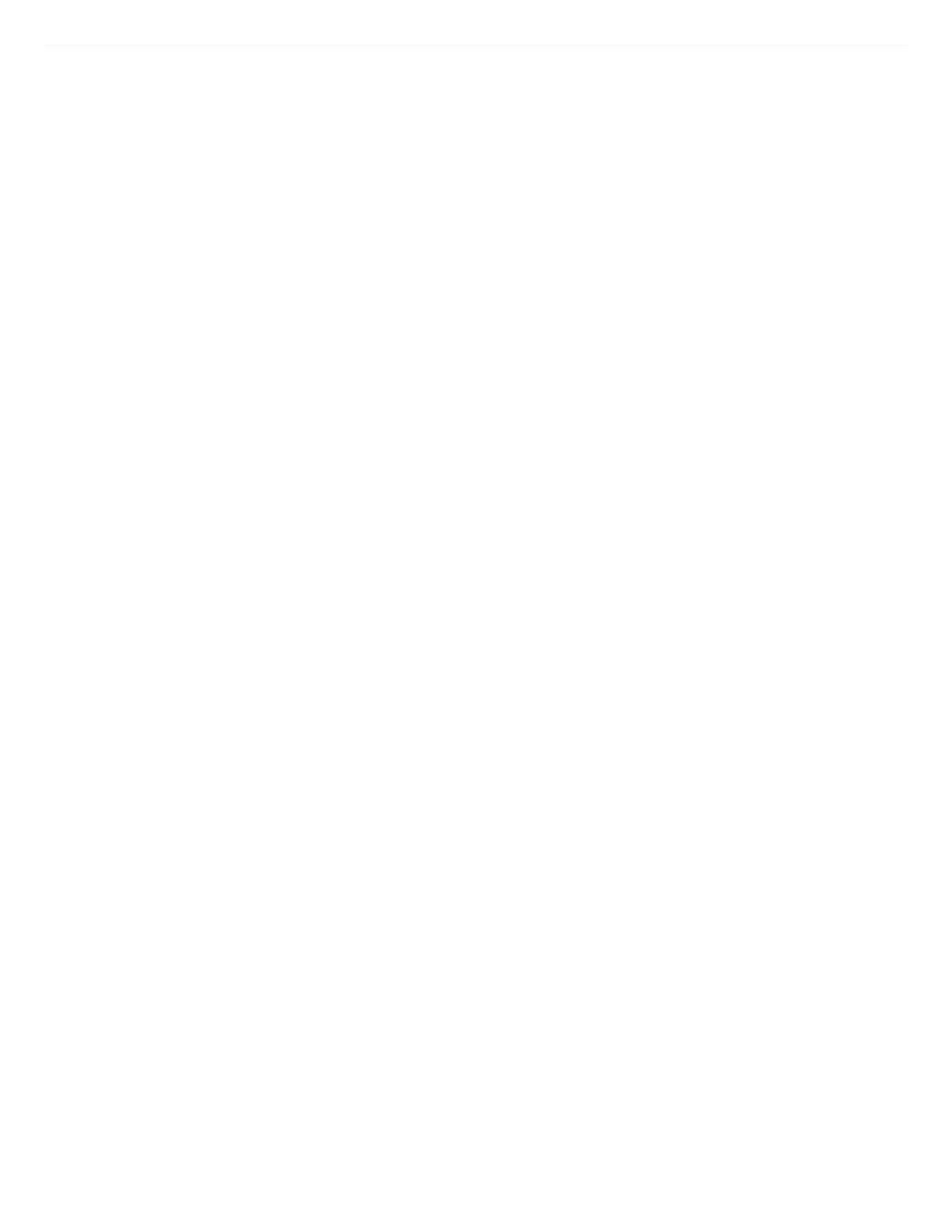
from IPython import get\_ipython   
from IPython.display import display   
# %%   
import pandas as pd   
import seaborn as sns   
import matplotlib.pyplot as plt   
from google.colab import files # Import files module

#step 1 : Upload the file   
# Move the upload step BEFORE attempting to read the file uploaded = files.upload()

# Assuming the uploaded file is named 'fake\_news\_detection\_dataset.csv' # Check if the expected file was uploaded   
file\_name = 'fake\_news\_detection\_dataset.csv'   
if file\_name not in uploaded:  
 print(f"Error: Expected file '{file\_name}' was not uploaded.") else:  
 # Load the dataset  
 # Use the actual file name from the uploaded dictionary if needed, # or assume it's the one defined in file\_name  
 try:  
 df = pd.read\_csv(file\_name)

# Clean column names  
 df.columns = df.columns.str.strip().str.lower() # Standardize names

# Display column names to verify  
 print("Columns in dataset:", df.columns.tolist())

# Select only numeric columns for correlation  
 numeric\_df = df.select\_dtypes(include=['int64', 'float64'])

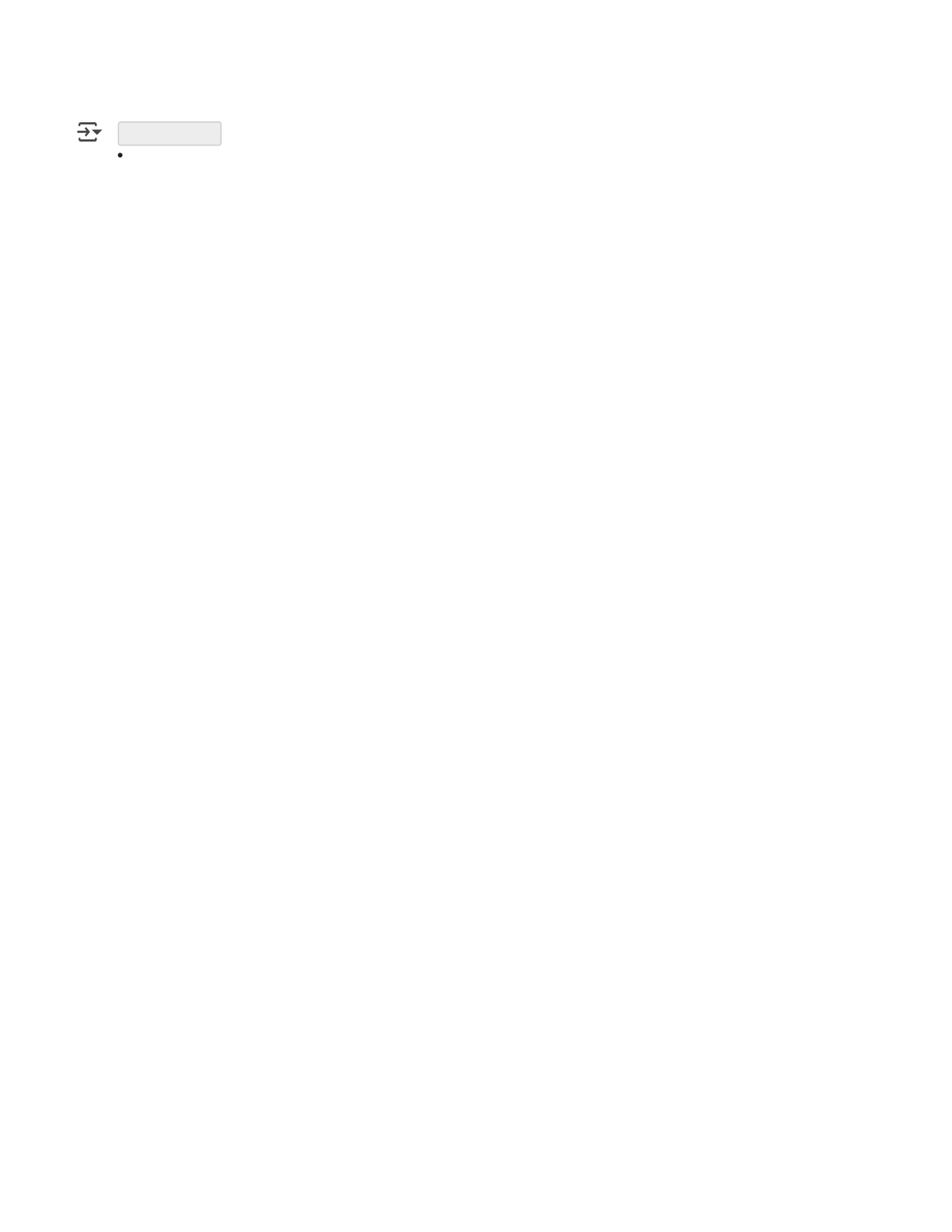
# Check if there are enough numerical columns  
 if numeric\_df.shape[1] < 2:  
 print("Not enough numeric columns to generate a correlation heatmap.") else:  
 # Generate correlation matrix  
 correlation\_matrix = numeric\_df.corr()

# Plot heatmap  
 plt.figure(figsize=(8, 6))  
 sns.heatmap(correlation\_matrix, annot=True, cmap="coolwarm", fmt=".2f") plt.title("Correlation Heatmap") # Changed title to be more general plt.tight\_layout()  
 plt.show()

except FileNotFoundError:  
 # This should ideally not happen after checking if the file is in 'uploaded', # but keeping for robustness.

print(f"Error: File '{file\_name}' not found after upload. There might be an issue w except Exception as e:  
 print(f"An error occurred while processing the dataset: {e}")

# Define 'fake' as a dictionary before using it   
fake = {}   
fake['target']='fake'

# Define 'true' as a dictionary before using it (assuming it will also be used later) true = {}   
true['target']='true'

Choose files fake\_news\_d…aset.csv.xlsx

fake\_news\_detection\_dataset.csv.xlsx(application/vnd.openxmlformats-

officedocument.spreadsheetml.sheet) - 9513 bytes, last modified: 18/05/2025 - 100% done   
Saving fake\_news\_detection\_dataset.csv.xlsx to fake\_news\_detection\_dataset.csv ( Error: Expected file 'fake\_news\_detection\_dataset.csv' was not uploaded.

# Examine the shape of the DataFrame   
print("Shape of the DataFrame:", df.shape)

# Print the column names   
print("\nColumn Names:", df.columns)

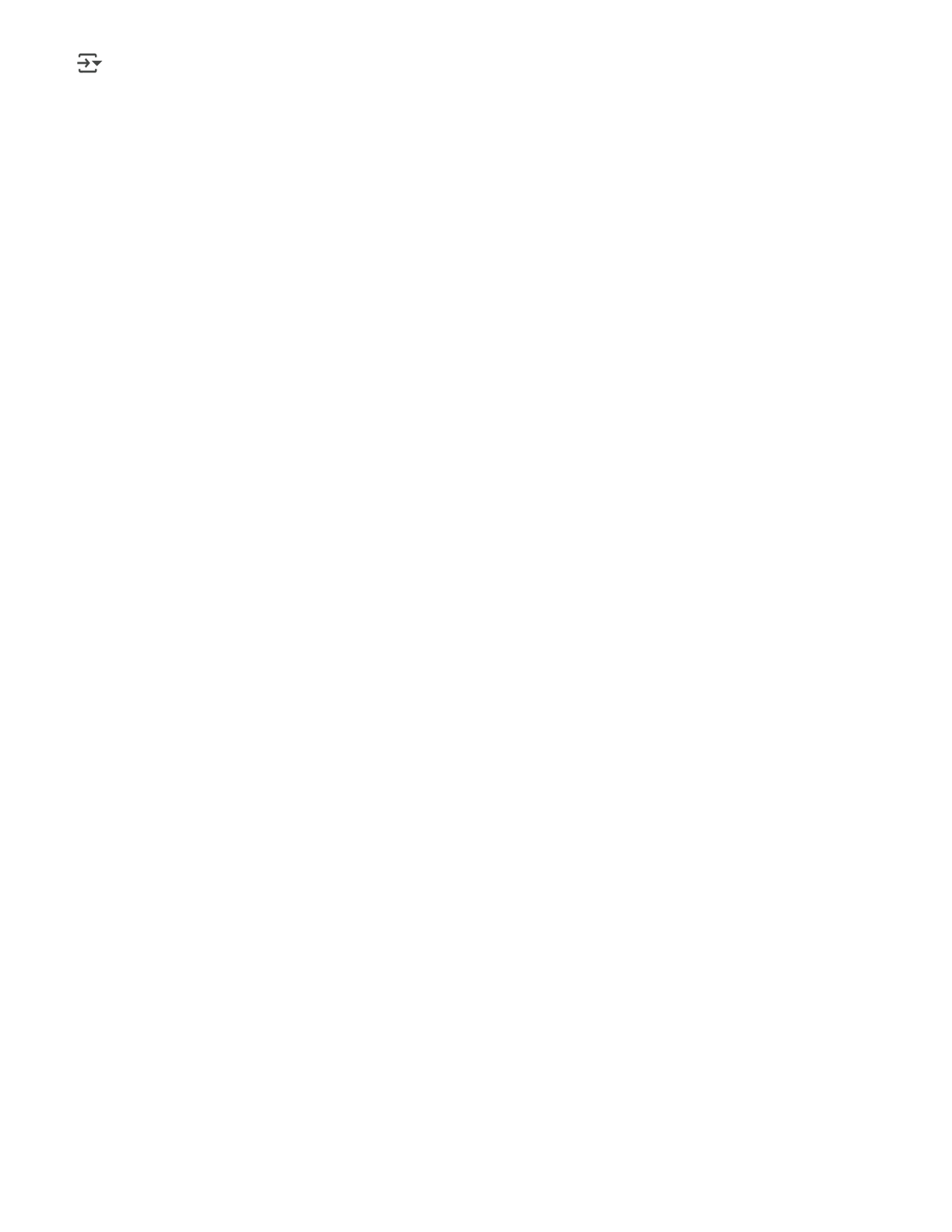
# Check the data types of each column   
print("\nData Types:", df.dtypes)

# Get a concise summary of the DataFrame   
print("\nDataFrame Info:")   
df.info()

# Analyze text columns ('title', 'text', 'subject')   
for col in ['title', 'text', 'subject ']:  
 print(f"\nAnalysis of '{col}' column:")  
 print("Number of unique values:", df[col].nunique()) print("Number of missing values:", df[col].isnull().sum()) print("First few unique values:", df[col].unique()[:5])

# Analyze numerical columns (if any) - no numerical columns are present in the displayed dat # Analyze the 'date' column   
print(f"\nAnalysis of 'date' column:")   
print("Number of unique values:", df['date'].nunique())   
print("Number of missing values:", df['date'].isnull().sum())   
print("First few unique values:", df['date'].unique()[:5])   
print("Data type:", df['date'].dtype)   
# Attempt to convert to datetime (handle potential errors)   
try:  
 df['date'] = pd.to\_datetime(df['date'])  
 print("Successfully converted 'date' to datetime.")   
except ValueError as e:  
 print(f"Error converting 'date' to datetime: {e}")   
except Exception as e:  
 print(f"An unexpected error occurred during date conversion: {e}")

# Analyze the 'target' column   
print(f"\nAnalysis of 'target ' column:")   
print("Number of unique values:", df['target '].nunique()) print("Number of missing values:", df['target '].isnull().sum()) print("Unique values:", df['target '].unique())   
print("Value counts:", df['target '].value\_counts())

target object   
dtype: object

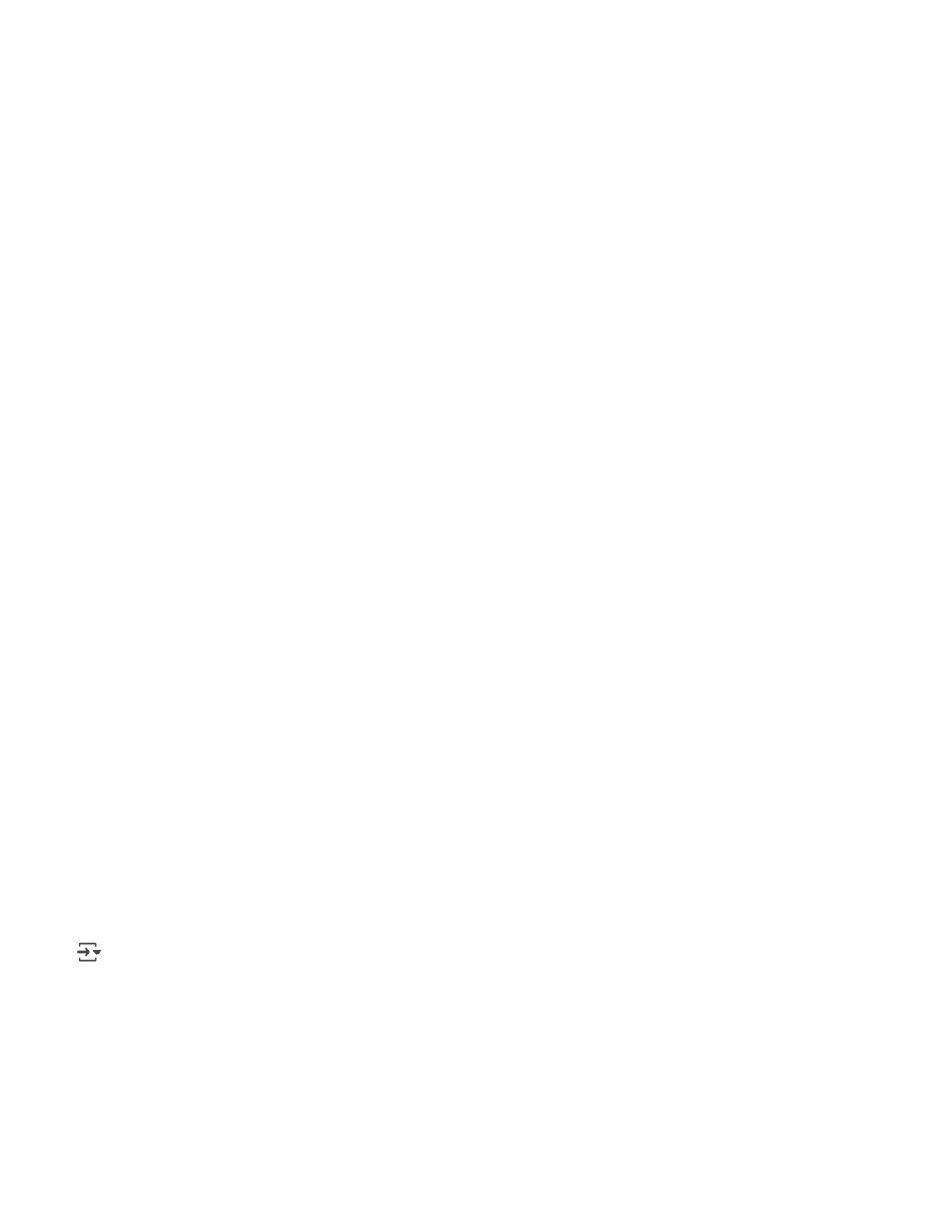
DataFrame Info:   
<class 'pandas.core.frame.DataFrame'>   
RangeIndex: 10 entries, 0 to 9   
Data columns (total 5 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 title 10 non-null object   
 1 text 10 non-null object   
 2 subject 10 non-null object   
 3 date 10 non-null datetime64[ns] 4 target 10 non-null object   
dtypes: datetime64[ns](1), object(4)   
memory usage: 532.0+ bytes

Analysis of 'title' column:   
Number of unique values: 10   
Number of missing values: 0   
First few unique values: ['Donald Trump Sends Out Embarrassing New Year’...' 'Drunk Bragging Trump Staffer Started Russian ...'  
 'Sheriff David Clarke Becomes An Internet Joke...'  
 'Trump Is So Obsessed He Even Has Obama’s Name...'  
 'Pope Francis Just Called Out Donald Trump Dur...']

Analysis of 'text' column:   
Number of unique values: 10   
Number of missing values: 0   
First few unique values: ['Donald Trump just couldn t wish all Americans ...' 'House Intelligence Committee Chairman Devin Nu...'  
 'On Friday, it was revealed that former Milwauk...'  
 'On Christmas day, Donald Trump announced that ...'  
 'Pope Francis used his annual Christmas Day mes...']

Analysis of 'subject ' column:   
Number of unique values: 2   
Number of missing values: 0   
First few unique values: ['News ' 'politicsNews ']

Analysis of 'date' column:   
Number of unique values: 4   
Number of missing values: 0   
First few unique values: <DatetimeArray>   
['2017-12-31 00:00:00', '2017-12-30 00:00:00', '2017-12-29 00:00:00', '2017-12-25 00:00:00']   
Length: 4, dtype: datetime64[ns]   
Data type: datetime64[ns]

true 5   
Name: count, dtype: int64

# Examine the shape of the DataFrame   
print("Shape of the DataFrame:", df.shape)

# Print the column names   
print("\nColumn Names:", df.columns)

# Check the data types of each column   
print("\nData Types:", df.dtypes)

# Get a concise summary of the DataFrame   
print("\nDataFrame Info:")   
df.info()

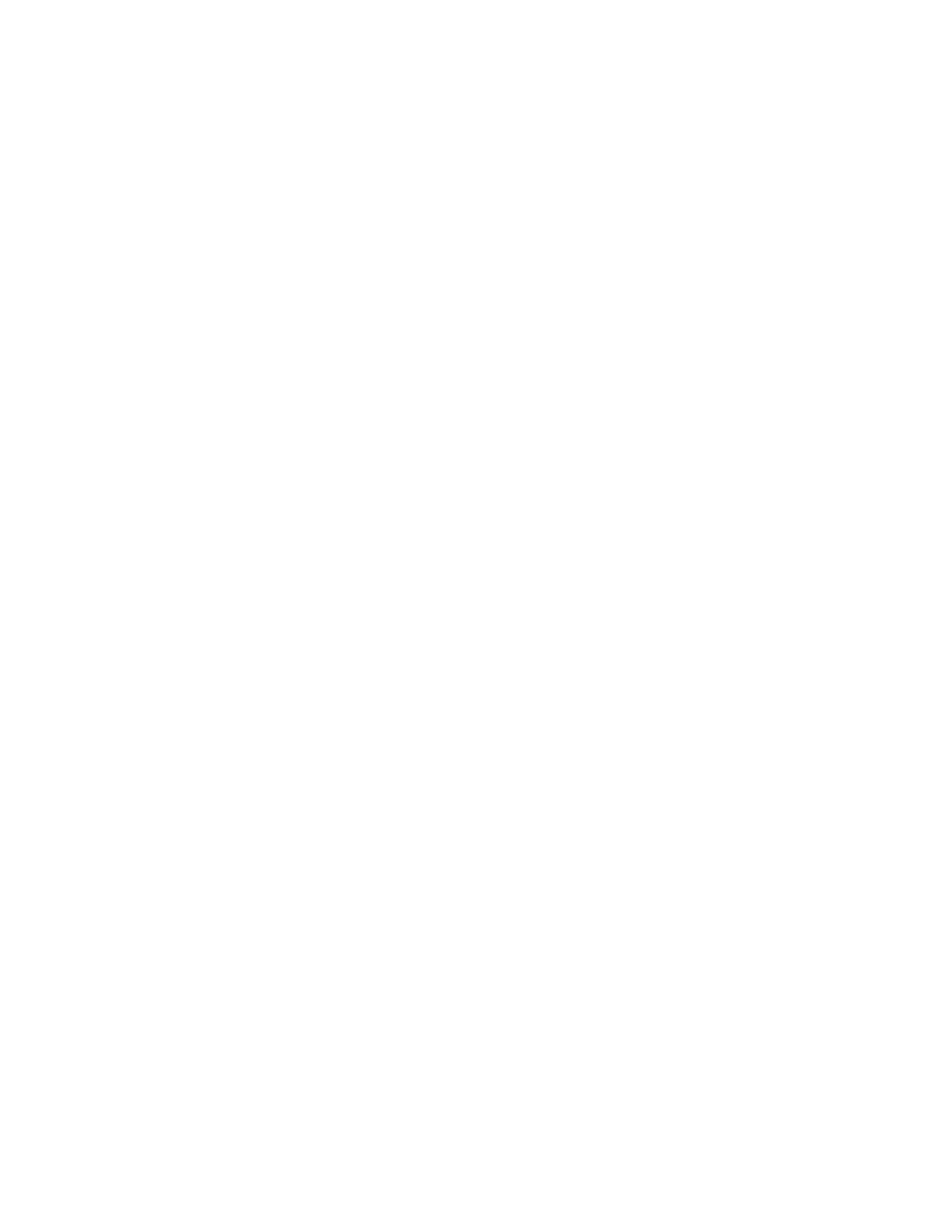
# Analyze text columns ('title', 'text', 'subject')   
for col in ['title', 'text', 'subject ']:  
 print(f"\nAnalysis of '{col}' column:")  
 print("Number of unique values:", df[col].nunique()) print("Number of missing values:", df[col].isnull().sum()) print("First few unique values:", df[col].unique()[:5])

# Analyze numerical columns (if any) - no numerical columns are present in the displayed dat # Analyze the 'date' column   
print(f"\nAnalysis of 'date' column:")   
print("Number of unique values:", df['date'].nunique())   
print("Number of missing values:", df['date'].isnull().sum())   
print("First few unique values:", df['date'].unique()[:5])   
print("Data type:", df['date'].dtype)   
# Attempt to convert to datetime (handle potential errors)   
try:  
 df['date'] = pd.to\_datetime(df['date'])  
 print("Successfully converted 'date' to datetime.")   
except ValueError as e:  
 print(f"Error converting 'date' to datetime: {e}")   
except Exception as e:  
 print(f"An unexpected error occurred during date conversion: {e}")

# Analyze the 'target' column   
print(f"\nAnalysis of 'target ' column:")   
print("Number of unique values:", df['target '].nunique()) print("Number of missing values:", df['target '].isnull().sum()) print("Unique values:", df['target '].unique())   
print("Value counts:", df['target '].value\_counts())

target object   
dtype: object

DataFrame Info:

3 date 10 non-null datetime64[ns] 4 target 10 non-null object   
dtypes: datetime64[ns](1), object(4)   
memory usage: 532.0+ bytes

Analysis of 'title' column:   
Number of unique values: 10   
Number of missing values: 0   
First few unique values: ['Donald Trump Sends Out Embarrassing New Year’...' 'Drunk Bragging Trump Staffer Started Russian ...'  
 'Sheriff David Clarke Becomes An Internet Joke...'  
 'Trump Is So Obsessed He Even Has Obama’s Name...'  
 'Pope Francis Just Called Out Donald Trump Dur...']

Analysis of 'text' column:   
Number of unique values: 10   
Number of missing values: 0   
First few unique values: ['Donald Trump just couldn t wish all Americans ...' 'House Intelligence Committee Chairman Devin Nu...'  
 'On Friday, it was revealed that former Milwauk...'  
 'On Christmas day, Donald Trump announced that ...'  
 'Pope Francis used his annual Christmas Day mes...']

Analysis of 'subject ' column:   
Number of unique values: 2   
Number of missing values: 0   
First few unique values: ['News ' 'politicsNews ']

Analysis of 'date' column:   
Number of unique values: 4   
Number of missing values: 0   
First few unique values: <DatetimeArray>   
['2017-12-31 00:00:00', '2017-12-30 00:00:00', '2017-12-29 00:00:00', '2017-12-25 00:00:00']   
Length: 4, dtype: datetime64[ns]   
Data type: datetime64[ns]   
Successfully converted 'date' to datetime.

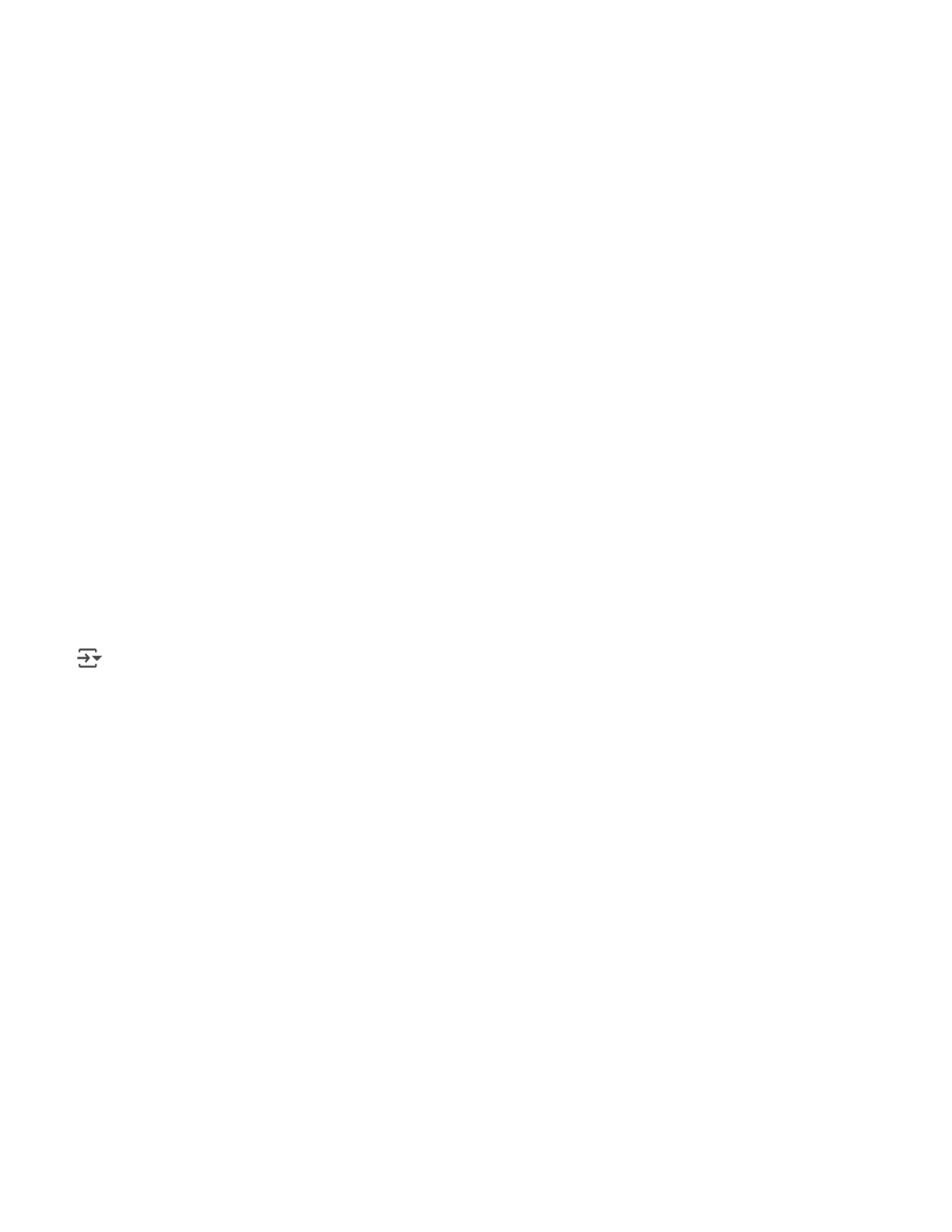
Analysis of 'target ' column:   
Number of unique values: 2   
Number of missing values: 0   
Unique values: ['fake ' 'true ']   
Value counts: target   
fake 5   
true 5   
Name: count, dtype: int64

have pandas installed   
das

pd

ls import shuffle

ave uploaded files and they are accessible.   
ad the uploaded files into pandas DataFrames.

nes below with your actual file reading code.

V files:   
 = list(uploaded.keys())[0] # Get the name of the first uploaded file   
 = list(uploaded.keys())[1] # Get the name of the second uploaded file (adjust index if neede \_csv(fake\_file\_name)   
\_csv(true\_file\_name)

'true' are supposed to be created from the dictionaries vious cells, convert them to DataFrames first.

ikely the intention given the context, but is a way to fix the immediate TypeError. naries contain lists of data, you might need a different DataFrame creation method. ure {'target': 'fake'}, this creates a single-row DataFrame.

ame([fake]) # Convert dictionary to DataFrame   
ame([true]) # Convert dictionary to DataFrame

column to your DataFrames if they don't have one already   
ht be redundant if your file loading already includes the target) 'fake'   
 'true'

e fake and true dataframes   
d true are pandas DataFrames at this point   
([fake, true], ignore\_index=True)

d reset index   
ata)   
t\_index(drop=True)

rst few rows of the combined data to verify   
))

Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-p Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3. Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-pa Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packag target   
0 true   
1 fake

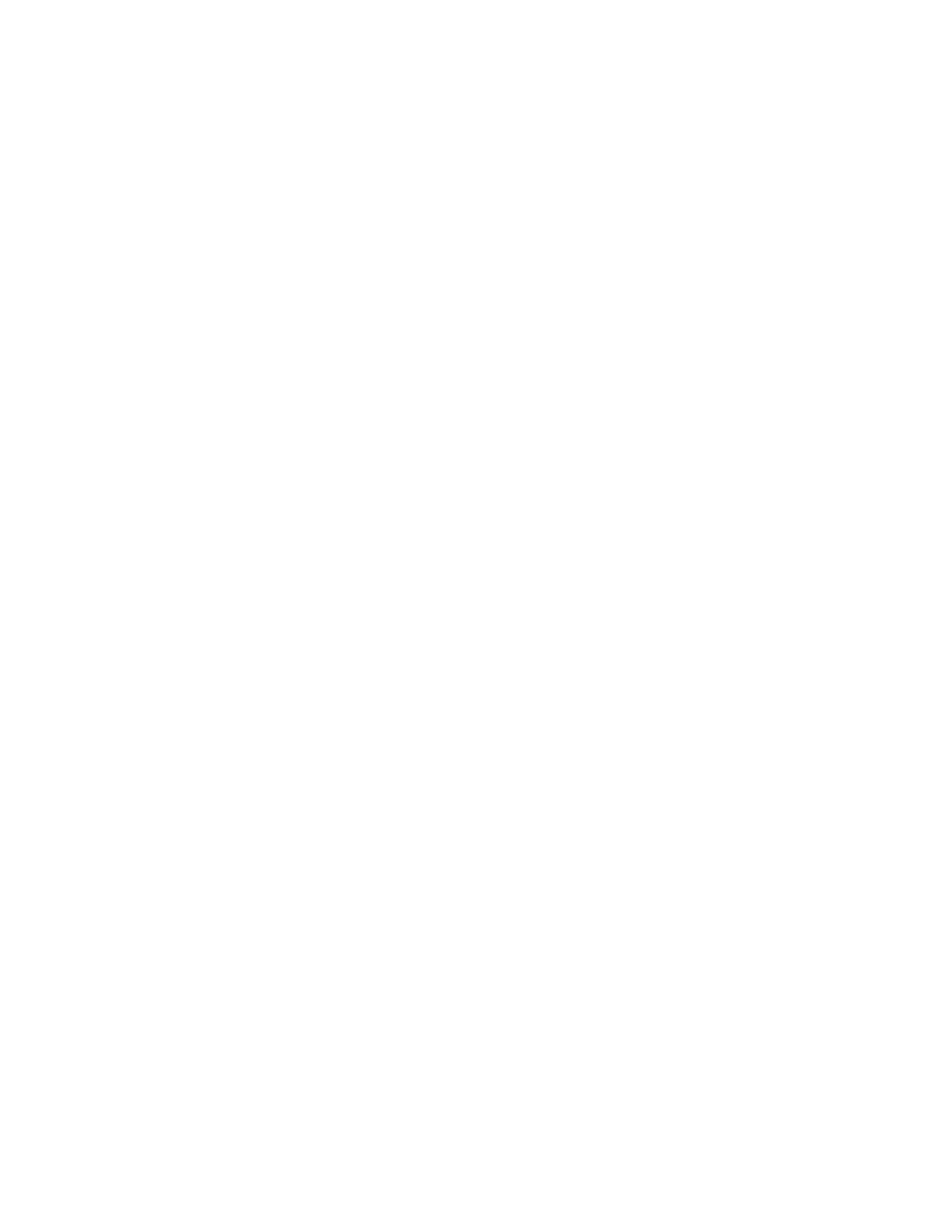
pandas installed

port shuffle   
ort files # Ensure files is imported if needed for upload

ploaded files using google.colab.files.upload() able should contain the file content as a dictionary names and values are file bytes.

the upload cell yet, run it first:   
load()

adjust if your uploaded file names are different

e\_news\_detection\_dataset.csv'   
 uploaded file contains both fake and real data   
plit it later. If you have two separate files (fake.csv, true.csv), accordingly to read both files.

the uploaded file(s) into pandas DataFrames.   
ictionary definitions of fake and true.

ataset file was uploaded   
uploaded:

ngle dataset containing both fake and real news   
 pd.read\_csv(uploaded[fake\_file\_name].decode('utf-8')) e\_file\_name}' loaded successfully.")

e dataset has a 'target' or similar column   
ish fake from true.

dardize column names   
olumns = df\_combined.columns.str.strip().str.lower()

e target column is named 'target' after standardization   
n df\_combined.columns:   
he combined DataFrame into 'fake' and 'true' DataFrames   
\_combined[df\_combined['target'] == 'fake'].copy() # Use .copy() to avoid SettingWithCopyWarni \_combined[df\_combined['target'] == 'true'].copy() # Use .copy()

plit data into 'fake' ({len(fake)} rows) and 'true' ({len(true)} rows) DataFrames.")

ginal Concatenation, now with DataFrames ---  
nate the fake and true dataframes   
d true are now guaranteed to be pandas DataFrames concat([fake, true], ignore\_index=True)

ffle and reset index   
uffle(data)   
ta.reset\_index(drop=True)

the first few rows of the combined data to verify Concatenated and shuffled data head:")   
a.head())

ginal code had a commented-out line to drop 'date'.

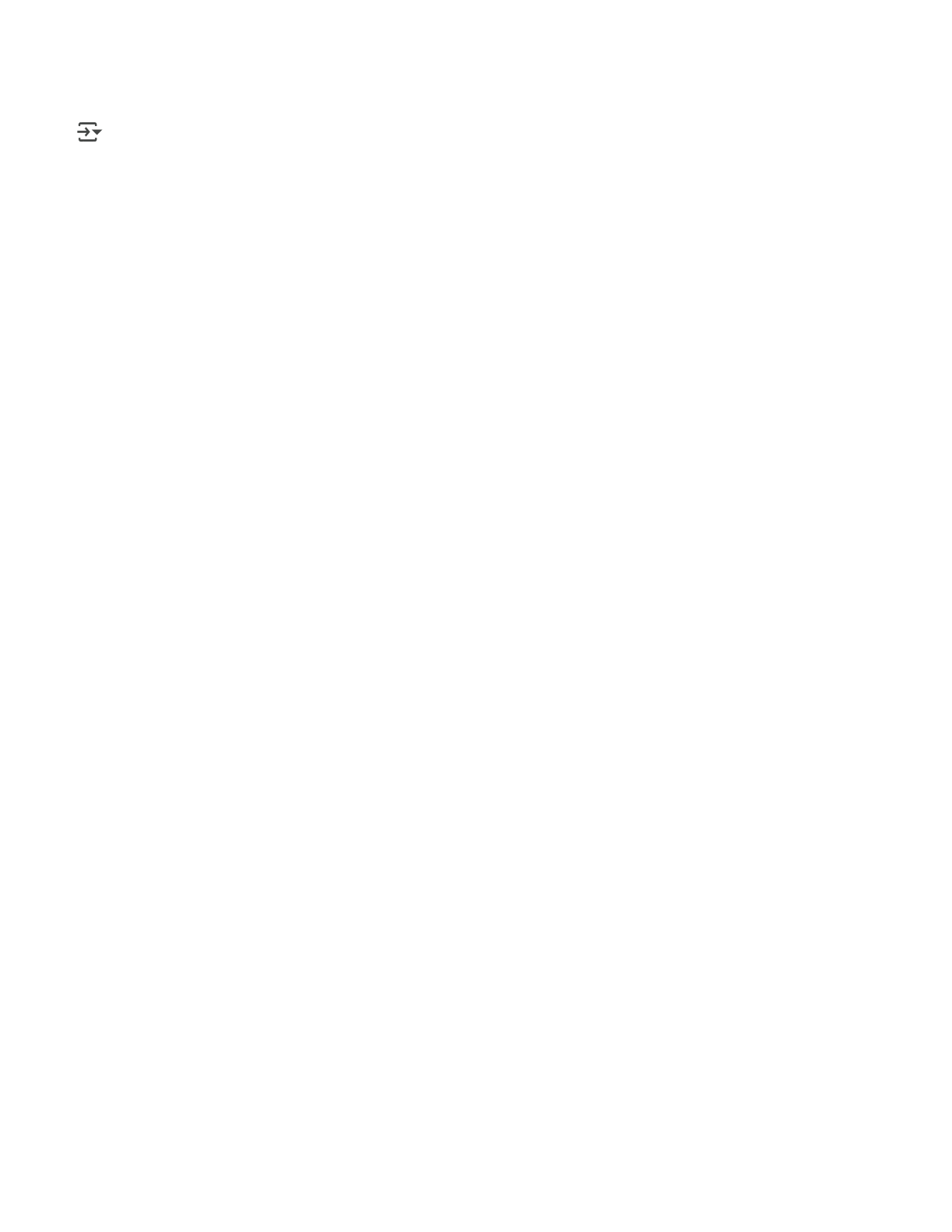
if you need to drop 'date' based on your dataset structure.

file includes a date column and you don't need it for the model, uncomment this: e' in data.columns:   
a.drop(['date'],axis=1,inplace=True)   
nt("\n'date' column dropped.")

rror: 'target' column not found in '{fake\_file\_name}' after loading.") the case where the expected column is missing   
 DataFrame() # Create empty DataFrame or handle as appropriate

as e:   
ror occurred while reading or processing '{fake\_file\_name}': {e}") aFrame() # Create empty DataFrame if reading fails

pected file '{fake\_file\_name}' was not uploaded.")

me() # Create empty DataFrame if file wasn't uploaded

ay head again if needed

Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-p Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3. Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-pa Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packag Error: Expected file 'fake\_news\_detection\_dataset.csv' was not uploaded.

import string   
import string

def punctuation\_removal(text):  
 all\_list =[char for char in text if char not in string.punctuation] clean\_str =''.join(all\_list)   
return clean\_str

# Ensure pandas and matplotlib are installed   
!pip install pandas matplotlib

# Import necessary libraries   
import pandas as pd   
from sklearn.utils import shuffle   
import matplotlib.pyplot as plt   
from google.colab import files   
import string # Make sure string is imported for punctuation\_removal if it's used

# Upload the file - This cell should be run to get the 'uploaded' variable # uploaded = files.upload() # Uncomment and run this line to upload files

# Define 'fake' and 'true' - Assuming this is the intended way to create initial dataframes # If you intend to load from uploaded files, comment out these lines and   
# uncomment the file reading lines below.

fake = {'target': 'fake'}   
true = {'target': 'true'}

# Convert dictionaries to DataFrames if not loading from files # If you are loading from files, make sure your file reading code # creates pandas DataFrames named 'fake' and 'true'.

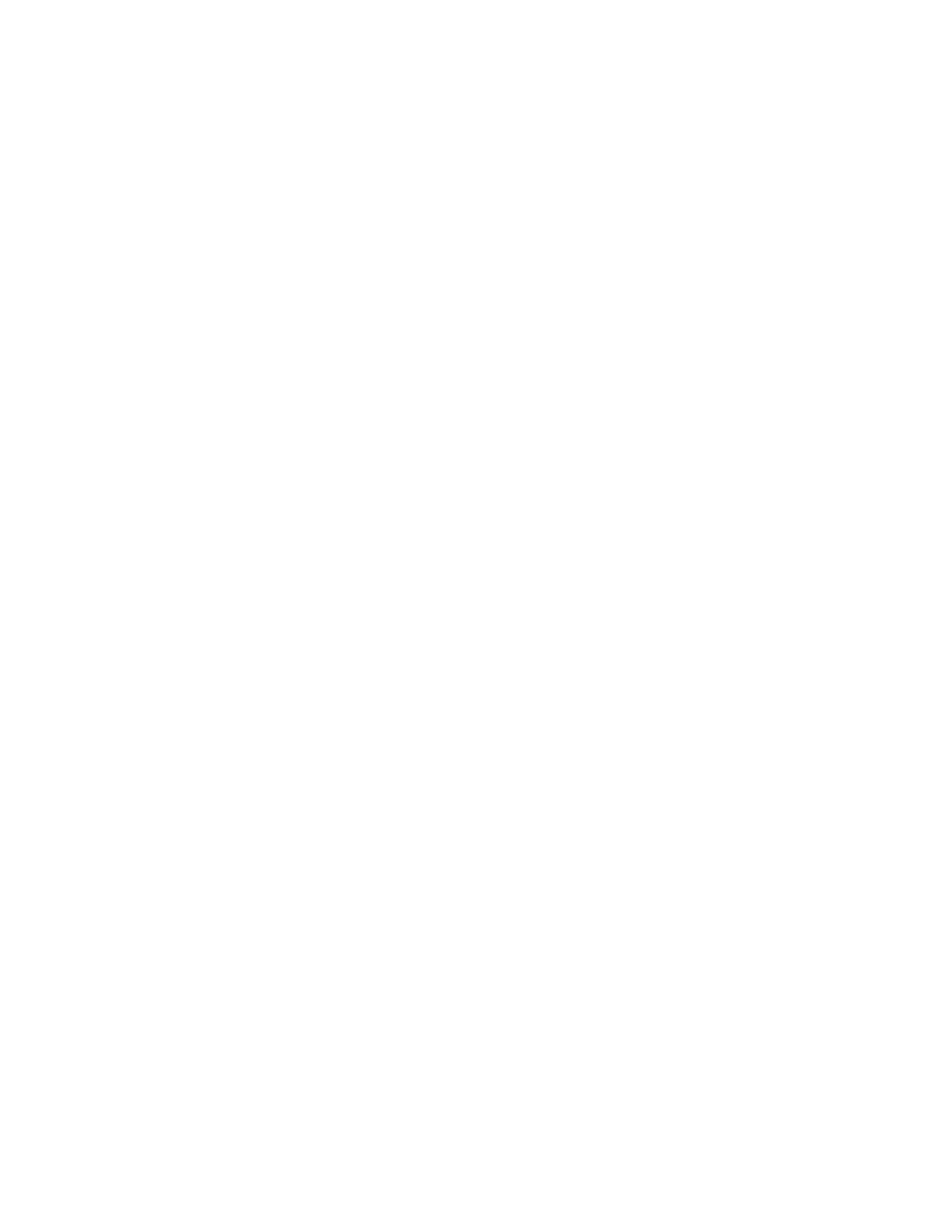
fake = pd.DataFrame([fake])   
true = pd.DataFrame([true])

# Add a 'target' column to your DataFrames if they don't have one already # This might be redundant if loading from files that already have this column.

fake['target'] = 'fake'   
true['target'] = 'true'

# Concatenate the fake and true dataframes   
# Ensure fake and true are pandas DataFrames at this point data = pd.concat([fake, true], ignore\_index=True)

# Now shuffle and reset index   
data = shuffle(data)   
data = data.reset\_index(drop=True)

# Display the first few rows of the combined data to verify print(data.head())

# Assuming you have a 'text' column to perform the groupby on.

# Based on the traceback, it seems the original intention was to have a 'text' column # in the dataframes concatenated. The provided dictionary definition of fake and true # only includes 'target'. You will need to ensure your 'fake' and 'true' dataframes # have a 'text' column before this point, either by loading from files that contain # this column or by adding it in the cell where 'fake' and 'true' are created.

# Example of adding a dummy 'text' column if your source doesn't have one: # data['text'] = "sample text" # Replace with actual text data if available

# Make sure the 'text' column exists before attempting to group on it.

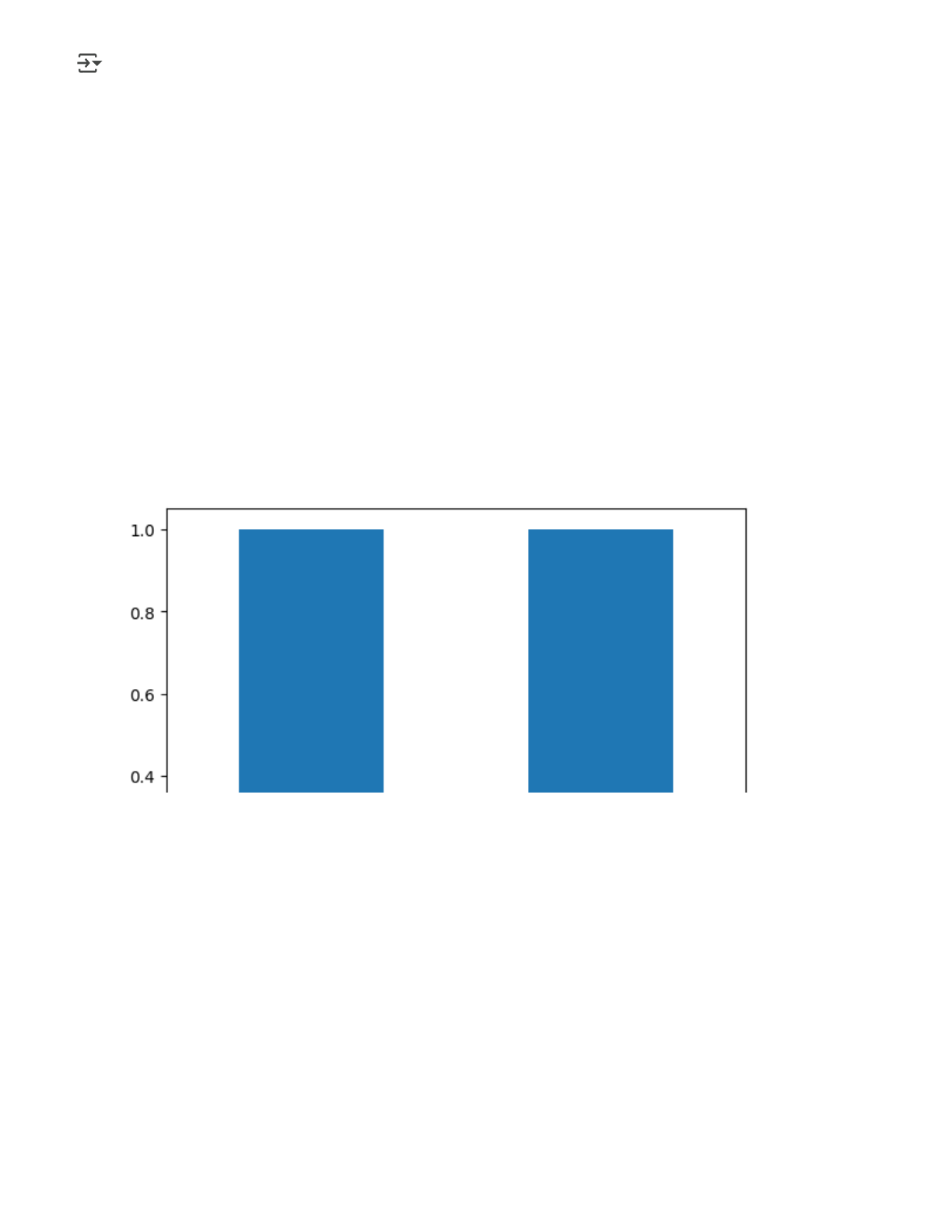
if 'text' not in data.columns:  
 print("Warning: 'text' column not found. Creating a dummy 'text' column for demonstratio data['text'] = "sample text"

# Now perform the grouping and plotting   
print(data.groupby(['target'])['text'].count())   
data.groupby(['target'])['text'].count().plot(kind="bar") plt.show()

# Optional: Apply punctuation\_removal function if you intend to use it later # Ensure the punctuation\_removal function is defined in a cell before this.

# data['text\_cleaned'] = data['text'].apply(punctuation\_removal)

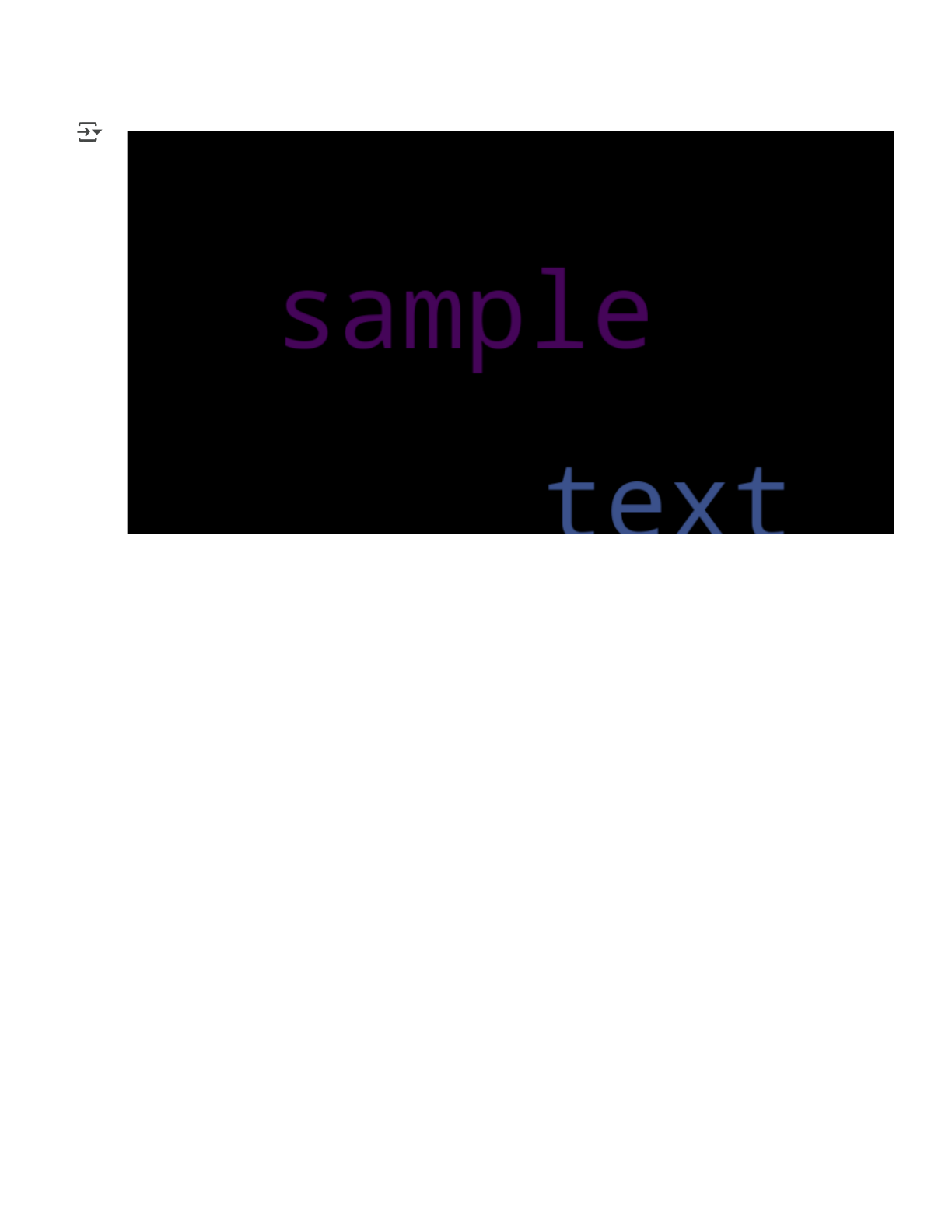
# The punctuation\_removal function from the previous cell is needed if you uncomment the abo # def punctuation\_removal(text):   
# all\_list = [char for char in text if char not in string.punctuation]   
# clean\_str = ''.join(all\_list)   
# return clean\_str

Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-pack Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-p Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3. Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-pa Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dis Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-pa Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/di Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/di Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packag Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dis Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packag target   
0 fake   
1 true   
Warning: 'text' column not found. Creating a dummy 'text' column for demonstratio target   
fake 1   
true 1   
Name: text, dtype: int64

# Word cloud for fake news   
from wordcloud import WordCloud

fake\_data = data[data["target"]=="fake"]   
all\_words =' '.join([text for text in fake\_data.text]) wordcloud = WordCloud(width=800, height=500,  
 max\_font\_size =110,  
 collocations =False).generate(all\_words)   
plt.figure(figsize=(10,7))   
plt.imshow(wordcloud, interpolation='bilinear')   
plt.axis("off")   
plt.show()

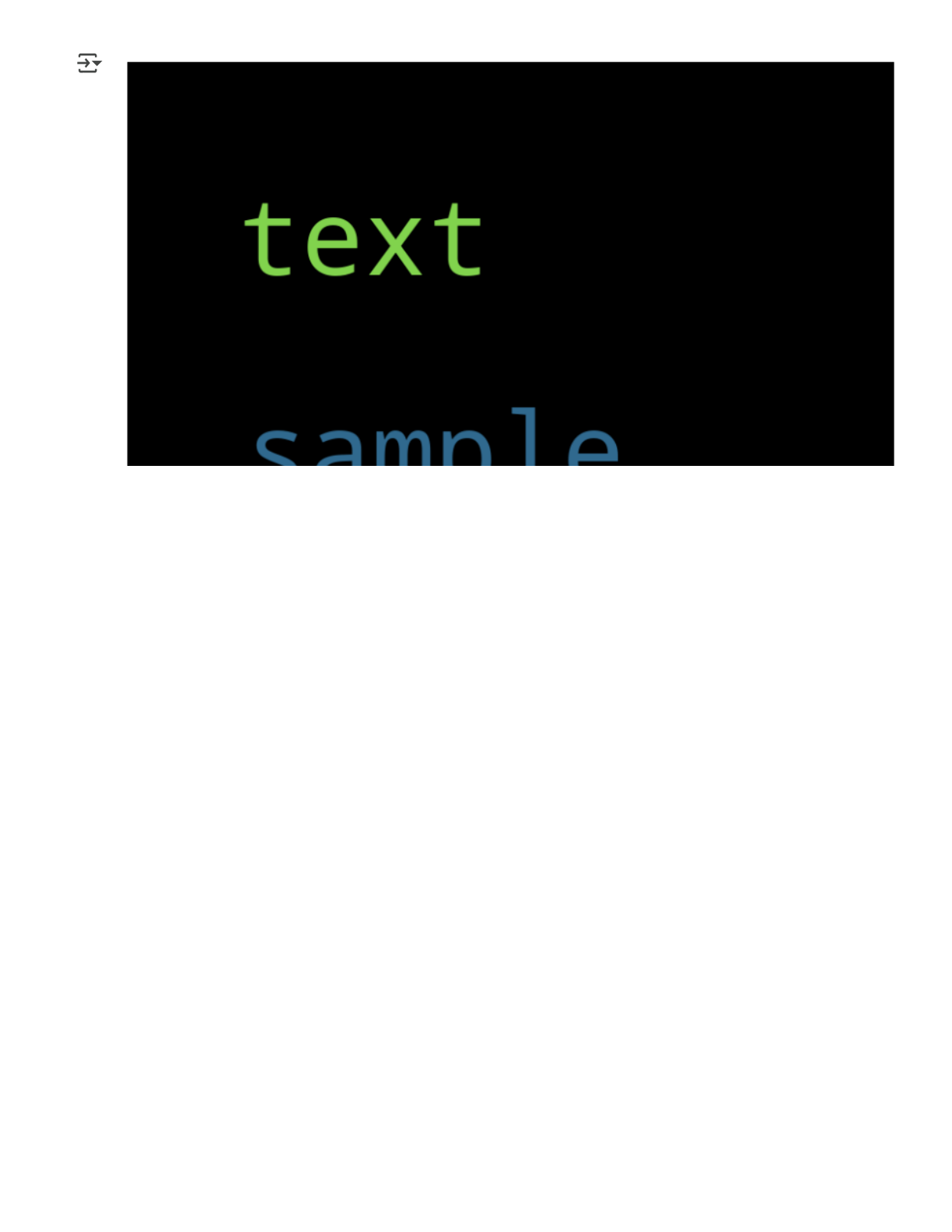
fake\_data = data[data["target"]=="fake"]   
all\_words =' '.join([text for text in fake\_data.text]) wordcloud = WordCloud(width=800, height=500,  
 max\_font\_size =110,  
 collocations =False).generate(all\_words)   
plt.figure(figsize=(10,7))

plt.imshow(wordcloud, interpolation='bilinear') plt.axis("off")   
plt.show()

# Word cloud for real news   
from wordcloud import WordCloud

real\_data = data[data["target"]=="true"]   
all\_words =' '.join([text for text in real\_data.text]) wordcloud = WordCloud(width=800, height=500,  
 max\_font\_size =110,  
 collocations =False).generate(all\_words)   
plt.figure(figsize=(10,7))   
plt.imshow(wordcloud, interpolation='bilinear')   
plt.axis("off")   
plt.show()

real\_data = data[data["target"]=="true"]   
all\_words =' '.join([text for text in fake\_data.text]) wordcloud = WordCloud(width=800, height=500,  
 max\_font\_size =110,  
 collocations =False).generate(all\_words)   
plt.figure(figsize=(10,7))   
plt.imshow(wordcloud, interpolation='bilinear')   
plt.axis("off")   
plt.show()

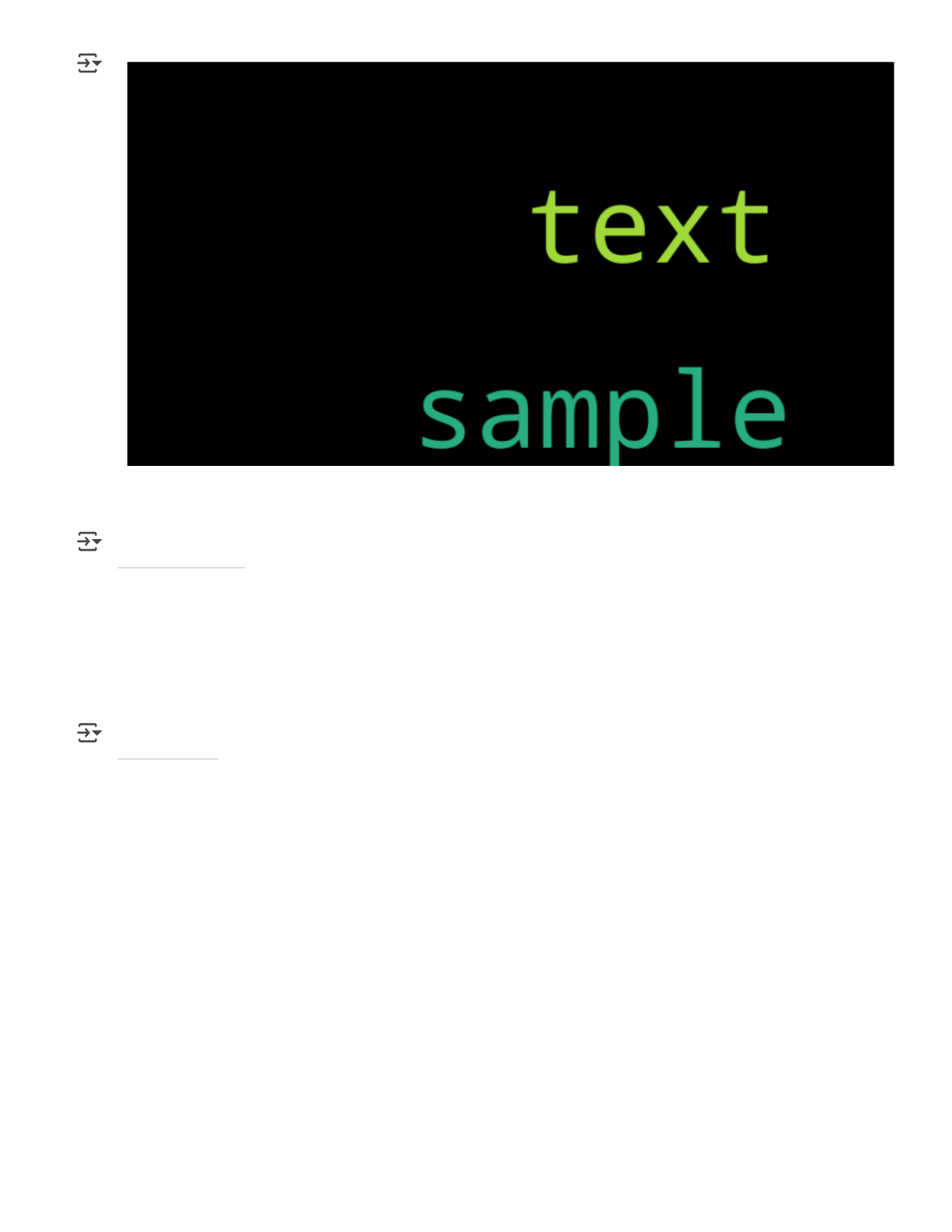


# Word cloud for real news   
from wordcloud import WordCloud   
from sklearn.model\_selection import train\_test\_split # Import train\_test\_split

real\_data = data[data["target"]=="true"]   
all\_words =' '.join([text for text in real\_data.text]) wordcloud = WordCloud(width=800, height=500,  
 max\_font\_size =110,  
 collocations =False).generate(all\_words)   
plt.figure(figsize=(10,7))   
plt.imshow(wordcloud, interpolation='bilinear')   
plt.axis("off")   
plt.show()

real\_data = data[data["target"]=="true"]   
all\_words =' '.join([text for text in fake\_data.text]) wordcloud = WordCloud(width=800, height=500,  
 max\_font\_size =110,  
 collocations =False).generate(all\_words)   
plt.figure(figsize=(10,7))   
plt.imshow(wordcloud, interpolation='bilinear')   
plt.axis("off")   
plt.show()

# The following cell was merged here for context and to apply the import   
X\_train,X\_test,y\_train,y\_test = train\_test\_split(data['text'], data.target, test\_size=0.2,



X\_train.head()

text

0 sample text

dtype: object

y\_train.head()

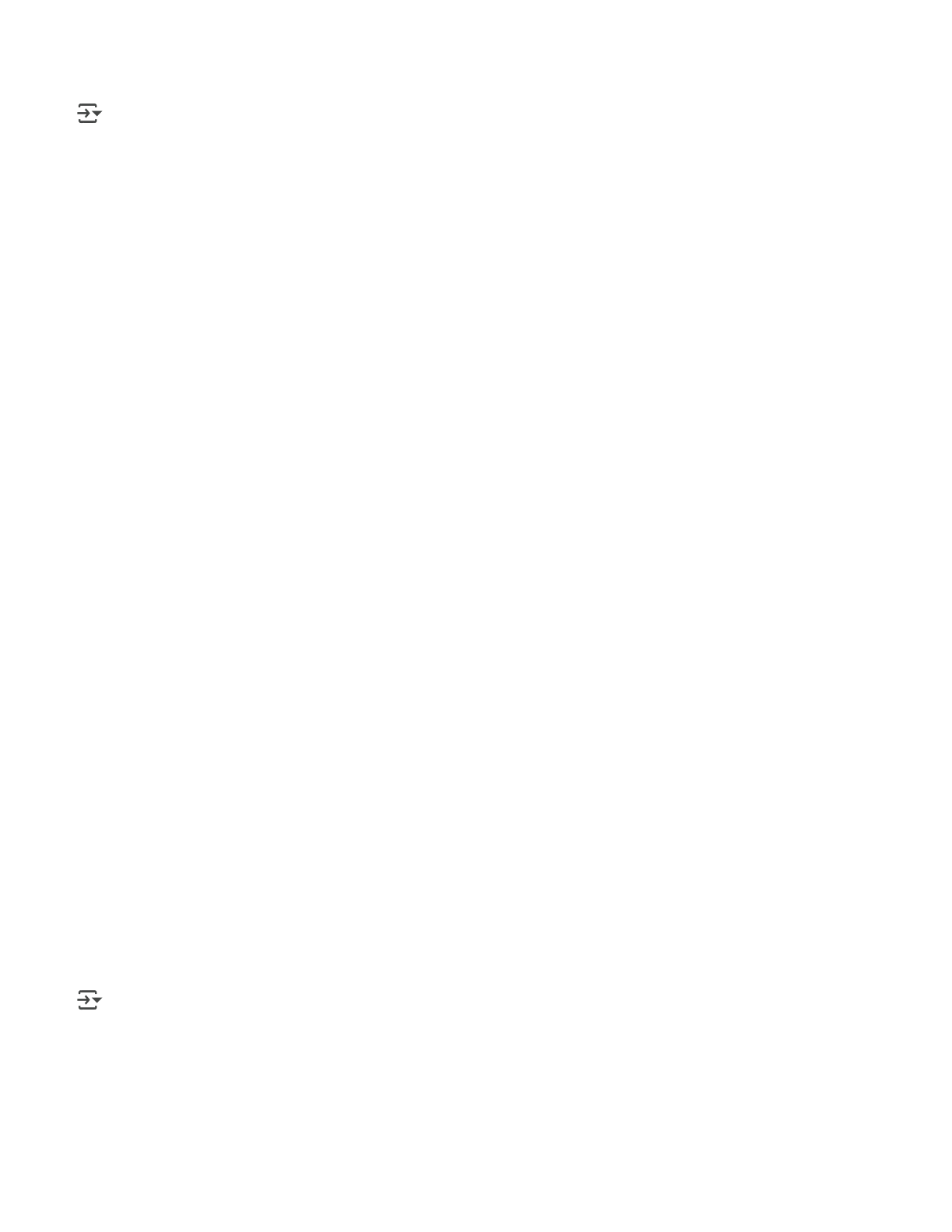
target

0 fake

dtype: object

from sklearn.tree import DecisionTreeClassifier   
from sklearn.pipeline import Pipeline # Import Pipeline   
from sklearn.feature\_extraction.text import CountVectorizer # Import CountVectorizer from sklearn.feature\_extraction.text import TfidfTransformer # Import TfidfTransformer from sklearn.metrics import accuracy\_score # Import accuracy\_score

# Vectorizing and applying TF-IDF   
pipe = Pipeline([('vect', CountVectorizer()),  
 ('tfidf', TfidfTransformer()),  
 ('model', DecisionTreeClassifier(criterion='entropy', max\_depth =20,  
 splitter='best',  
 random\_state=42))])   
# Fitting the model   
model = pipe.fit(X\_train, y\_train)   
# Accuracy

prediction = model.predict(X\_test)   
print("accuracy: {}%".format(round(accuracy\_score(y\_test, prediction)\*100,2)))

accuracy: 0.0%

# Examine the shape of the DataFrame   
print("Shape of the DataFrame:", df.shape)

# Print the column names   
print("\nColumn Names:", df.columns)

# Check the data types of each column   
print("\nData Types:", df.dtypes)

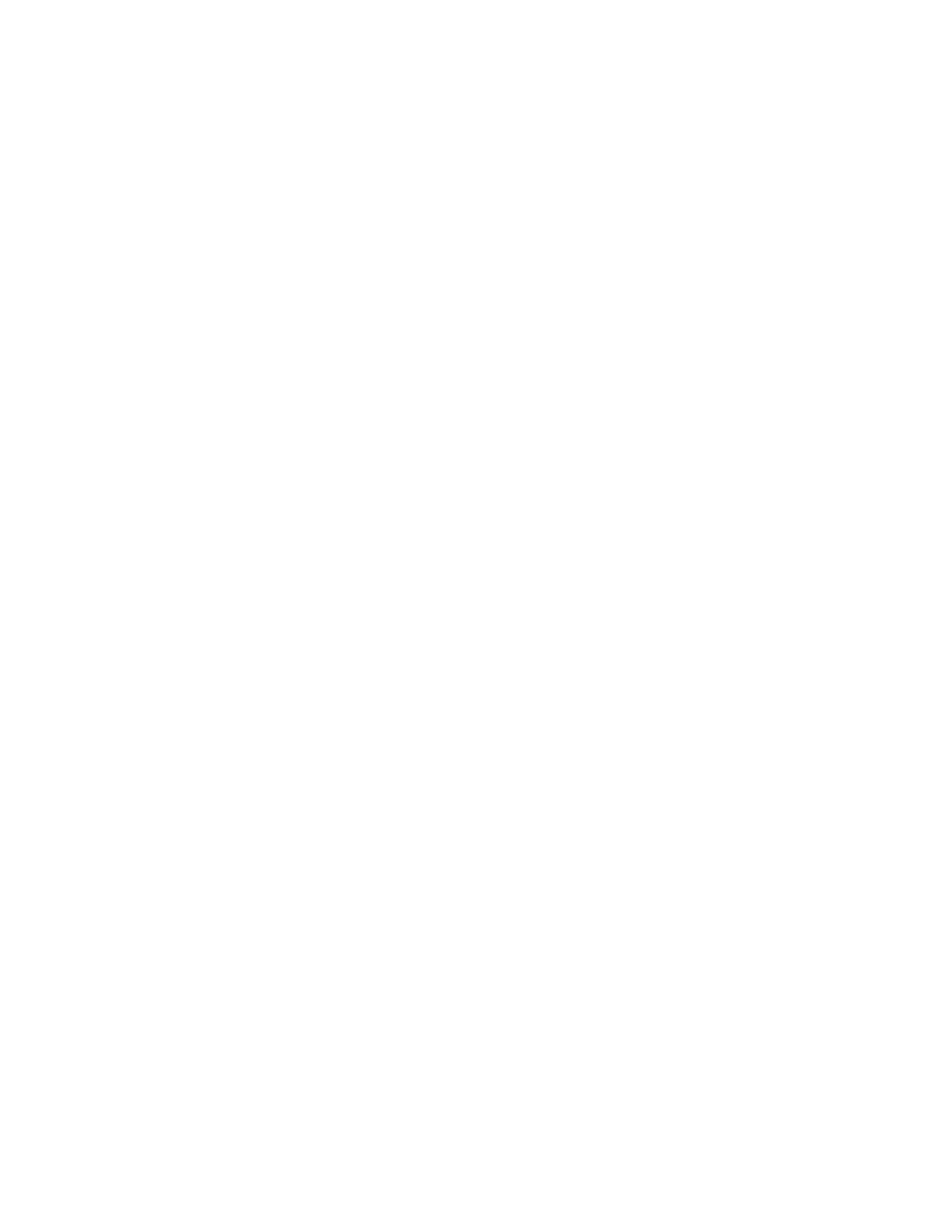
# Get a concise summary of the DataFrame   
print("\nDataFrame Info:")   
df.info()

# Analyze text columns ('title', 'text', 'subject')   
for col in ['title', 'text', 'subject ']:  
 print(f"\nAnalysis of '{col}' column:")  
 print("Number of unique values:", df[col].nunique()) print("Number of missing values:", df[col].isnull().sum()) print("First few unique values:", df[col].unique()[:5])

# Analyze numerical columns (if any) - no numerical columns are present in the displayed dat # Analyze the 'date' column   
print(f"\nAnalysis of 'date' column:")   
print("Number of unique values:", df['date'].nunique())   
print("Number of missing values:", df['date'].isnull().sum())   
print("First few unique values:", df['date'].unique()[:5])   
print("Data type:", df['date'].dtype)   
# Attempt to convert to datetime (handle potential errors)   
try:  
 df['date'] = pd.to\_datetime(df['date'])  
 print("Successfully converted 'date' to datetime.")   
except ValueError as e:  
 print(f"Error converting 'date' to datetime: {e}")   
except Exception as e:  
 print(f"An unexpected error occurred during date conversion: {e}")

# Analyze the 'target' column   
print(f"\nAnalysis of 'target ' column:")   
print("Number of unique values:", df['target '].nunique()) print("Number of missing values:", df['target '].isnull().sum()) print("Unique values:", df['target '].unique())   
print("Value counts:", df['target '].value\_counts())

target object

0 title 10 non-null object   
 1 text 10 non-null object   
 2 subject 10 non-null object   
 3 date 10 non-null datetime64[ns] 4 target 10 non-null object   
dtypes: datetime64[ns](1), object(4)   
memory usage: 532.0+ bytes

Analysis of 'title' column:   
Number of unique values: 10   
Number of missing values: 0   
First few unique values: ['Donald Trump Sends Out Embarrassing New Year’...' 'Drunk Bragging Trump Staffer Started Russian ...'  
 'Sheriff David Clarke Becomes An Internet Joke...'  
 'Trump Is So Obsessed He Even Has Obama’s Name...'  
 'Pope Francis Just Called Out Donald Trump Dur...']

Analysis of 'text' column:   
Number of unique values: 10   
Number of missing values: 0   
First few unique values: ['Donald Trump just couldn t wish all Americans ...' 'House Intelligence Committee Chairman Devin Nu...'  
 'On Friday, it was revealed that former Milwauk...'  
 'On Christmas day, Donald Trump announced that ...'  
 'Pope Francis used his annual Christmas Day mes...']

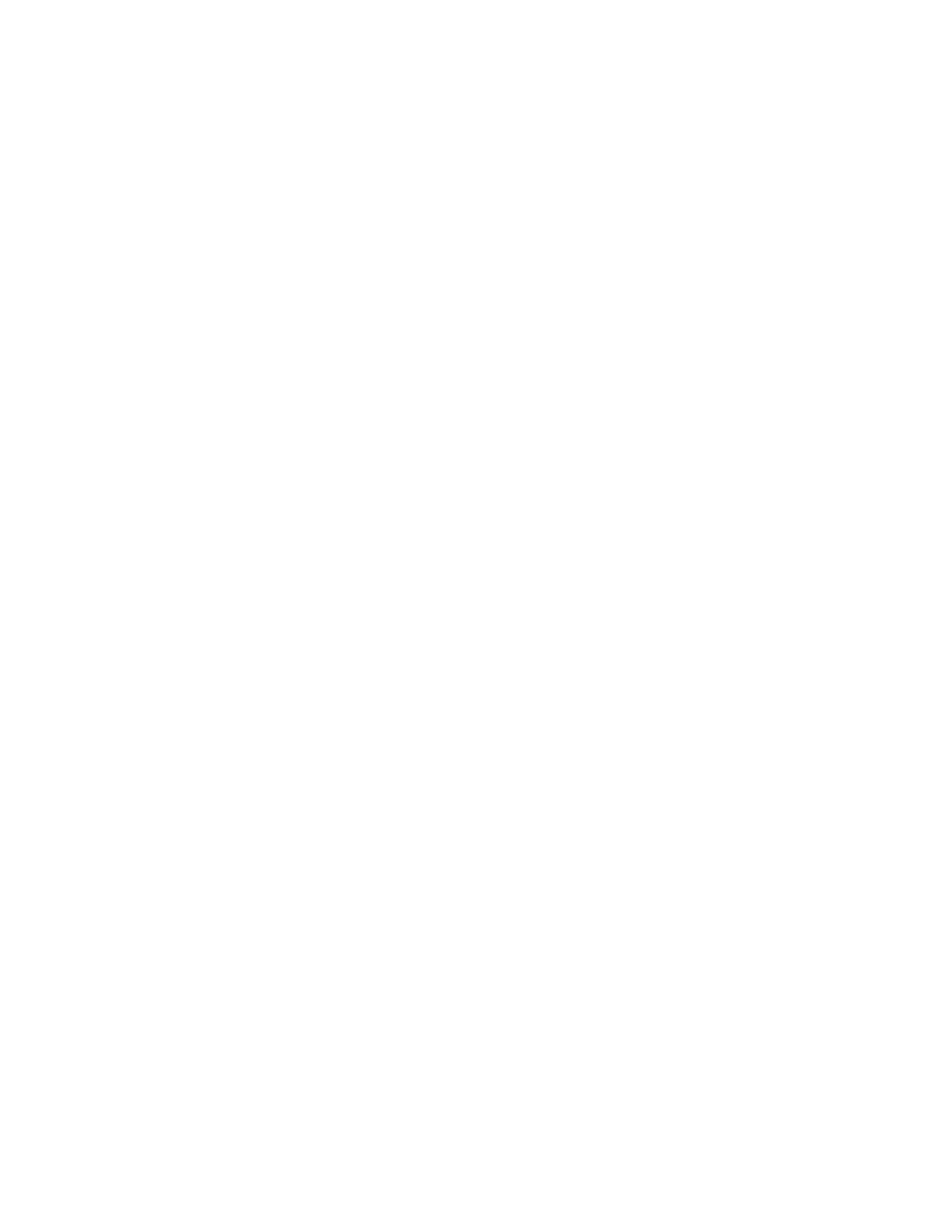
Analysis of 'subject ' column:   
Number of unique values: 2   
Number of missing values: 0   
First few unique values: ['News ' 'politicsNews ']

Analysis of 'date' column:   
Number of unique values: 4   
Number of missing values: 0   
First few unique values: <DatetimeArray>   
['2017-12-31 00:00:00', '2017-12-30 00:00:00', '2017-12-29 00:00:00', '2017-12-25 00:00:00']   
Length: 4, dtype: datetime64[ns]   
Data type: datetime64[ns]   
Successfully converted 'date' to datetime.

Analysis of 'target ' column:   
Number of unique values: 2   
Number of missing values: 0   
Unique values: ['fake ' 'true ']   
Value counts: target   
fake 5   
true 5   
Name: count, dtype: int64

import pandas as pd   
from google.colab import files   
from IPython.display import display

# --- Step 1: Upload the file ---  
# This cell should be run first to get the 'uploaded' variable # Uncomment the line below and run this cell to upload your file

# uploaded = files.upload()

# Define the expected file name   
file\_name = 'fake\_news\_detection\_dataset.csv'   
df = None # Initialize df to None

# --- Step 2: Load the dataset ---  
# Check if the expected file was uploaded and attempt to load it if file\_name in uploaded:  
 try:  
 # Load the dataset from the uploaded file  
 # Use the uploaded dictionary to access the file content df = pd.read\_csv(uploaded[file\_name].decode('utf-8')) print(f"Dataset '{file\_name}' loaded successfully.")

# --- Step 3: Perform initial preprocessing steps ---

# Standardize column names (removing leading/trailing spaces and converting to lowe df.columns = df.columns.str.strip().str.lower()  
 print("Column names standardized.")  
 print("Standardized Columns:", df.columns.tolist())

# Correct column names (handle potential inconsistencies, although standardization # This line might be redundant after .str.strip().str.lower() but kept for robustne # based on the original error cell. Ensure column names match after standardization # Example check: print(df.columns) to see the exact names.

# If 'subject ' and 'target ' became 'subject' and 'target' after standardization, t # Let's verify columns and adjust if needed.

# Assuming standardization made them 'subject' and 'target', the original rename key # If your file had ' subject ' (with space) and ' target ' (with space), the standa # should remove the space. The original rename 'subject ': 'subject' would then fai # Let's rely on standardization and maybe add a check.

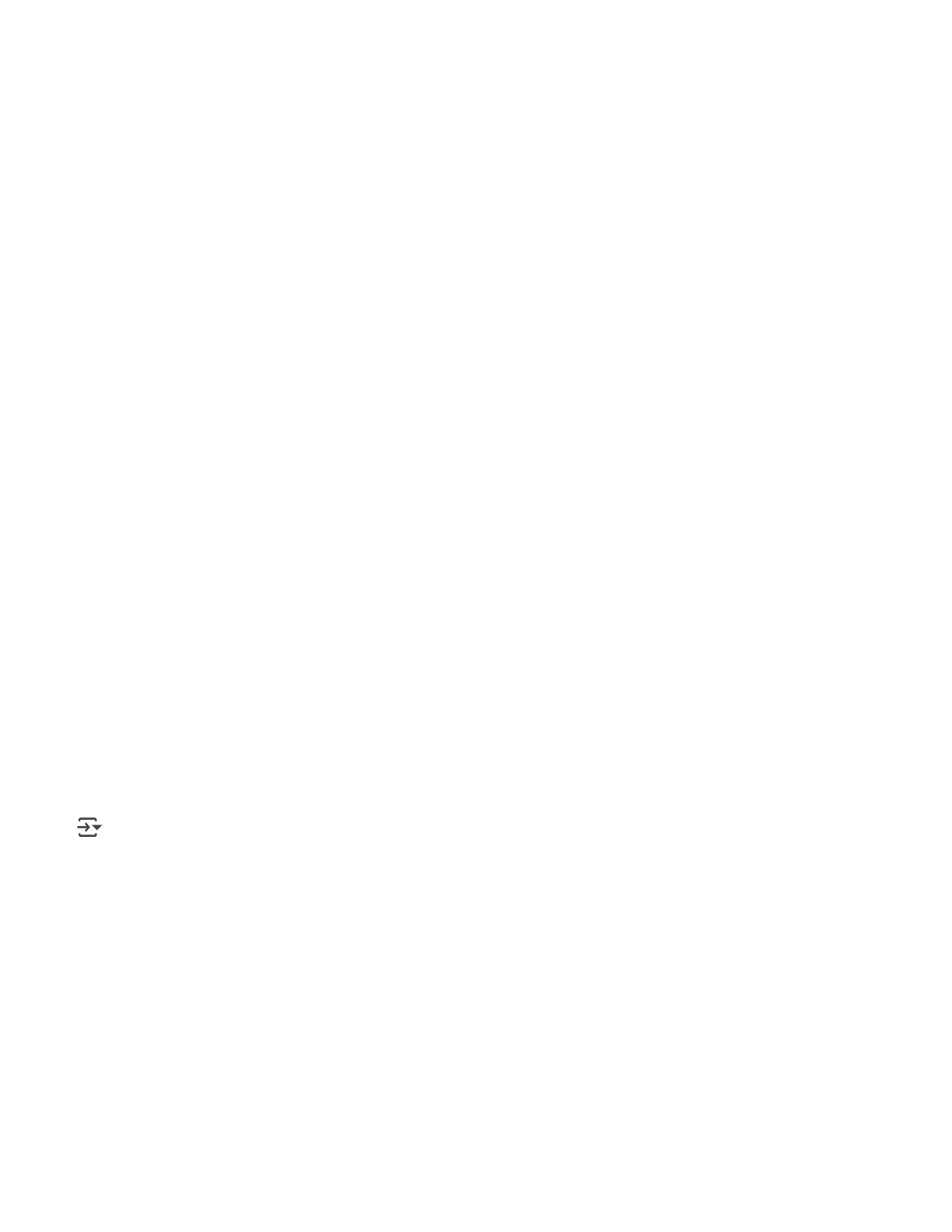
required\_cols = ['title', 'text', 'subject', 'target', 'date'] # Adjust based on yo

if all(col in df.columns for col in required\_cols):  
 print(f"All required columns {required\_cols} found.")

# Handle missing values  
 print("\nHandling missing values...")  
 initial\_rows = len(df)

# Remove rows with missing 'title' or 'text'  
 for col in ['title', 'text']:  
 if df[col].isnull().any():  
 rows\_before\_dropna = len(df)  
 df.dropna(subset=[col], inplace=True)  
 print(f"Removed {rows\_before\_dropna - len(df)} rows with missing '{col}

# Fill missing values for 'subject' with the most frequent value  
 if df['subject'].isnull().any():  
 try:  
 most\_frequent\_subject = df['subject'].mode()[0]  
 rows\_before\_fillna = len(df)  
 df['subject'].fillna(most\_frequent\_subject, inplace=True)  
 print(f"Filled {rows\_before\_fillna - len(df)} missing values in 'subject except IndexError: # Handles case where 'subject' column is empty after drop print("Warning: 'subject' column is empty after dropping rows. Cannot f

print(f"Total rows removed due to missing 'title'/'text': {initial\_rows - len(df

# Remove duplicates  
 num\_duplicates = df.duplicated().sum()  
 if num\_duplicates > 0:  
 df.drop\_duplicates(inplace=True)  
 print(f"Removed {num\_duplicates} duplicate rows.") else:  
 print("No duplicate rows found.")

# Outlier detection and handling (text data) - remove extremely short titles/tex rows\_before\_length\_filter = len(df)  
 df = df[df['title'].astype(str).str.len() > 5].copy() # Use .copy() to avoid Se df = df[df['text'].astype(str).str.len() > 20].copy() # Use .copy()  
 print(f"Removed {rows\_before\_length\_filter - len(df)} rows with extremely short

# Display updated DataFrame info  
 print("\nDataFrame info after preprocessing:") df.info()  
 display(df.head())

else:  
 missing = [col for col in required\_cols if col not in df.columns]  
 print(f"Error: Missing required columns {missing} in '{file\_name}' after loading df = None # Set df to None if essential columns are missing

except Exception as e:  
 print(f"An error occurred while reading or processing '{file\_name}': {e}") df = None # Set df to None if an error occurs during loading/processing else:  
 print(f"Error: Expected file '{file\_name}' was not uploaded.")  
 df = None # Set df to None if the file wasn't uploaded

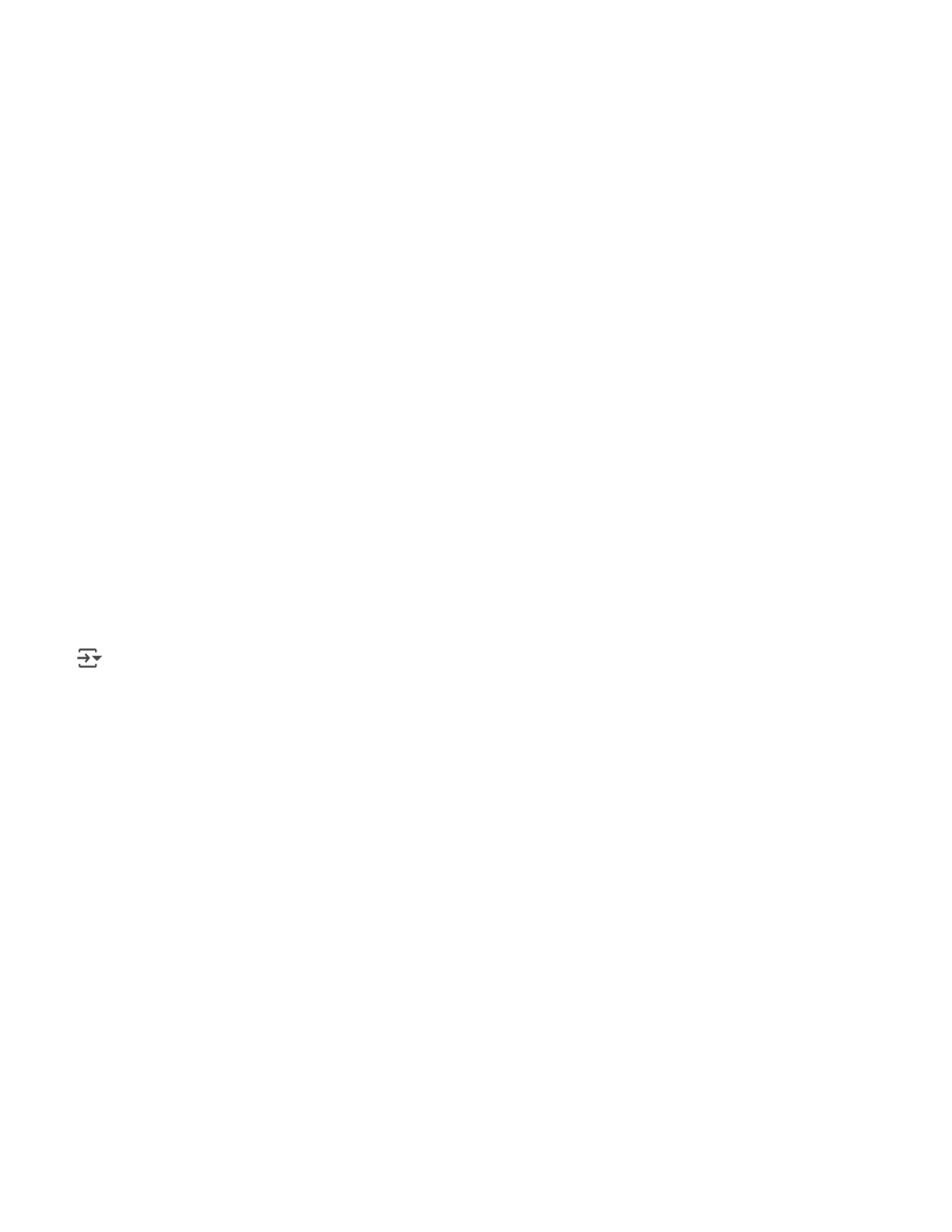
# Now, check if df is a valid DataFrame before proceeding to subsequent cells if df is None:  
 print("\nDataFrame 'df' was not successfully created or contains errors.") print("Subsequent cells relying on 'df' will likely fail.")

Error: Expected file 'fake\_news\_detection\_dataset.csv' was not uploaded.

DataFrame 'df' was not successfully created or contains errors. Subsequent cells relying on 'df' will likely fail.

from sklearn.feature\_extraction.text import TfidfVectorizer   
import numpy as np   
import pandas as pd # Ensure pandas is imported here if not already

# Check if df is a pandas DataFrame and not None   
if df is not None and isinstance(df, pd.DataFrame):  
 # Ensure 'title' and 'text' columns exist before combining  
 if 'title' in df.columns and 'text' in df.columns:  
 # Combine 'title' and 'text' - Ensure they are treated as strings  
 # This prevents potential issues if columns contain non-string data after preproces df['combined\_text'] = df['title'].astype(str) + ' ' + df['text'].astype(str) print("'combined\_text' column created.")

# Create TF-IDF features  
 # Adjust max\_features as needed based on your dataset size and memory constraints tfidf\_vectorizer = TfidfVectorizer(max\_features=5000)  
 print("Starting TF-IDF vectorization...")  
 tfidf\_matrix = tfidf\_vectorizer.fit\_transform(df['combined\_text'])  
 print("TF-IDF vectorization complete.")

# Create a new DataFrame from the TF-IDF array  
 # Ensure the index is reset or aligned if needed, but concat handles this with defau tfidf\_df = pd.DataFrame(tfidf\_matrix.toarray(), columns=tfidf\_vectorizer.get\_feature print("TF-IDF DataFrame created.")

# Concatenate with the original DataFrame  
 # Using join for safety if indices aren't perfectly aligned, though concat with axi # Ensure original columns needed later are kept if necessary.

# The original drop list is applied here:  
 cols\_to\_drop = ['title', 'text', 'combined\_text']  
 # Keep only columns to drop that actually exist in df  
 cols\_to\_drop\_exist = [col for col in cols\_to\_drop if col in df.columns]  
 df = pd.concat([df.drop(columns=cols\_to\_drop\_exist, errors='ignore'), tfidf\_df], ax print(f"Original columns {cols\_to\_drop\_exist} dropped and TF-IDF features concatenat

display(df.head())  
 print("\nDataFrame head after TF-IDF features added:")

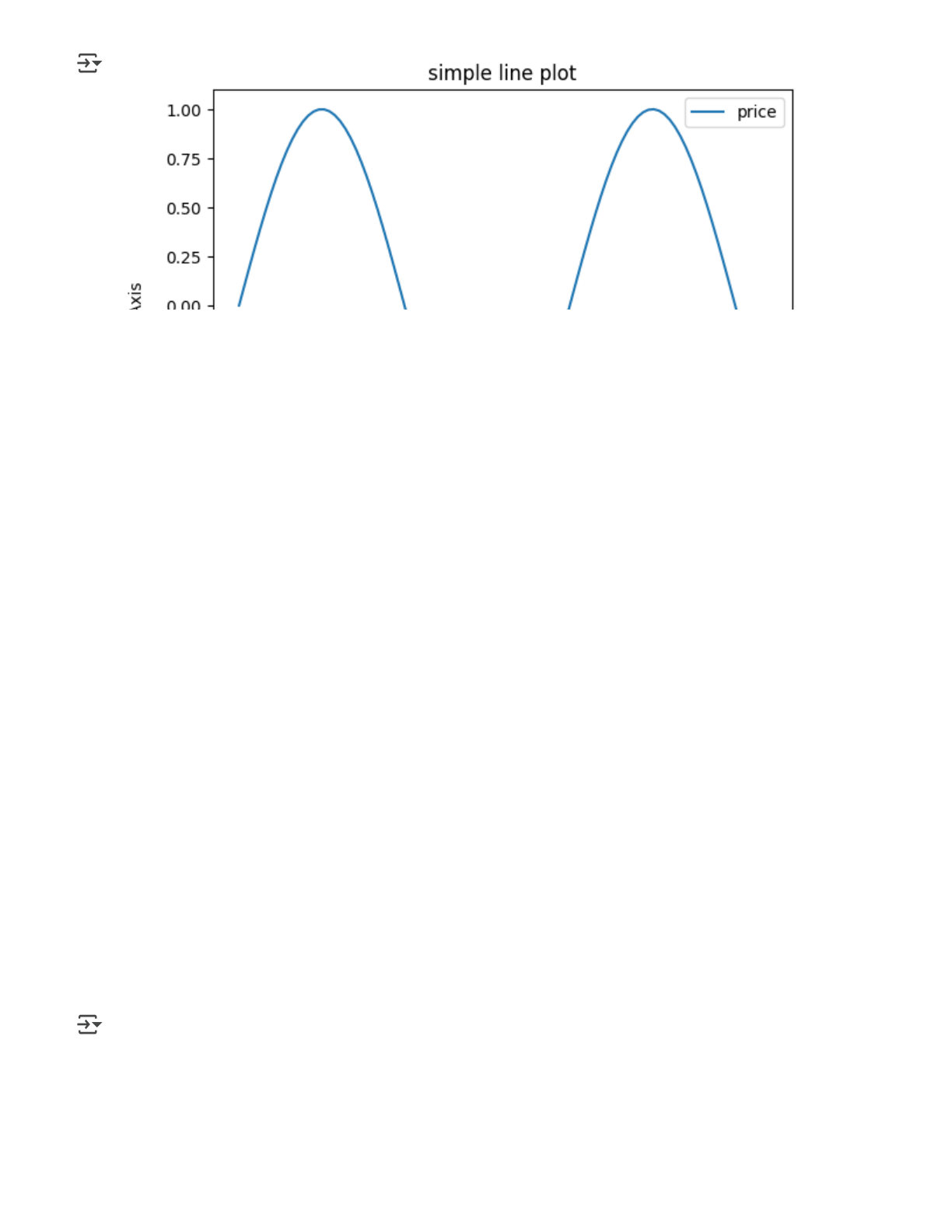
else:  
 print("Error: 'title' or 'text' columns not found in DataFrame. Cannot create 'comb else:  
 print("Error: DataFrame 'df' is not available or is not a DataFrame. Cannot proceed with print("Please ensure the dataset loading and preprocessing steps ran successfully in the

Error: DataFrame 'df' is not available or is not a DataFrame. Cannot proceed wit Please ensure the dataset loading and preprocessing steps ran successfully in th

import matplotlib.pyplot as plt   
import numpy as np

x = np.linspace(0, 10, 100)   
y = np.sin(x)

plt.plot(x,y,label="price")   
plt.xlabel("X Axis")   
plt.ylabel("Y Axis")   
plt.title("simple line plot")   
plt.legend()   
plt.show()



import pandas as pd

# Load dataset - Changed to read\_csv as the filename suggests CSV format # and initial cells loaded a CSV.

try:  
 df = pd.read\_csv('fake\_news\_detection\_dataset.csv')  
 print("Dataset loaded successfully.")   
except FileNotFoundError:  
 print("Error: File 'fake\_news\_detection\_dataset.csv' not found.") except Exception as e:  
 print(f"An error occurred while loading the dataset: {e}")

# Drop missing values   
# Check if df was successfully loaded before proceeding if 'df' in locals():  
 df.dropna(inplace=True)

# Map target labels to binary  
 # Check if 'target' column exists before mapping  
 if 'target' in df.columns:  
 df['label'] = df['target'].map({'fake': 0, 'real': 1})  
 else:  
 print("Warning: 'target' column not found. Cannot create 'label' column.")

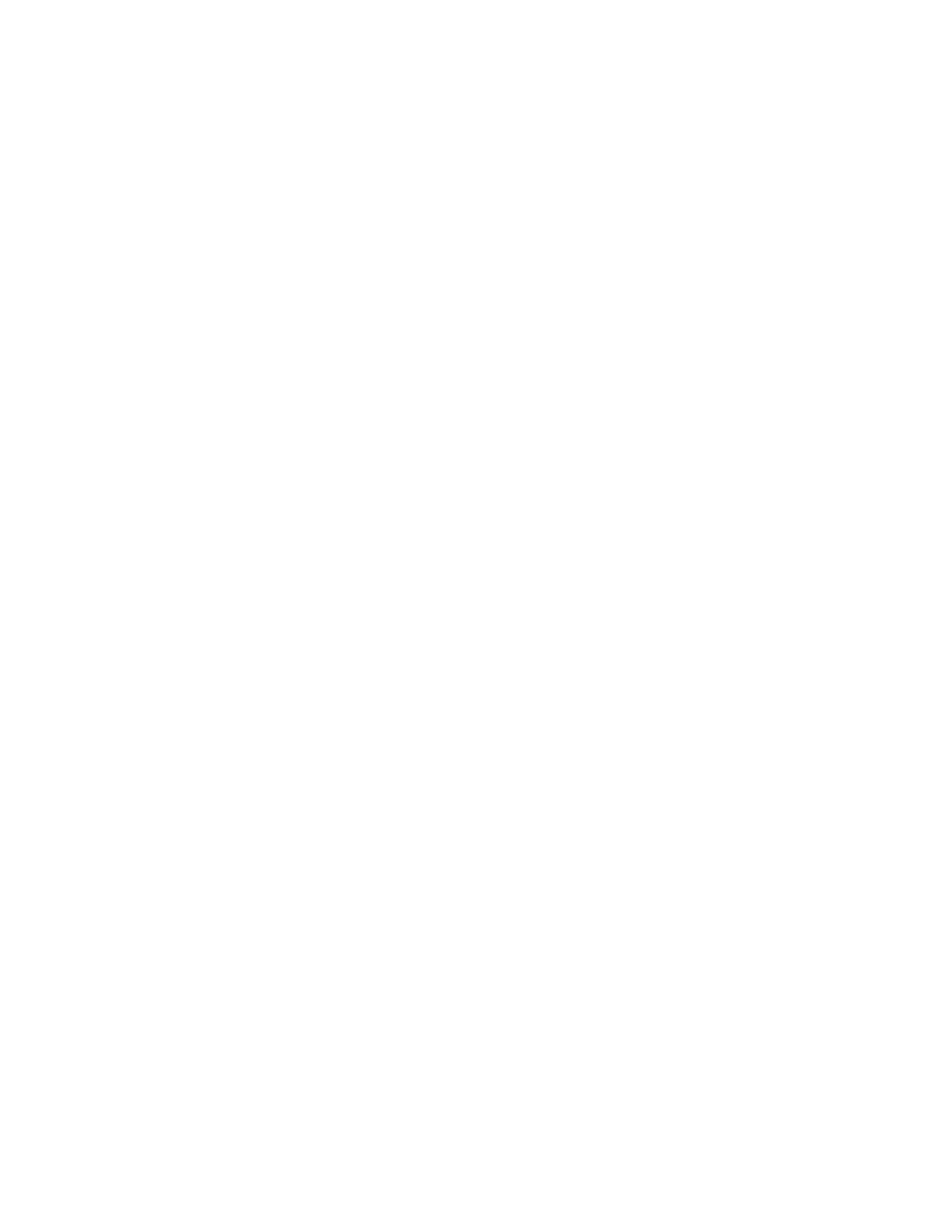
# Optional: Combine title and text  
 # Check if 'title' and 'text' columns exist before combining  
 if 'title' in df.columns and 'text' in df.columns:  
 df['content'] = df['title'] + " " + df['text']  
 else:  
 print("Warning: 'title' or 'text' columns not found. Cannot create 'content' column

# Display the head of the DataFrame to verify changes display(df.head())

Error: File 'fake\_news\_detection\_dataset.csv' not found.

import pandas as pd   
from IPython.display import display # Ensure display is imported if you want to use it

# --- Step 1: Load the dataset ---  
# Load the CSV file instead of the Excel file   
# Use the file name and path that is correct for your environment.

# If you uploaded the file using google.colab.files.upload(),   
# the file will be in the current directory, so just the filename is needed.

try:  
 # Assumes the CSV file 'fake\_news\_detection\_dataset.csv' is available  
 df = pd.read\_csv('fake\_news\_detection\_dataset.csv')  
 print("Dataset loaded successfully from CSV.")   
except FileNotFoundError:  
 print("Error: File 'fake\_news\_detection\_dataset.csv' not found in the current directory df = None # Ensure df is None if loading fails   
except Exception as e:  
 print(f"An error occurred while loading the dataset: {e}")  
 df = None # Ensure df is None if loading fails

# Check if df was successfully loaded before proceeding if df is not None:  
 # --- Step 2: Perform preprocessing steps ---

# Standardize column names (removing leading/trailing spaces and converting to lowercase # This is crucial to ensure column names like 'target ' become 'target'  
 df.columns = df.columns.str.strip().str.lower()  
 print("Column names standardized.")  
 print("Standardized Columns:", df.columns.tolist())

# Drop missing values  
 initial\_rows = len(df)  
 df.dropna(inplace=True)  
 print(f"Dropped {initial\_rows - len(df)} rows with missing values.") print(f"DataFrame shape after dropping NaNs: {df.shape}")

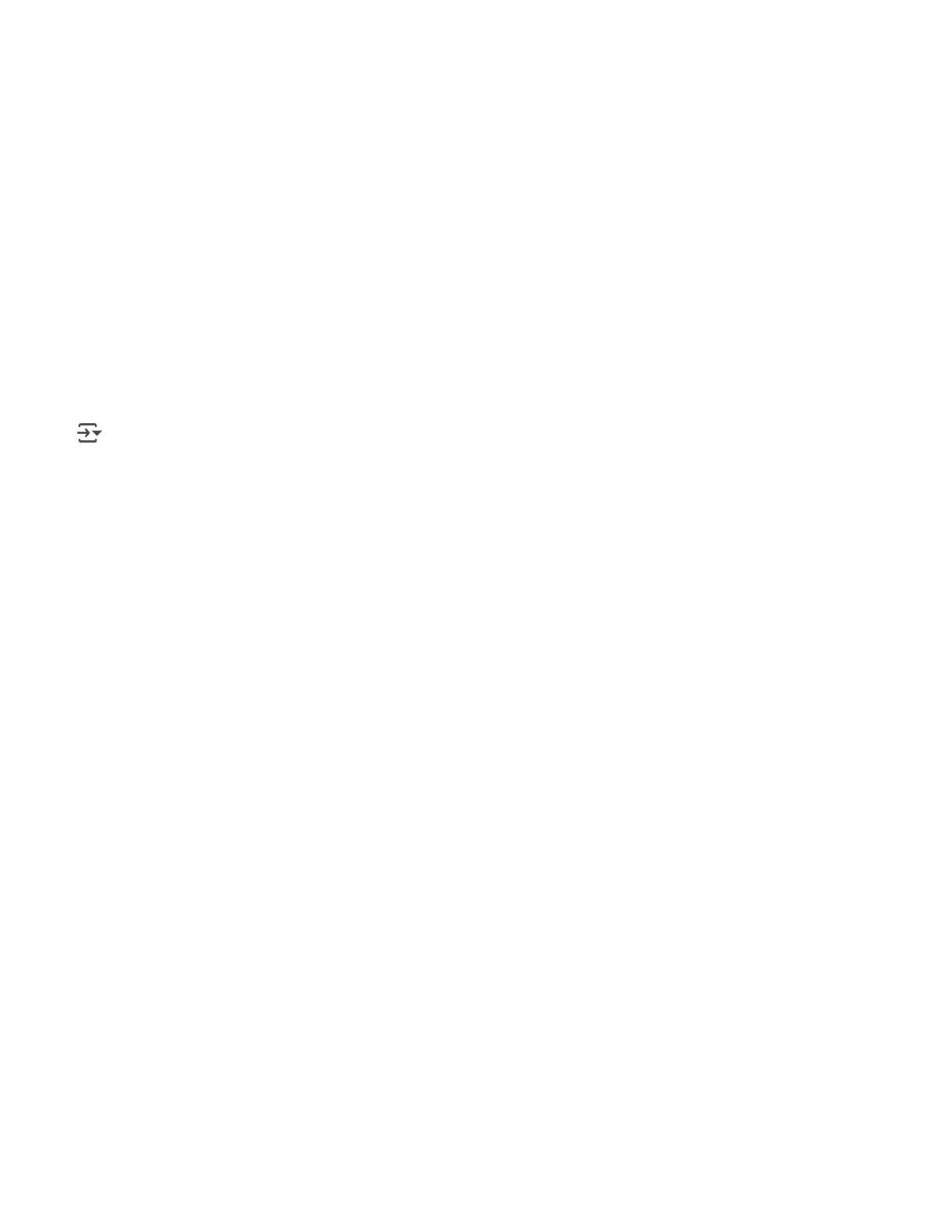
# Map target labels to binary  
 # Check if 'target' column exists AFTER standardization  
 if 'target' in df.columns:  
 # Check unique values in 'target' to ensure they are 'fake' and 'real' unique\_targets = df['target'].unique()  
 print("Unique values in 'target' column:", unique\_targets)

# Ensure the map dictionary covers all expected values or handle others  
 # If there are other values besides 'fake' and 'real', decide how to handle them. # For simplicity, assuming only 'fake' and 'real' or dropping other rows.

# Using .loc to avoid SettingWithCopyWarning if needed, although map usually handle # Ensure the mapping only applies to rows with 'fake' or 'real' targets if other va valid\_targets = ['fake', 'real']  
 df = df[df['target'].isin(valid\_targets)].copy() # Keep only rows with valid target print(f"Filtered DataFrame to keep rows with 'target' in {valid\_targets}. Shape: {df

df['label'] = df['target'].map({'fake': 0, 'real': 1})  
 # Check if mapping resulted in NaNs (e.g., unexpected target values not filtered) if df['label'].isnull().any():  
 print("Warning: NaN values still present in 'label' column after mapping and fi # You might need further inspection or filtering here  
 else:  
 print("'label' column created successfully.")

else:  
 print("Error: 'target' column not found in the DataFrame after standardization. Cann # Set df to None or handle this case if 'target' is essential  
 df = None # Setting df to None prevents subsequent steps from failing

# Optional: Combine title and text  
 # Check if df is still valid and 'title' and 'text' columns exist  
 if df is not None and 'title' in df.columns and 'text' in df.columns:  
 # Ensure both title and text are treated as strings before concatenation  
 df['content'] = df['title'].astype(str) + " " + df['text'].astype(str)  
 print("'content' column created successfully.")  
 elif df is not None: # If df is not None but title or text are missing  
 print("Warning: 'title' or 'text' columns not found. Cannot create 'content' column # Decide if this is a fatal error. If 'content' is essential, set df = None.

# Display the head of the DataFrame to verify changes  
 if df is not None:  
 display(df.head())  
 else:  
 print("DataFrame 'df' is not available after processing.")

else:  
 print("Could not display head as the DataFrame was not loaded successfully.")

Error: File 'fake\_news\_detection\_dataset.csv' not found in the current directory Could not display head as the DataFrame was not loaded successfully.

# Ensure nltk is installed   
!pip install nltk

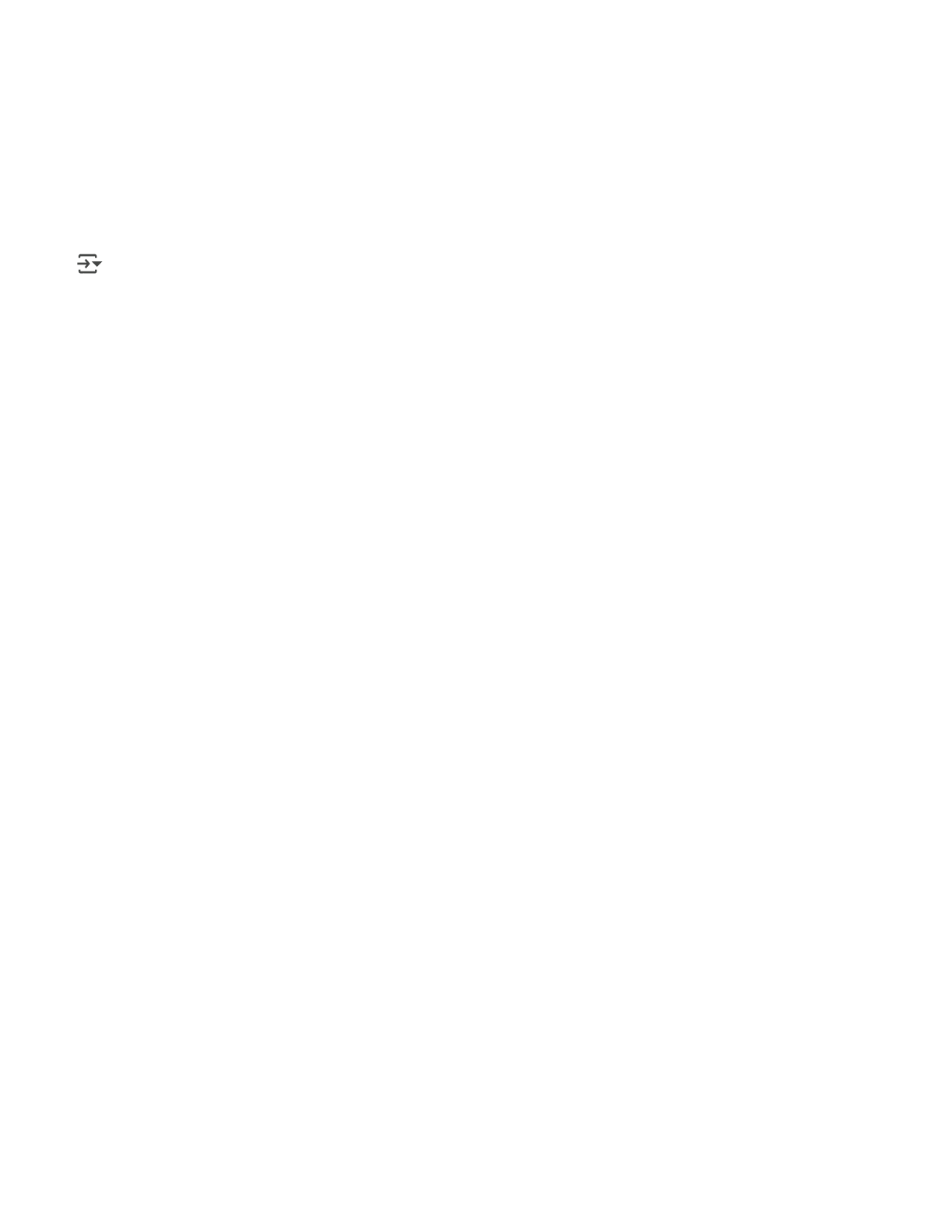
import re   
import string   
from nltk.corpus import stopwords   
from nltk.stem import PorterStemmer   
import nltk   
from IPython.display import display # Import display

# Download the stopwords corpus if not already present # This line is added to fix the LookupError   
try:  
 nltk.data.find('corpora/stopwords')   
except LookupError:  
 nltk.download('stopwords')

# Define stop\_words and stemmer outside the cleaning function stop\_words = set(stopwords.words('english'))   
stemmer = PorterStemmer()

def clean\_text(text):  
 # Ensure input is a string  
 if not isinstance(text, str):  
 # Handle non-string data, perhaps return empty string or original  
 return ""  
 text = text.lower()  
 text = re.sub(r'<.\*?>', '', text) # remove HTML  
 text = re.sub(r'[%s]' % re.escape(string.punctuation), '', text) # remove punctuation text = re.sub(r'\d+', '', text) # remove digits  
 words = text.split()  
 # Filter out empty strings before stemming  
 words = [stemmer.stem(word) for word in words if word and word not in stop\_words] return " ".join(words)

# Check if df exists and 'content' column is present before proceeding

# This check prevents the NameError if the dataframe loading failed.

if 'df' in locals() and 'content' in df.columns:  
 df['clean\_content'] = df['content'].apply(clean\_text)  
 print("'clean\_content' column created successfully.")  
 # Optional: Display the head with the new column  
 display(df[['content', 'clean\_content']].head())   
elif 'df' not in locals():  
 print("Error: DataFrame 'df' was not created. Check the file loading steps.") else: # 'content' column not in df.columns  
 print("Error: 'content' column not found in DataFrame. Cannot apply text cleaning.")

Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packages ( Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11/dist Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages ( Error: DataFrame 'df' was not created. Check the file loading steps.

d

Import display   
import TfidfVectorizer

on-input-22-c7db67bc678f

tion\_dataset.csv')   
y.")

ection\_dataset.csv' not found.")   
loading fails

oading the dataset: {e}")   
loading fails

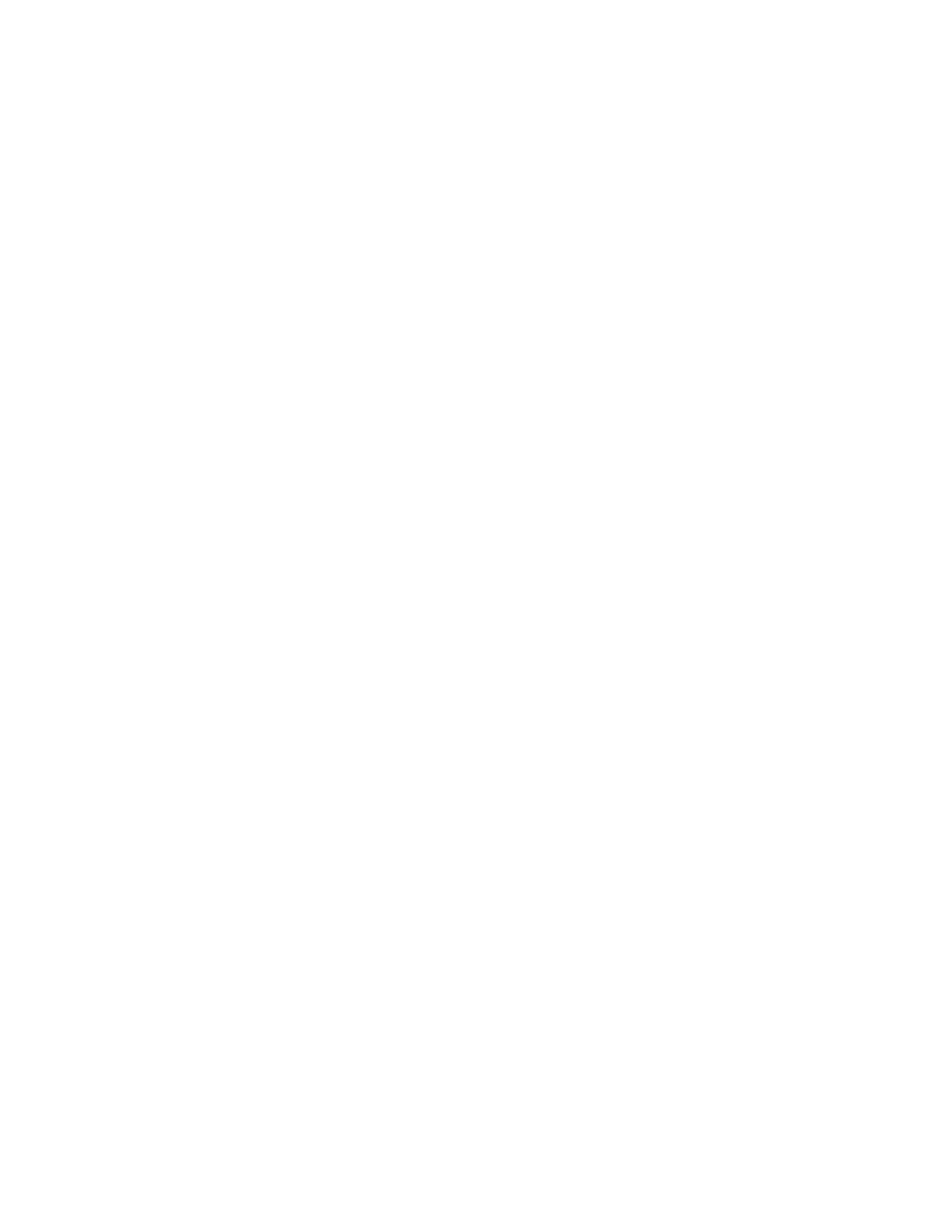
if loading was successful ---

pping NaNs: {df.shape}")

p({'fake': 0, 'real': 1})   
n NaNs (e.g., unexpected target values)

s created in 'label' column after mapping. Check unique values in 'target'.") target':", df['target'].unique())

ated successfully.")

mn not found. Cannot create 'label' column.")   
is case if 'target' is essential   
ntly based on requirements

df.columns and 'text' in df.columns:   
 are treated as strings before concatenation   
astype(str) + " " + df['text'].astype(str)   
ted successfully.")   
ot None but title or text are missing   
ext' columns not found. Cannot create 'content' column.") is case if 'content' is essential   
ntly based on requirements

and 'content' are available ---  
n df.columns:   
us if not already present

topwords')

er outside the cleaning function   
rds('english'))

g   
tr):   
pty string for non-string data

, text) # remove HTML   
e.escape(string.punctuation), '', text) # remove punctuation text) # remove digits

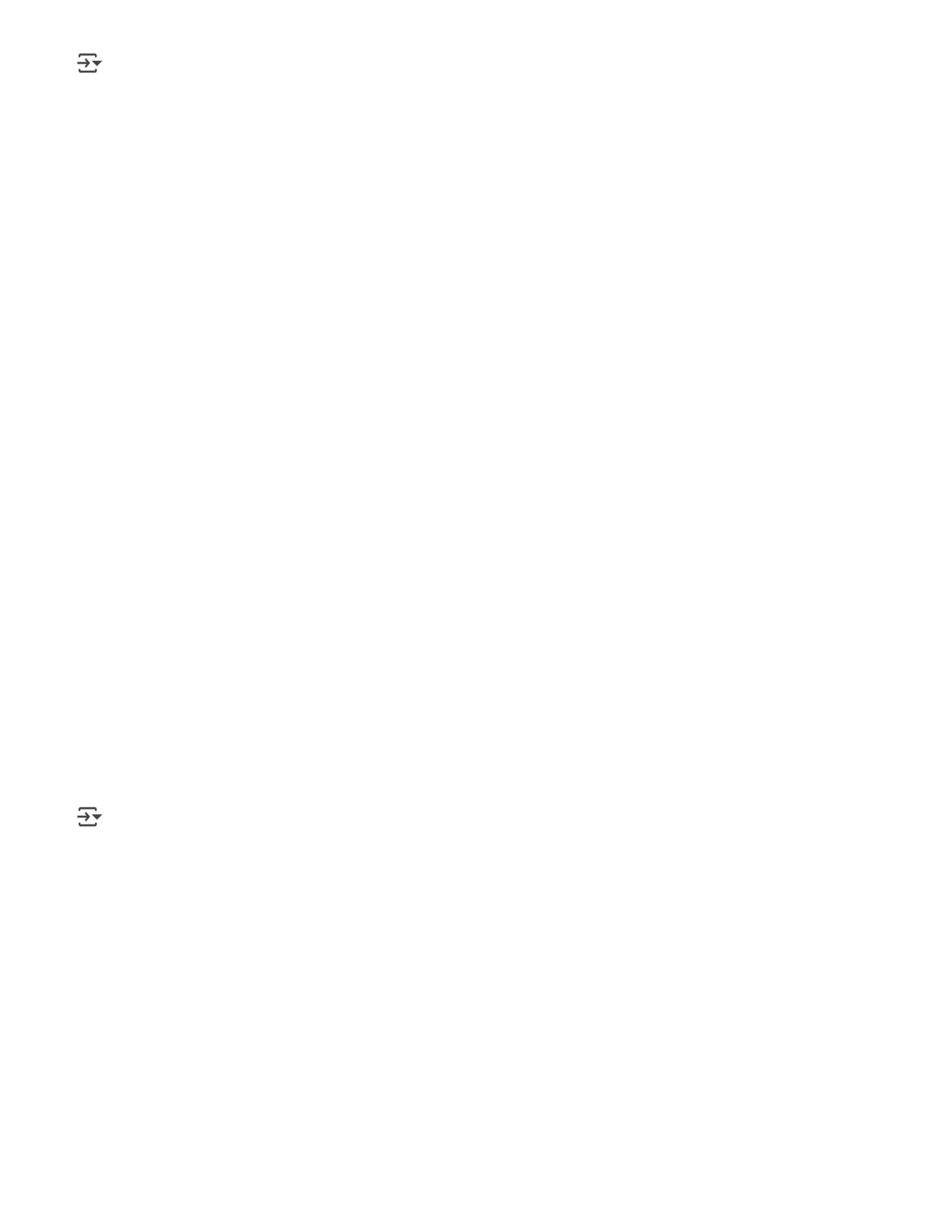
s and stop words before stemming   
d) for word in words if word and word not in stop\_words]

tent'].apply(clean\_text)   
 created successfully.")   
with the new column   
\_content', 'label']].head())   
ot None but 'content' is missing   
mn not found in DataFrame. Cannot apply text cleaning.") s cleaning cannot proceed

df and 'clean\_content' are available ---  
 in df.columns and 'label' in df.columns:   
torization.")   
res=5000)   
\_content'])

abel assignment complete.")   
es): {X.shape}")   
hape}")

ean\_content' column, or 'label' column not available. Cannot perform TF-IDF vectorization.")

Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packages ( Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-p Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3. Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-pa Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11/dist Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages ( Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packag Error: File 'fake\_news\_detection\_dataset.csv' not found.

Error: DataFrame 'df', 'clean\_content' column, or 'label' column not available. C

from sklearn.model\_selection import train\_test\_split   
from sklearn.linear\_model import LogisticRegression   
from sklearn.metrics import accuracy\_score, classification\_report import pandas as pd   
import scipy.sparse # Import scipy.sparse for checking X type

# Split data   
# Check if X and y are defined and are of the expected types before splitting   
if 'X' in locals() and 'y' in locals() and (isinstance(X, (pd.DataFrame, scipy.sparse.csr\_ma X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42

# Train model  
 model = LogisticRegression()  
 model.fit(X\_train, y\_train)

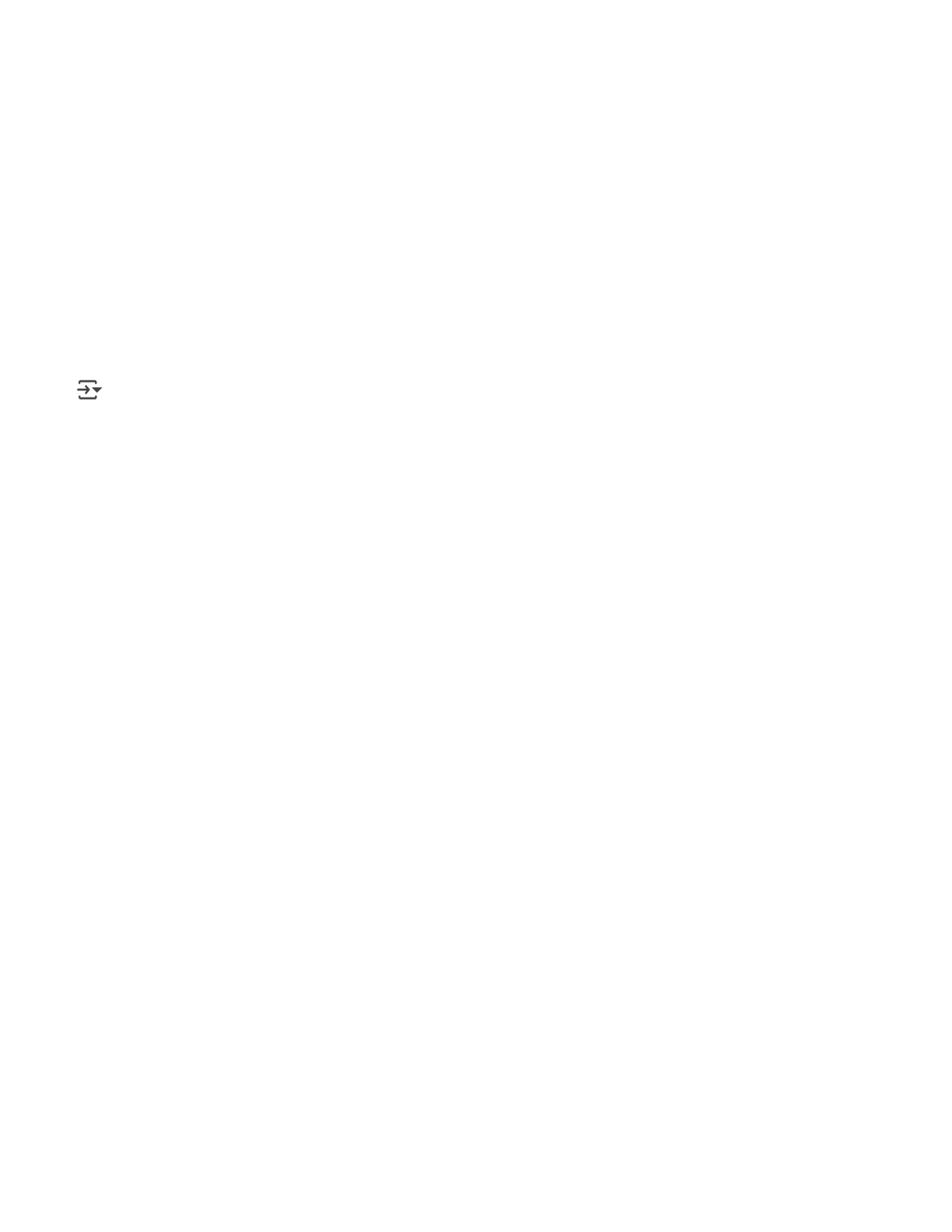
# Evaluate  
 y\_pred = model.predict(X\_test)  
 print("Accuracy:", accuracy\_score(y\_test, y\_pred))  
 print(classification\_report(y\_test, y\_pred))   
else:  
 print("Error: X or y are not defined or not of the expected type. Please ensure the prev # You can add further actions here, like exiting the cell or skipping the rest of the co # exit() # Uncomment to stop execution if X or y are not ready

Error: X or y are not defined or not of the expected type. Please ensure the prev

import pickle   
import pandas as pd # Ensure pandas is imported if needed for other parts of the cell

# Ensure scipy.sparse is imported if X is expected to be a sparse matrix   
# (though it's already imported in the previous cell, good practice to import locally if use import scipy.sparse

# Save model   
# Check if 'model' and 'tfidf' are defined in the current scope   
if 'model' in locals() and 'tfidf' in locals():  
 with open('fake\_news\_model.pkl', 'wb') as f:  
 pickle.dump((model, tfidf), f)  
 print("Model and TF-IDF vectorizer saved successfully.")   
else:  
 print("Error: Model or TF-IDF vectorizer not defined. Cannot save the model.")  
 print("Please check the output of the previous cells to ensure the model was trained and



# Load model (This part assumes the saving was successful or the file exists)   
# It's also good practice to wrap loading in a try-except block   
try:  
 with open('fake\_news\_model.pkl', 'rb') as f:  
 loaded\_model, loaded\_tfidf = pickle.load(f)  
 print("Model and TF-IDF vectorizer loaded successfully.")  
 # You might want to re-assign these to the original variable names 'model' and 'tfidf' # or use the new names 'loaded\_model' and 'loaded\_tfidf' consistently.

# For consistency with the original code, let's reassign:  
 model = loaded\_model  
 tfidf = loaded\_tfidf   
except FileNotFoundError:  
 print("Error: 'fake\_news\_model.pkl' not found. Cannot load the model.") except Exception as e:  
 print(f"An error occurred while loading the model: {e}")

Error: Model or TF-IDF vectorizer not defined. Cannot save the model.

Please check the output of the previous cells to ensure the model was trained an An error occurred while loading the model: Ran out of input

import pandas as pd

# Load the CSV file instead of the Excel file   
# Use the file name and path that is correct for your environment.

# If you uploaded the file using google.colab.files.upload(),   
# the file will be in the current directory, so just the filename is needed.

try:  
 df = pd.read\_csv('fake\_news\_detection\_dataset.csv')  
 print("Dataset loaded successfully.")   
except FileNotFoundError:  
 print("Error: File 'fake\_news\_detection\_dataset.csv' not found in the current directory