1. Upload the Dataset

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Fake.csv to Fake.csv
```

2. Load the Dataset

```
import pandas as pd

df = pd.read_csv("Fake.csv")  # Replace with your uploaded filename
df.head()
```

3. Data Exploration

```
print("Dataset Info:")
print(df.info())

print("\nDataset Description:")
print(df.describe(include='all'))

print("\nMissing Values:")
print(df.isnull().sum())

print("\nDuplicate Rows:")
print(df.duplicated().sum())
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23481 entries, 0 to 23480
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   title    23481 non-null  object
 1   text     23481 non-null  object
 2   subject  23481 non-null  object
 3   date     23481 non-null  object
dtypes: object(4)
memory usage: 733.9+ KB
None

Dataset Description:
                                                    title   text subject  \
count                                               23481  23481   23481   
unique                                              17903  17455       6   
top     MEDIA IGNORES Time That Bill Clinton FIRED His...           News   
freq                                                    6    626    9050   
```

```
                date
count          23481
unique          1681
top     May 10, 2017
freq              46

Missing Values:
title      0
text       0
subject    0
date       0
dtype: int64

Duplicate Rows:
3
```

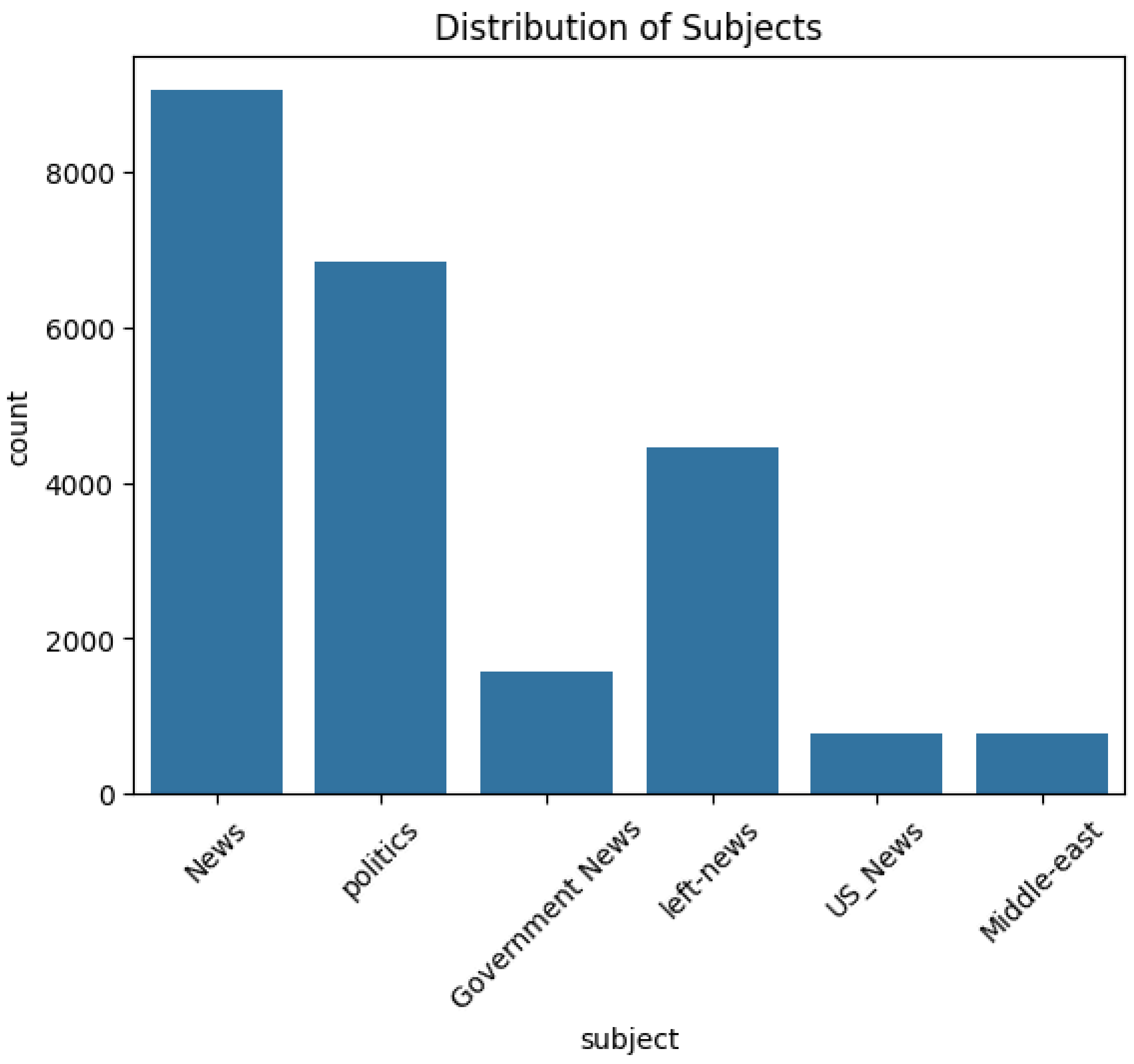
4. Check for Missing Values and Duplicates

```
# Drop duplicates
df = df.drop_duplicates()

# Check again
df.isnull().sum(), df.duplicated().sum()
```

```
(title      0
 text       0
 subject    0
 date       0
 dtype: int64,
 np.int64(0))
```

5. Visualize a Few Features

```
import seaborn as sns
import matplotlib.pyplot as plt

# Plot count of subjects
sns.countplot(x='subject', data=df)
plt.xticks(rotation=45)
plt.title("Distribution of Subjects")
plt.show()
```



6. Identify Target and Features

```
# We'll use 'text' as feature and create a fake news label (1 = Fake)
df['label'] = 1  # Since this dataset contains only fake news, label all as 1
X = df['text']
y = df['label']
```

7. Convert Categorical Columns to Numerical

```
# Not required at this point because 'text' is the only feature, and it's already textual
# However, if needed later, we can convert 'subject' using label encoding.
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df['subject_encoded'] = le.fit_transform(df['subject'])
```

8. One-Hot Encoding

```
# Again, not necessary here since we aren't using 'subject' directly.
# If you were using categorical features like 'subject', you'd do:
df_encoded = pd.get_dummies(df, columns=['subject'])
```

9. Feature Scaling

```
# Scaling is not applied to text features. This step is skipped unless you have numeric f
# However, we can mention it if you later add numerical features like word counts or sent
```

10. Train-Test Split

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

11. Model Building

```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression

# Ensure no nulls and proper format
X_train = X_train.fillna('').astype(str)  # Ensure X_train is of string type
X_test = X_test.fillna('').astype(str)    # Ensure X_test is of string type

# Ensure y_train is a 1D array
y_train = y_train.squeeze()

# Build the model pipeline
model = Pipeline([
    ('tfidf', TfidfVectorizer(stop_words='english', max_df=0.7)),
    ('clf', LogisticRegression(solver='liblinear'))
])

# Fit the model
model.fit(X_train, y_train)

# Make predictions (optional)
y_pred = model.predict(X_test)

# Evaluate the model (optional)
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-27-71ad4f789471> in <cell line: 0>()
      4
      5 # Ensure no nulls and proper format
----> 6 X_train = X_train.fillna('').astype(str)  # Ensure X_train is of string type
      7 X_test = X_test.fillna('').astype(str)    # Ensure X_test is of string type
      8

NameError: name 'X_train' is not defined
```

12. Evaluation

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

try:
    print("➡️ Checking X_test and y_test formats...")
    X_test = X_test.fillna('').astype(str)
    print("✅ Format OK.")

    print("➡️ Checking model training...")
    model.named_steps['clf'].coef_  # test if model is trained
    print("✅ Model is trained.")

    print("➡️ Checking length match...")
    print(f"X_test: {X_test.shape}, y_test: {y_test.shape}")
    if len(X_test) != len(y_test):
        raise ValueError("❌ Mismatch between X_test and y_test length.")

    print("➡️ Predicting...")
    y_pred = model.predict(X_test)
    print("✅ Prediction complete.")

    print("\n🎯 Evaluation Metrics:")
    print("Accuracy Score:", accuracy_score(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))

except Exception as e:
    print("🚨 ERROR OCCURRED DURING EVALUATION:")
    print(type(e).__name__, ":", e)
```

```
➡️ Checking X_test and y_test formats...
🚨 ERROR OCCURRED DURING EVALUATION:
NameError : name 'X_test' is not defined
```

13. Make Predictions from New Input

```
# Step 13: Make Predictions from New Input
new_input = ["Breaking news: NASA discovers water on Mars!"]
```

```
# Ensure input is valid
if not isinstance(new_input, list) or not all(isinstance(i, str) for i in new_input):
    raise ValueError("Input must be a list of strings")

try:
    prediction = model.predict(new_input)
    print("Prediction:", "Fake" if prediction[0] == 1 else "Real")
except Exception as e:
    print("🚨 ERROR during prediction:", type(e).__name__, "→", e)
```

```
🚨 ERROR during prediction: NameError → name 'model' is not defined
```

14. Convert to DataFrame and Encode

```
# Step 14: Convert to DataFrame and Predict

import pandas as pd

# Sample new data
new_data = [
    "New vaccine has been approved by the government",
    "Aliens have landed in California according to reports"
]

# Convert to DataFrame
new_df = pd.DataFrame(new_data, columns=['text'])

# Clean the text column
new_df['text'] = new_df['text'].fillna('').astype(str)

# Predict using your trained model
try:
    new_df['prediction'] = model.predict(new_df['text'])
    new_df['label'] = new_df['prediction'].apply(lambda x: "Fake" if x == 1 else "Real")
    print(new_df)
except Exception as e:
    print("🚨 ERROR during batch prediction:", type(e).__name__, "→", e)
```

```
🚨 ERROR during batch prediction: NameError → name 'model' is not defined
```

15. Predict the Final Grade

```
# Step 15: Predict the confidence score ("final grade")

# Make sure you define new_input correctly
new_input = ["Breaking news: NASA discovers water on Mars!"]

# Ensure model and input are ready
try:
    prob = model.predict_proba(new_input)
    print("Confidence Score (Fake):", prob[0][1])  # Probability that it's fake (label=1)
except Exception as e:
    print("🚨 ERROR during confidence prediction:", type(e).__name__, "→", e)
```

```
🚨 ERROR during confidence prediction: NameError → name 'model' is not defined
```

16. Deployment – Building an Interactive App

```
!pip install gradio
import gradio as gr
```

```
Collecting gradio
  Downloading gradio-5.29.0-py3-none-any.whl.metadata (16 kB)
Collecting aiofiles<25.0,>=22.0 (from gradio)
  Downloading aiofiles-24.1.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/local/lib/python3.11/dist-p
Collecting fastapi<1.0,>=0.115.2 (from gradio)
  Downloading fastapi-0.115.12-py3-none-any.whl.metadata (27 kB)
Collecting ffmpy (from gradio)
  Downloading ffmpy-0.5.0-py3-none-any.whl.metadata (3.0 kB)
Collecting gradio-client==1.10.0 (from gradio)
  Downloading gradio_client-1.10.0-py3-none-any.whl.metadata (7.1 kB)
Collecting groovy~=0.1 (from gradio)
  Downloading groovy-0.1.2-py3-none-any.whl.metadata (6.1 kB)
Requirement already satisfied: httpx>=0.24.1 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: huggingface-hub>=0.28.1 in /usr/local/lib/python3.1
Requirement already satisfied: jinja2<4.0 in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: markupsafe<4.0,>=2.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: numpy<3.0,>=1.0 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: orjson~=3.0 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: pandas<3.0,>=1.0 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: pillow<12.0,>=8.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: pydantic<2.12,>=2.0 in /usr/local/lib/python3.11/di
Collecting pydub (from gradio)
  Downloading pydub-0.25.1-py2.py3-none-any.whl.metadata (1.4 kB)
Collecting python-multipart>=0.0.18 (from gradio)
  Downloading python_multipart-0.0.20-py3-none-any.whl.metadata (1.8 kB)
Requirement already satisfied: pyyaml<7.0,>=5.0 in /usr/local/lib/python3.11/dist-
Collecting ruff>=0.9.3 (from gradio)
  Downloading ruff-0.11.8-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.
Collecting safehttpx<0.2.0,>=0.1.6 (from gradio)
  Downloading safehttpx-0.1.6-py3-none-any.whl.metadata (4.2 kB)
Collecting semantic-version~=2.0 (from gradio)
  Downloading semantic_version-2.10.0-py2.py3-none-any.whl.metadata (9.7 kB)
Collecting starlette<1.0,>=0.40.0 (from gradio)
  Downloading starlette-0.46.2-py3-none-any.whl.metadata (6.2 kB)
Collecting tomlkit<0.14.0,>=0.12.0 (from gradio)
  Downloading tomlkit-0.13.2-py3-none-any.whl.metadata (2.7 kB)
Requirement already satisfied: typer<1.0,>=0.12 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: typing-extensions~=4.0 in /usr/local/lib/python3.11
Collecting uvicorn>=0.14.0 (from gradio)
  Downloading uvicorn-0.34.2-py3-none-any.whl.metadata (6.5 kB)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: websockets<16.0,>=10.0 in /usr/local/lib/python3.11,
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11,
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11,
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.11/
```

17. Create a Prediction Function

```
def fake_news_predictor(text):
    pred = model.predict([text])[0]
    proba = model.predict_proba([text])[0][1]
    label = "Fake" if pred == 1 else "Real"
    return f"{label} News (Confidence: {proba:.2f})"
```

18. Create the Gradio Interface

```
def fake_news_predictor(text):
    try:
        prediction = model.predict([text])[0]
        proba = model.predict_proba([text])[0][1]  # probability of being Fake
        label = "Fake" if prediction == 1 else "Real"
        return f"{label} News (Confidence: {proba:.2f})"
    except Exception as e:
        return f"❌ Error: {str(e)}"
```

```
# Make sure Gradio is installed
!pip install gradio --quiet

import gradio as gr

# Your prediction function
def fake_news_predictor(text):
    try:
        prediction = model.predict([text])[0]
        proba = model.predict_proba([text])[0][1]
        label = "Fake" if prediction == 1 else "Real"
        return f"📰 Prediction: {label}\n🔍 Confidence (Fake): {proba:.2f}"
    except Exception as e:
        return f"❌ Error: {str(e)}"

# Launch the interface
```

```
iface = gr.Interface(
    fn=fake_news_predictor,
    inputs="text",
    outputs="text",
    title="🧠 Fake News Detection Chatbot",
    description="Enter a news article to check if it's Fake or Real. Powered by Logistic
)

iface.launch()
```

```
It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio a

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://d900428432d5c82bf5.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `
```

gradio

No interface is running right now