LAB 03: Demonstrate KNN classification.

CODE:

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

from sklearn.neighbors import KNeighborsClassifier

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

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

from sklearn.model\_selection import GridSearchCV

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

data.isnull().sum()

print(data.dtypes)

data.describe()

target\_col = 'Attrition\_Flag'

target\_counts = data[target\_col].value\_counts()

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

plt.pie(target\_counts, labels=target\_counts.index, autopct='%1.1f%%', startangle=90)

plt.title(f"Distribution of Attrition Falg")

plt.show()

# Drop the CLIENTNUM column as it's not useful for prediction

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)

# Display the remaining columns

print(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

# Convert the target variable

data['Attrition\_Flag'] = data['Attrition\_Flag'].map({'Existing Customer': 0, 'Attrited Customer': 1})

# Display the first few rows to verify encoding

print(data.head())

# 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']

# Split the data into training and testing sets (70% train, 30% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Initialize the StandardScaler to scale the features for KNN

scaler = StandardScaler()

# Fit the scaler on the training data and transform both the training and test data

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Define a parameter grid to search for the best number of neighbors (n\_neighbors) for the KNN classifier

param\_grid = {'n\_neighbors': range(3, 15)}

# Perform GridSearchCV to find the best number of neighbors for KNN

grid\_search = GridSearchCV(KNeighborsClassifier(), param\_grid, scoring='f1\_weighted')

# Fit the grid search to the training data to find the optimal K

grid\_search.fit(X\_train, y\_train)

# Print the best number of neighbors found during the grid search

print(f"Best K: {grid\_search.best\_params\_['n\_neighbors']}")

# Get the best K from grid search

best\_k = grid\_search.best\_params\_['n\_neighbors']

# Initialize the KNN classifier with the best K

knn = KNeighborsClassifier(n\_neighbors=best\_k)

# Fit the model

knn.fit(X\_train\_scaled, y\_train)

# Predict the labels for the test set

y\_pred = knn.predict(X\_test\_scaled)

# Print the predicted labels for the test data

print("Predicted Labels: ",y\_pred)

# Calculate and print the accuracy of the model

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

# Print a detailed classification report that includes precision, recall, and F1 score

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

# Print the confusion matrix to see the distribution of true positives, false positives, etc.

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

OUTPUT:















