***Enhancing Customer Churn Prediction: A Decision Tree Approach***

**Introduction:**

In this analysis, we applied a Decision Tree Classifier to predict customer churn using the provided "customers.csv" dataset. The goal was to build an accurate model that predicts whether a customer will churn based on various features.

**Data Preprocessing:**

The dataset was loaded, and its structure was examined. Categorical columns (Device, payment, Gender, OrderCat, MaritalStatus) were encoded using Label Encoding for model compatibility. Numerical features were scaled using StandardScaler, MinMaxScaler, and RobustScaler to standardize the data. The dataset was loaded and its structure was examined. The following preprocessing steps were applied:

import pandas as pd

from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler, LabelEncoder

customers = pd.read\_csv('customers.csv')

# Encode categorical columns

categorical\_cols = ['Device', 'payment', 'Gender', 'OrderCat', 'MaritalStatus']

for col in categorical\_cols:

label\_encoders[col] = LabelEncoder()

customers[col] = label\_encoders[col].fit\_transform(customers[col])

# StandardScaler

standard\_scaler = StandardScaler()

standard\_scaled\_data = standard\_scaler.fit\_transform(numerical\_features)

standard\_scaled\_df = pd.DataFrame(standard\_scaled\_data, columns=numerical\_features.columns)

# MinMaxScaler

min\_max\_scaler = MinMaxScaler()

min\_max\_scaled\_data = min\_max\_scaler.fit\_transform(numerical\_features)

min\_max\_scaled\_df = pd.DataFrame(min\_max\_scaled\_data, columns=numerical\_features.columns)

# RobustScaler

robust\_scaler = RobustScaler()

robust\_scaled\_data = robust\_scaler.fit\_transform(numerical\_features)

robust\_scaled\_df = pd.DataFrame(robust\_scaled\_data, columns=numerical\_features.columns)

**Feature Selection:**

Feature correlations were analyzed, and the top correlated features with the target variable ('Churn') were selected. The model was trained using these top N correlated features to assess their impact on accuracy.

feature\_correlations = customers.corr(numeric\_only=True)['Churn'].abs().sort\_values(ascending=False)

top\_features = feature\_correlations[1:6].index # Exclude 'Churn' itself and select top 5 features

new\_features\_matrix = customers[top\_features]

**Training and Testing Data Size:**

The impact of changing the training and test data size was explored. Different sizes (60%, 70%, 80%, and 90% training data) were used to observe the accuracy variations.

# Vary the training data size and observe the impact on accuracy

train\_sizes = [0.6, 0.7, 0.8, 0.9]

for size in train\_sizes:

# Split the data into training and test sets with the current size

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_encoded, y\_encoded, test\_size=1 - size, random\_state=42)

# Initialize and train the decision tree classifier

clf = DecisionTreeClassifier(random\_state=42)

clf.fit(X\_train, y\_train)

# Make predictions and calculate accuracy

y\_pred = clf.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

# Print the training size and accuracy

print(f"Training Size: {size}, Test Size: {1 - size}, Accuracy: {accuracy}")

**Parameter Tuning:**

A parameter tuning process was executed to find the optimal values for 'max\_depth' in the Decision Tree model. Training and validation accuracies were recorded for different 'max\_depth' values.

parameters = [3,5,7,10]

dfs = []

for param\_value in parameters:

clf = DecisionTreeClassifier(max\_depth=param\_value)

clf.fit(X\_train, y\_train)

training\_accuracy = accuracy\_score(y\_train, clf.predict(X\_train))

validation\_accuracy = accuracy\_score(y\_test, clf.predict(X\_test))

dfs.append(pd.DataFrame({'Parameter': [param\_value],

'Training Accuracy': [training\_accuracy],

'Validation Accuracy': [validation\_accuracy]}))

df = pd.concat(dfs, ignore\_index=True)

print(df)

A plot was created to visualize the relationship between training and validation accuracies with changing 'max\_depth'.

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))

plt.plot(df['Parameter'], df['Training Accuracy'], marker='o', label='Training Accuracy')

plt.plot(df['Parameter'], df['Validation Accuracy'], marker='o', label='Validation Accuracy')

plt.xlabel('Parameter Value')

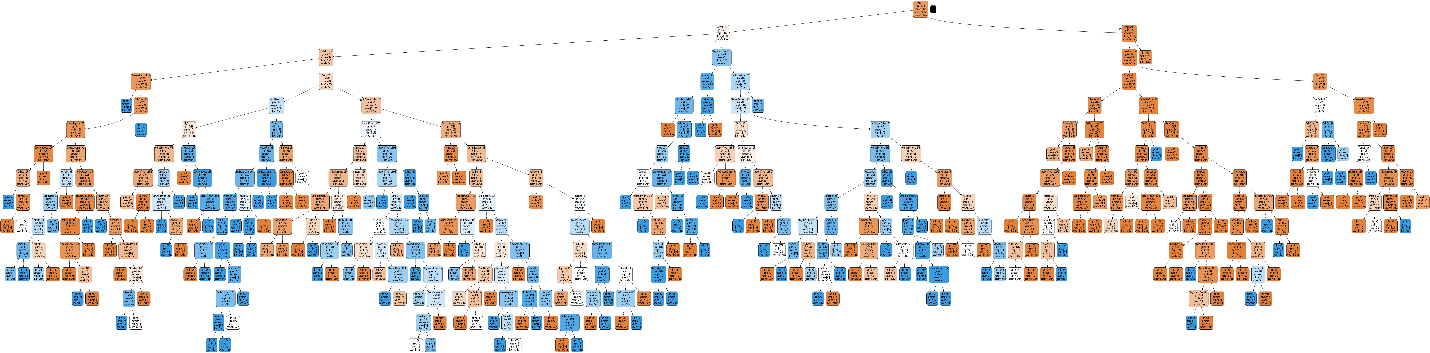
plt.ylabel('Accuracy')

plt.title('Training Accuracy vs Validation Accuracy')

plt.legend()

plt.grid(True)

plt.show()



**Binning Technique:**

Binning was applied to selected columns ('Tenure', 'HourActive', 'DevicesRegistered', 'OrderCount', 'DaySinceLastOrder') to discretize continuous data into categorical bins. The model was retrained with the binned data to assess its impact on accuracy. The accuracy with and without binning was compared, indicating a slight decrease in accuracy after implementing the binning technique.

# Define bin edges and labels for each column

tenure\_bins = [0, 2, 5, float('inf')]

tenure\_labels = ['New', 'Regular', 'Long-term']

customers['TenureCategory'] = pd.cut(customers['Tenure'], bins=tenure\_bins, labels=tenure\_labels)

# ... Repeat the binning process for other columns ...

#Training and testing the model again after binning:

X\_binned = customers[['TenureCategory', 'HourActiveCategory', 'DevicesCategory', 'OrderCountCategory', 'DaysSinceLastOrderCategory']]

X\_binned\_encoded = pd.get\_dummies(X\_binned, drop\_first=True)

# Assuming you have a target variable y defined

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_binned\_encoded, y, test\_size=0.2, random\_state=42)

clf = DecisionTreeClassifier(random\_state=42)

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

accuracy\_with\_binning = accuracy\_score(y\_test, y\_pred)

print("Accuracy with Binning:", accuracy\_with\_binning)

# Accuracy without binning

accuracy\_without\_binning = 0.8960923623445826

# Accuracy with binning

accuracy\_with\_binning = 0.8383658969804618

# Calculate the difference in accuracies

accuracy\_difference = accuracy\_with\_binning - accuracy\_without\_binning

# Print the results

print("Accuracy without Binning:", accuracy\_without\_binning)

print("Accuracy with Binning:", accuracy\_with\_binning)

print("Accuracy Difference (With Binning - Without Binning):", accuracy\_difference)

**Conclusion:**

In summary, the Decision Tree Classifier was employed to predict customer churn based on various features. Exploratory data analysis, feature selection, parameter tuning, and binning techniques were utilized to enhance the model's accuracy. While some techniques led to a decrease in accuracy, they provided valuable insights into the data and model behavior, facilitating better decision-making for churn prediction.

**Note:**

Throughout the analysis, different techniques were applied to fine-tune the model. The results indicated that the choice of features, data preprocessing methods, and parameter values significantly impact the model's performance. Further experimentation and refinement are recommended to improve the accuracy and robustness of the predictive model.