import pandas as pd

import numpy as np

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

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics import classification\_report

import re

import string

import matplotlib.pyplot as plt

# Set the path for the dataset files

true\_path = 'True.csv'

fake\_path = 'Fake.csv'

# Read the data

true\_news = pd.read\_csv(true\_path)

fake\_news = pd.read\_csv(fake\_path)

# Add 'class' column to both datasets

true\_news['class'] = 1

fake\_news['class'] = 0

# Remove 10 entries from the end of each dataset for manual testing

true\_manual\_testing = true\_news.tail(10)

fake\_manual\_testing = fake\_news.tail(10)

true\_news = true\_news.iloc[:-10]

fake\_news = fake\_news.iloc[:-10]

data\_manual\_testing = pd.concat([true\_manual\_testing, fake\_manual\_testing])

# Concatenate the remaining entries of both datasets

data\_merge = pd.concat([true\_news, fake\_news])

# Print the count of articles per subject and plot them in a bar chart

subject\_count = data\_merge['subject'].value\_counts()

print(subject\_count)

subject\_count.plot(kind='bar')

plt.show()

# Print the count of fake and true news articles and plot them in a pie chart

class\_count = data\_merge['class'].value\_counts()

print(class\_count)

class\_count.plot(kind='pie', autopct='%1.1f%%')

plt.show()

# Drop unnecessary columns

data = data\_merge.drop(columns=['title', 'subject', 'date'])

# Apply data shuffling

data = data.sample(frac=1).reset\_index(drop=True)

# Check for missing values

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

# Define the filtering function

def filtering(text):

text = re.sub(r'http\S+', '', text) # Remove URLs

text = re.sub(r'<.\*?>', '', text) # Remove HTML tags

text = re.sub(r'[%s]' % re.escape(string.punctuation), '', text) # Remove punctuation

text = re.sub(r'\d+', '', text) # Remove numbers

text = text.lower() # Convert to lowercase

return text

# Apply filtering function to 'data'

data['text'] = data['text'].apply(filtering)

# Split the data into text and class labels

x = data['text']

y = data['class']

# Split the data into train and test sets

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

# Convert text data into numerical vectors using TfidfVectorizer

vector = TfidfVectorizer()

x\_train = vector.fit\_transform(x\_train)

x\_test = vector.transform(x\_test)

# Train a Logistic Regression model

log\_reg = LogisticRegression()

log\_reg.fit(x\_train, y\_train)

y\_pred\_log\_reg = log\_reg.predict(x\_test)

# Print the classification report for Logistic Regression model

print("Logistic Regression Classification Report:\n")

print(classification\_report(y\_test, y\_pred\_log\_reg))

# Train a Decision Tree Classifier model

dec\_tree = DecisionTreeClassifier()

dec\_tree.fit(x\_train, y\_train)

y\_pred\_dec\_tree = dec\_tree.predict(x\_test)

# Print the classification report for Decision Tree Classifier model

print("Decision Tree Classifier Classification Report:\n")

print(classification\_report(y\_test, y\_pred\_dec\_tree))

# Define a function to convert numeric class labels to text labels

def output\_label(n):

return "True News" if n == 1 else "Fake News"

# Define a function for manual testing

def manual\_testing(news):

news = filtering(news)

news\_vector = vector.transform([news])

pred\_log\_reg = log\_reg.predict(news\_vector)

pred\_dec\_tree = dec\_tree.predict(news\_vector)

return f"Logistic Regression Prediction: {output\_label(pred\_log\_reg[0])}\nDecision Tree Prediction: {output\_label(pred\_dec\_tree[0])}"

# Take an input news text from the user and call 'manual\_testing' function

input\_news = input("Enter the news text to test: ")

print(manual\_testing(input\_news))