# Import required libraries

import pandas as pd

import numpy as np

import graphviz

from sklearn import tree

from sklearn.tree import DecisionTreeClassifier, export\_graphviz

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report, confusion\_matrix, roc\_curve, auc

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy\_score, precision\_recall\_fscore\_support

from imblearn.over\_sampling import SMOTE

import matplotlib.pyplot as plt

import seaborn as sns

data= pd.read\_csv("updated\_data.csv")

# Step 1: Detailed EDA

print("Dataset Overview:")

print(data.head())

print("\nDataset Info:")

print(data.info())

print("\nMissing Values:")

print(data.isnull().sum())

print("\nStatistical Summary:")

print(data.describe())

# Check for missing values

missing\_values = data.isnull().sum()

missing\_percentage = (missing\_values / len(data)) \* 100

print("\nMissing Values Count:")

print(missing\_values)

print("\nPercentage of Missing Values:")

print(missing\_percentage)

# Distribution of a sample numerical column (e.g., 'Customer\_Age' if present)

plt.figure(figsize=(8, 4))

sns.histplot(data['Customer\_Age'], kde=True, color='skyblue', bins=10)

plt.title("Customer Age Distribution")

plt.xlabel("Age")

plt.ylabel("Frequency")

plt.show()

categorical\_columns = ['Gender', 'Education\_Level', 'Marital\_Status', 'Income\_Category', 'Card\_Category']

for col in categorical\_columns:

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

value\_counts = data[col].value\_counts()

colors = sns.color\_palette("cool", len(value\_counts))

plt.pie(value\_counts, labels=value\_counts.index, autopct='%1.1f%%', colors=colors, textprops={'fontsize': 10})

plt.legend(title=col, loc="best", fontsize=10)

plt.title(f"Distribution of {col}")

plt.show()

excluded\_columns = [

'CLIENTNUM', 'Attrition\_Flag', 'Gender', 'Education\_Level',

'Marital\_Status', 'Income\_Category', 'Card\_Category',

'Naive\_Bayes\_Classifier\_Attrition\_Flag\_Card\_Category\_Contacts\_Count\_12\_mon\_Dependent\_count\_Education\_Level\_Months\_Inactive\_12\_mon\_1',

'Naive\_Bayes\_Classifier\_Attrition\_Flag\_Card\_Category\_Contacts\_Count\_12\_mon\_Dependent\_count\_Education\_Level\_Months\_Inactive\_12\_mon\_2'

]

numeric\_data = data.drop(columns=excluded\_columns, errors='ignore')

plt.figure(figsize=(12, 8))

sns.heatmap(numeric\_data.corr(), annot=True, fmt=".2f", cmap="coolwarm", linewidths=0.5)

plt.title("Heatmap of Numerical Features")

plt.show()

data.columns

data.drop('CLIENTNUM', axis=1, inplace=True)

data.drop('Naive\_Bayes\_Classifier\_Attrition\_Flag\_Card\_Category\_Contacts\_Count\_12\_mon\_Dependent\_count\_Education\_Level\_Months\_Inactive\_12\_mon\_1',axis=1,inplace=True)

data.drop('Naive\_Bayes\_Classifier\_Attrition\_Flag\_Card\_Category\_Contacts\_Count\_12\_mon\_Dependent\_count\_Education\_Level\_Months\_Inactive\_12\_mon\_2',axis=1,inplace=True)

data.columns

# List of categorical columns

categorical\_columns = ['Gender', 'Education\_Level', 'Marital\_Status', 'Income\_Category', 'Card\_Category']

# Encoding the categorical columns

label\_encoders = {}

for column in categorical\_columns:

le = LabelEncoder()

data[column] = le.fit\_transform(data[column])

label\_encoders[column] = le

data

# Drop the 'Attrition\_Flag' column from the dataset to create the features (X)

X = data.drop(columns=['Attrition\_Flag'])

# Define 'Attrition\_Flag' as the target variable (y)

y = data['Attrition\_Flag']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state = 42)

# Handle imbalance with SMOTE (Synthetic Minority Oversampling Technique).

smote = SMOTE(random\_state=42)

X\_train\_balanced, y\_train\_balanced = smote.fit\_resample(X\_train, y\_train)

my\_classifier = tree.DecisionTreeClassifier(criterion = 'entropy', random\_state=101)

my\_classifier = my\_classifier.fit(X\_train\_balanced, y\_train\_balanced)

tree.plot\_tree(my\_classifier)

# Visualize with Graphviz

dot\_data = export\_graphviz(my\_classifier, out\_file=None,

feature\_names=X.columns,

class\_names=['No', 'Yes'],

filled=True, rounded=True,

special\_characters=True)

graph = graphviz.Source(dot\_data)

graph.render("decision\_tree\_2")

graph.view()

y\_pred = my\_classifier.predict(X\_test)

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

print(f"Decision Tree Accuracy: {accuracy \* 100:.2f}%")

# Confusion Matrix

cm = confusion\_matrix(y\_test, y\_pred)

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

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])

plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

report = classification\_report(y\_test, y\_pred)

print("\nClassification Report:")

print(report)

# ROC Curve

y\_test\_binary = y\_test.map({'Attrited Customer': 1, 'Existing Customer': 0})

y\_prob = my\_classifier.predict\_proba(X\_test)[:, 1] # Probability estimates

fpr, tpr, \_ = roc\_curve(y\_test\_binary, y\_prob)

roc\_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='grey', linestyle='--')

plt.xlabel('False Positive Rate')

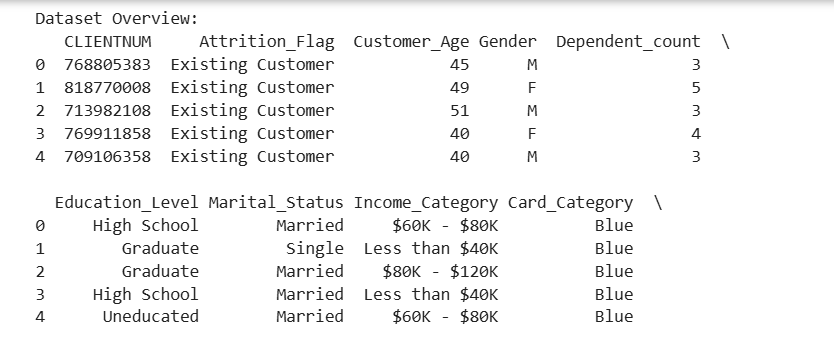
plt.ylabel('True Positive Rate')

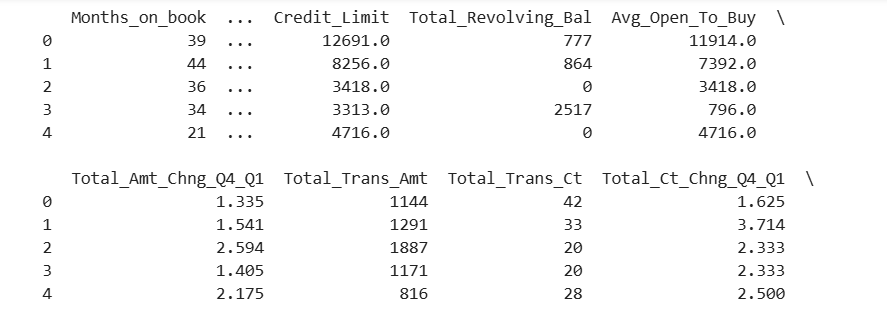
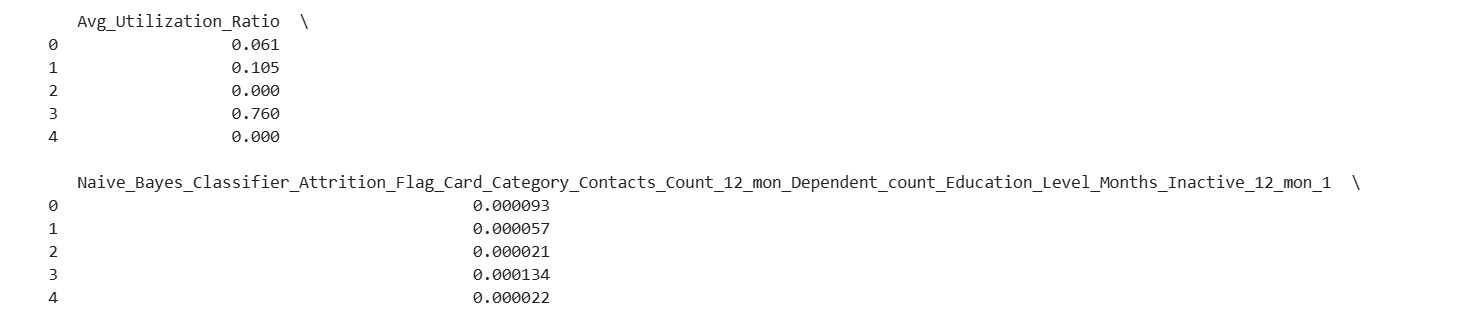
plt.title('Receiver Operating Characteristic (ROC) Curve')

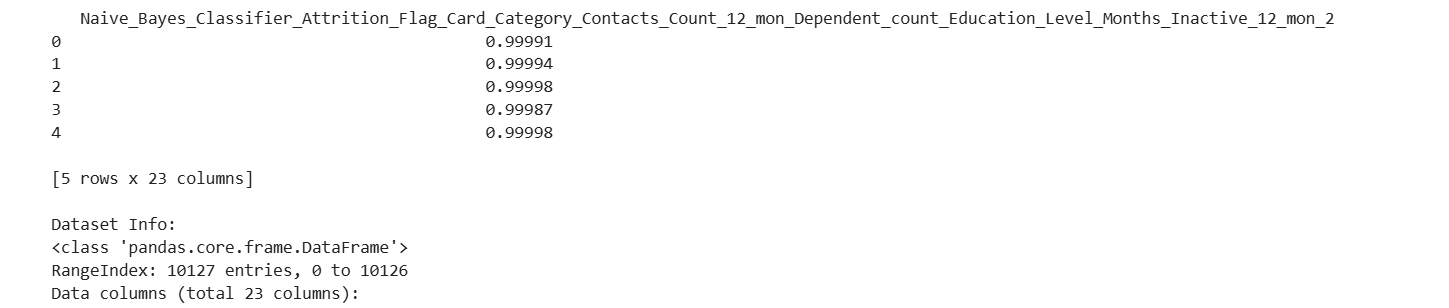
plt.legend(loc='lower right')

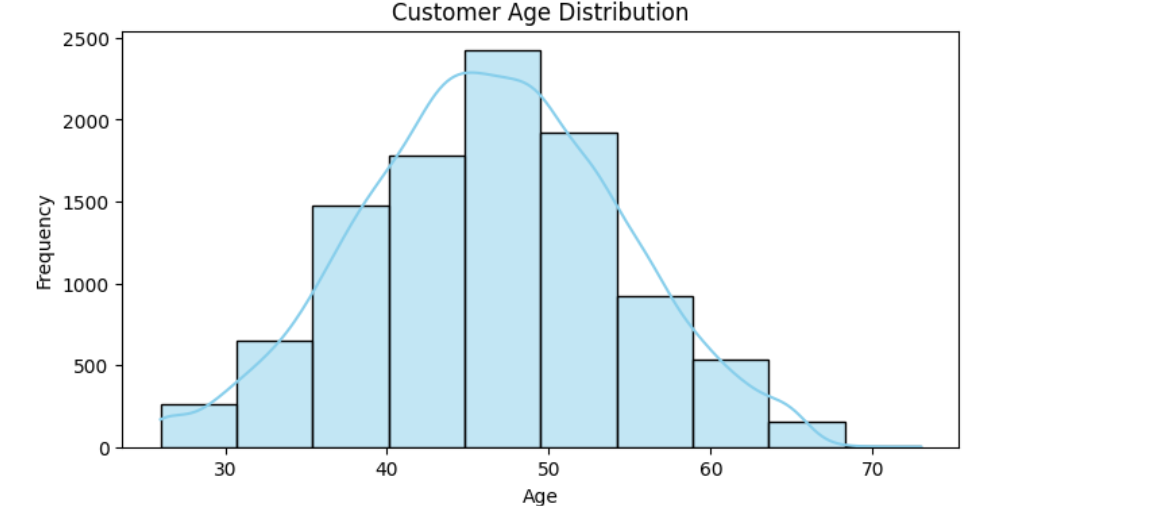
plt.show()

**OUTPUT:**

****

****

****

****

