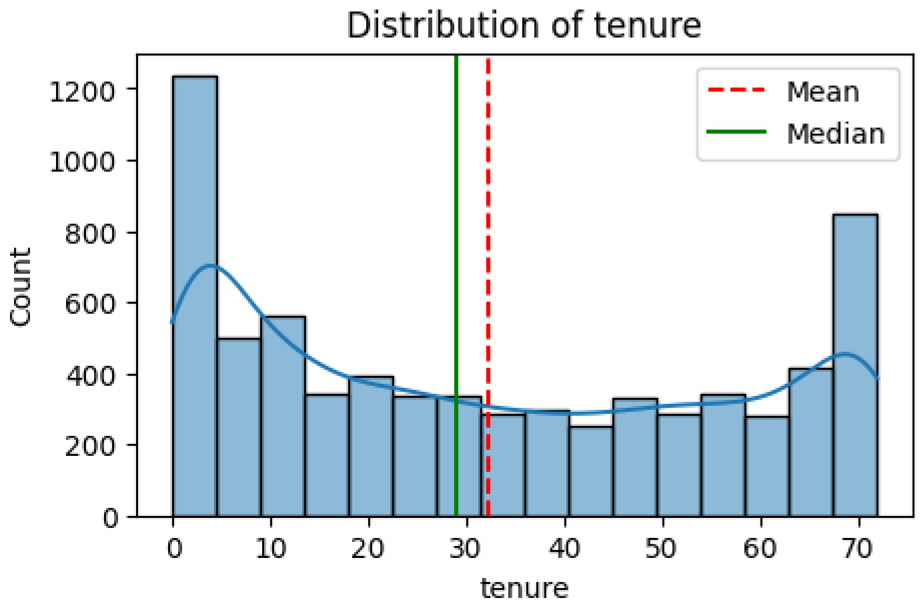
Numerical Features - Analysis

Understand the distribution of teh numerical features

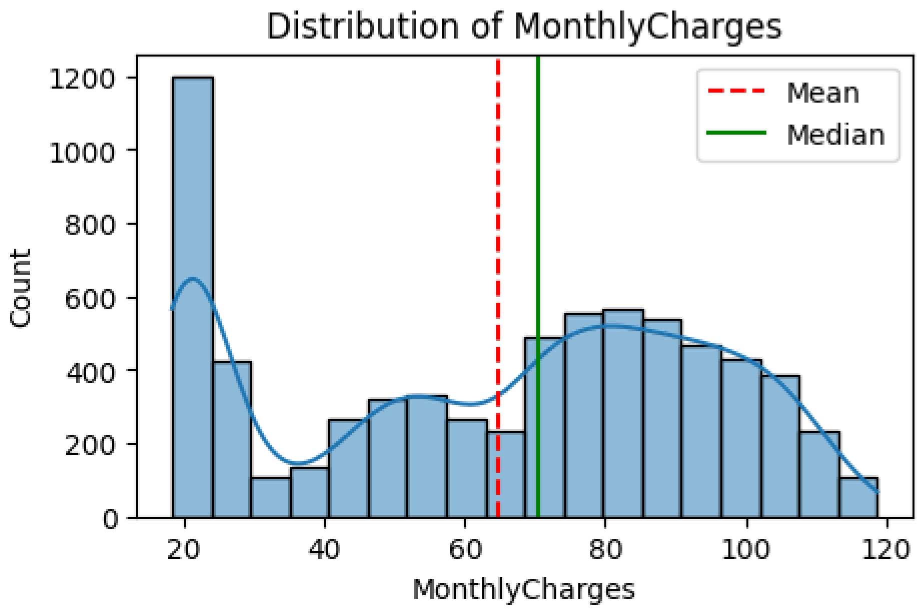
```
def plot_histogram(df, column_name):

  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()
```

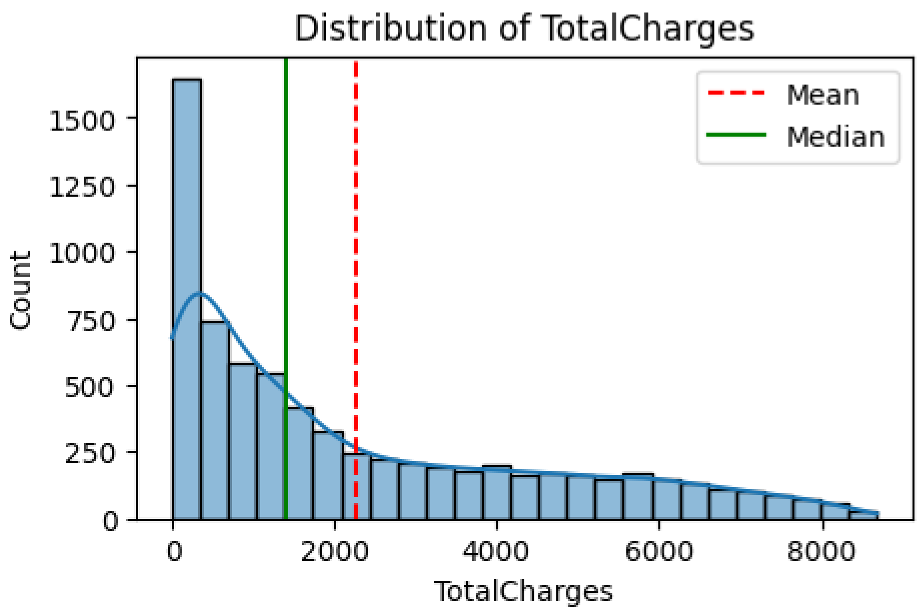
```
plot_histogram(df, "tenure")
```



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



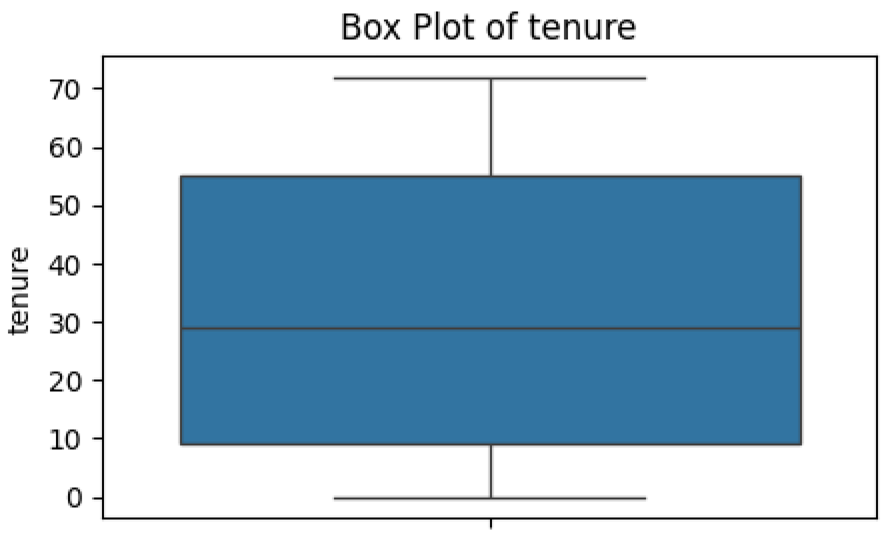
```
plot_histogram(df, "TotalCharges")
```



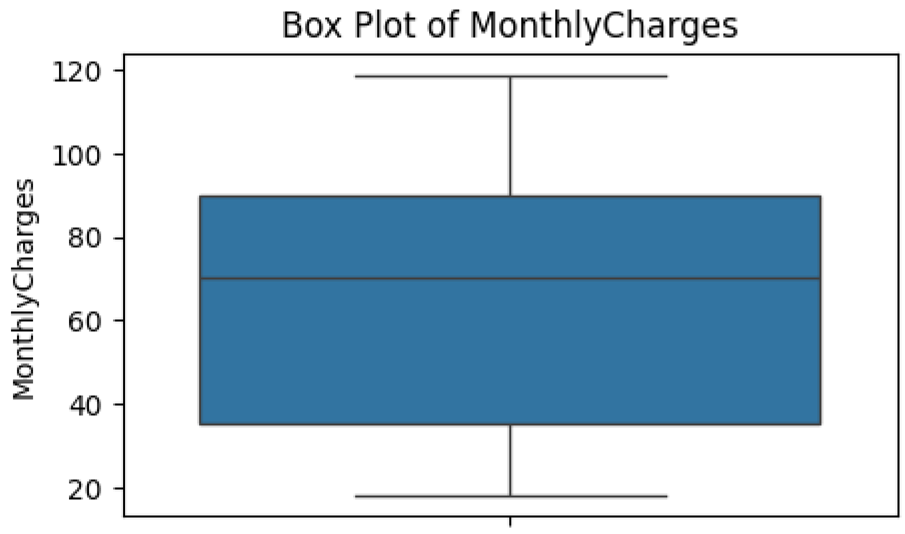
Box plot for numerical features

```
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
```

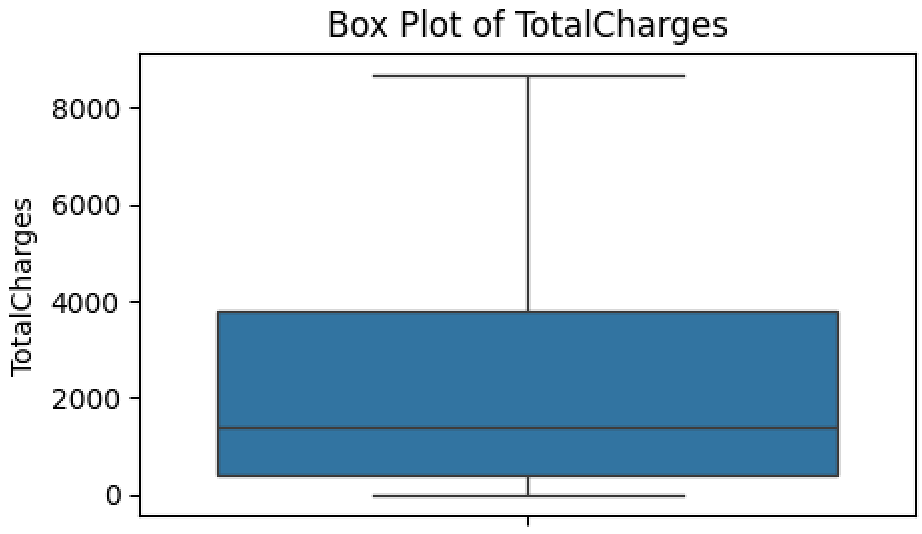
```
plot_boxplot(df, "tenure")
```



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

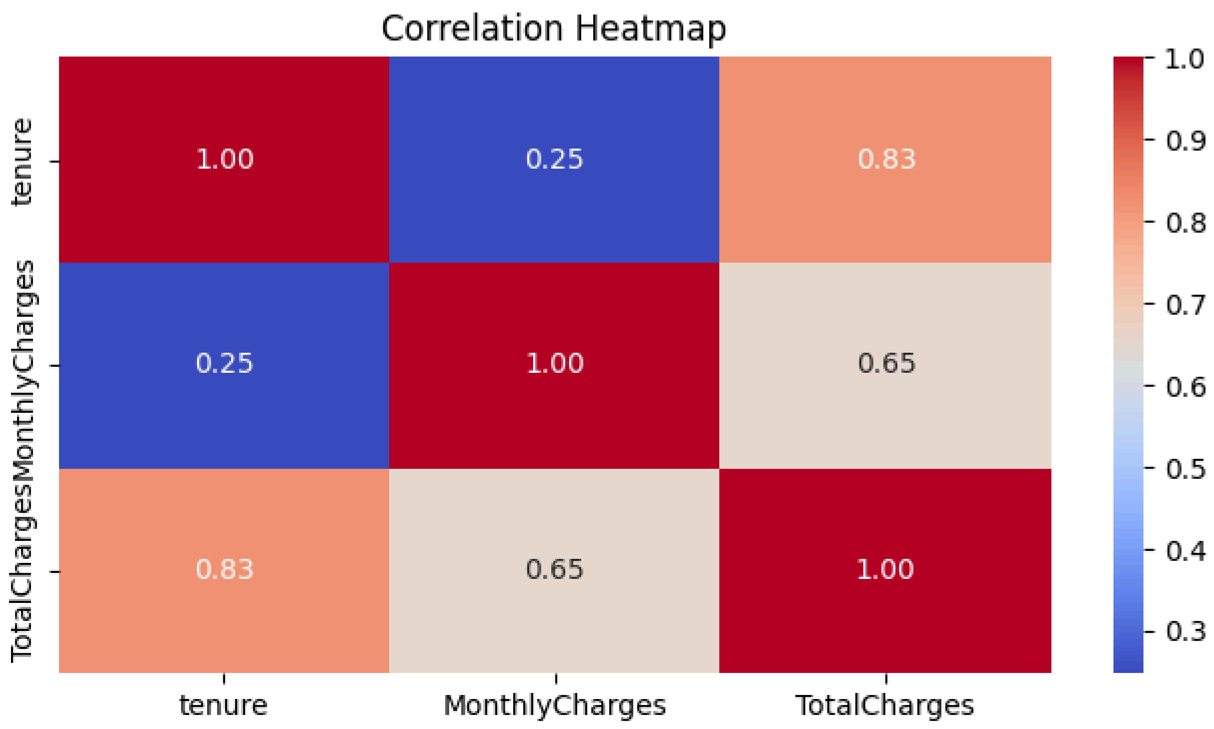


```
plot_boxplot(df, "TotalCharges")
```



Correlation Heatmap for numerical columns

```
# correlation matrix - heatmap
plt.figure(figsize=(8, 4))
sns.heatmap(df[["tenure", "MonthlyCharges", "TotalCharges"]].corr(), annot=True, cmap="coolw
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	Inte
0	Female	0	Yes	No	1	No	No phone service	
1	Male	0	No	No	34	Yes	No	
2	Male	0	No	No	2	Yes	No	

Label encoding of target column

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

```
<ipython-input-39-b6eb27bc3ee0>:1: FutureWarning: Downcasting behavior in `replace` is c
  df["Churn"] = df["Churn"].replace({"Yes": 1, "No": 0})
```

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

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Inte
0	Female	0	Yes	No	1	No	No phone service	
1	Male	0	No	No	34	Yes	No	
2	Male	0	No	No	2	Yes	No	

```
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	Inte
0	0	0	1	0	1	0	1	
1	1	0	0	0	34	1	0	
2	1	0	0	0	2	1	0	
3	1	0	0	0	45	0	1	
4	0	0	0	0	2	1	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
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
```

Conclusion and Future Enhancement

Conclusion

Churn prediction using machine learning models such as Decision Trees, Random Forests, and XGBoost is a critical approach for businesses looking to understand and mitigate customer attrition. This process involves the systematic analysis of customer data to identify patterns and indicators that signal the likelihood of a customer leaving a product or service. By leveraging these models, businesses can take proactive steps to retain their customers, thereby enhancing customer satisfaction and reducing revenue loss.

Decision Trees are one of the simplest and most interpretable machine learning models. They work by splitting the dataset into subsets based on feature values, creating a tree-like structure where each node represents a feature, each branch represents a decision rule, and each leaf node represents an outcome. The ease of interpretation makes decision trees particularly useful for gaining insights into the factors contributing to customer churn. However, they are prone to overfitting, especially with complex datasets, which can limit their predictive power.

Random Forests address the overfitting issue inherent in decision trees by creating an ensemble of multiple decision trees. This ensemble approach, known as bagging (Bootstrap Aggregating), involves generating multiple samples from the training data and building separate trees for each sample. The final prediction is made by aggregating the predictions of the individual trees, which enhances accuracy and robustness. Random Forests also introduce randomness by selecting a subset of features for each tree, which reduces correlation between trees and improves generalization. This model is well-suited for large datasets and high-dimensional spaces, making it a powerful tool for churn prediction. Additionally, Random Forests provide feature importance measures, which help identify the most influential factors driving customer churn.

XGBoost (Extreme Gradient Boosting) is an advanced and highly efficient implementation of gradient boosting. Unlike Random Forests, which build trees independently, XGBoost builds trees sequentially, with each new tree correcting the errors made by the previous ones. This iterative process allows XGBoost to achieve high levels of accuracy and performance. XGBoost incorporates regularization techniques (L1 and L2) to prevent overfitting, ensuring that the model generalizes well to new data. It also supports parallel processing, making it fast and scalable for large datasets. XGBoost is known for its flexibility, as it can handle missing data and offers built-in cross-validation for model evaluation and selection. The ability to handle large and complex datasets efficiently makes XGBoost a popular choice for churn prediction tasks.

In conclusion, churn prediction using machine learning models like Decision Trees, Random Forests, and XGBoost is a strategic approach that enables businesses to identify at-risk customers and take proactive measures to retain them. Each model offers unique strengths: Decision Trees provide simplicity and interpretability, Random Forests offer robustness and reduced overfitting, and XGBoost delivers high performance and efficiency.

Future Enhancement

Future enhancements in churn prediction can focus on integrating advanced algorithms like deep learning models, developing real-time prediction capabilities, and improving explainability and transparency. By investing in churn prediction and continuously evolving these models, businesses can enhance customer retention, improve satisfaction, and achieve long-term profitability.