Upload the Dataset

```
from google.colab import files
import pandas as pd

# Upload the file
uploaded = files.upload()

# Load the dataset
df = pd.read_csv('competition_test_bodies.csv')


# Show the first few rows
df.head()
```

Choose Files competition..._bodies.csv

- **competition_test_bodies.csv**(text/csv) - 2045680 bytes, last modified: 5/19/2025 - 100% done

```
Saving competition_test_bodies.csv to competition_test_bodies (2).csv
```

	Body ID	articleBody
0	1	Al-Sisi has denied Israeli reports stating tha...
1	2	A bereaved Afghan mother took revenge on the T...
2	3	CNBC is reporting Tesla has chosen Nevada as t...
3	12	A 4-inch version of the iPhone 6 is said to be...
4	19	GR editor's Note\n\nThere are no reports in th...

Next steps: Generate code with df View recommended plots New interactive sheet

Load the Dataset

```
import pandas as pd

# Read the uploaded CSV file
df = pd.read_csv('competition_test_bodies.csv')

# Display the first few rows
print("📌 Preview of the dataset:")
print(df.head())

# Display the shape of the dataset
print("\n✅ Dataset shape:", df.shape)

# Display column names
print("\n🧾 Columns:", df.columns.tolist())

# Display data types and non-null counts
print("\n🔍 Dataset Info:")
df.info()
```

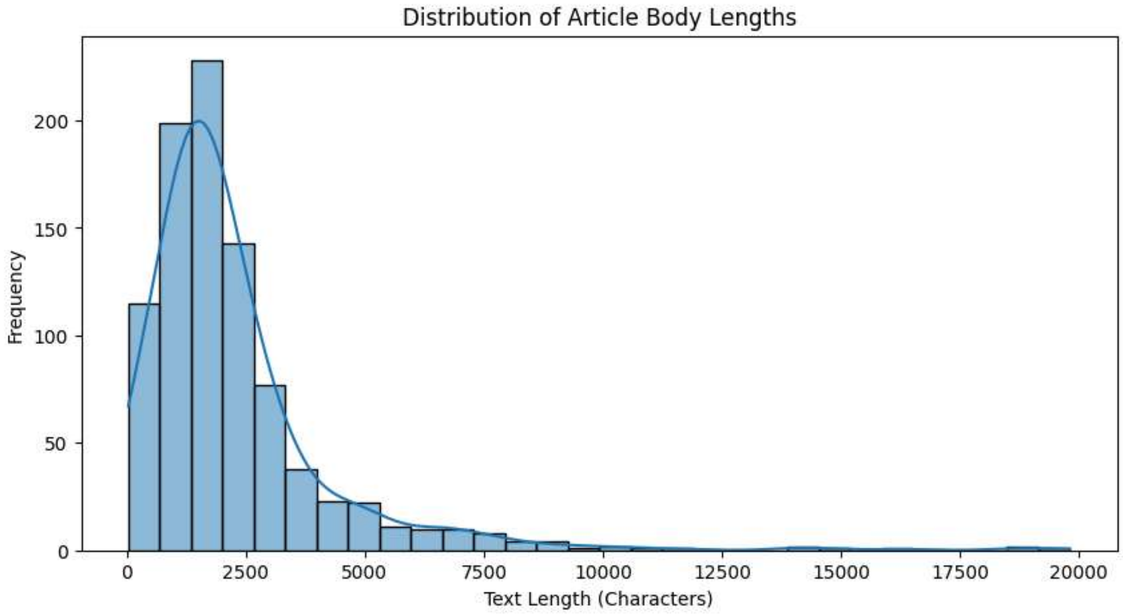
```
📌 Preview of the dataset:
   Body ID                                        articleBody
0        1  Al-Sisi has denied Israeli reports stating tha...
1        2  A bereaved Afghan mother took revenge on the T...
2        3  CNBC is reporting Tesla has chosen Nevada as t...
3       12  A 4-inch version of the iPhone 6 is said to be...
4       19  GR editor's Note\n\nThere are no reports in th...

✅ Dataset shape: (904, 2)

🧾 Columns: ['Body ID', 'articleBody']

🔍 Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 904 entries, 0 to 903
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Body ID      904 non-null    int64 
 1   articleBody  904 non-null    object
dtypes: int64(1), object(1)
```

```
memory usage: 14.3+ KB
```

Data Exploration

```
# Check for missing values
print("❗ Missing Values:\n", df.isnull().sum())

# Check for duplicate entries
print("\n🔁 Duplicate Rows:", df.duplicated().sum())

# Check unique articleBody lengths
df['text_length'] = df['articleBody'].apply(len)
print("\n📝 Text Length Stats:")
print(df['text_length'].describe())

# Histogram of text lengths
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 5))
sns.histplot(df['text_length'], bins=30, kde=True)
plt.title('Distribution of Article Body Lengths')
plt.xlabel('Text Length (Characters)')
plt.ylabel('Frequency')
plt.show()
```

```
❗ Missing Values:
 Body ID        0
articleBody    0
dtype: int64

🔁 Duplicate Rows: 0

📝 Text Length Stats:
count      904.000000
mean      2235.008850
std       2132.850585
min         29.000000
25%       1096.000000
50%       1730.500000
75%       2616.750000
max      19815.000000
Name: text_length, dtype: float64
```



Check for Missing Values and Duplicates

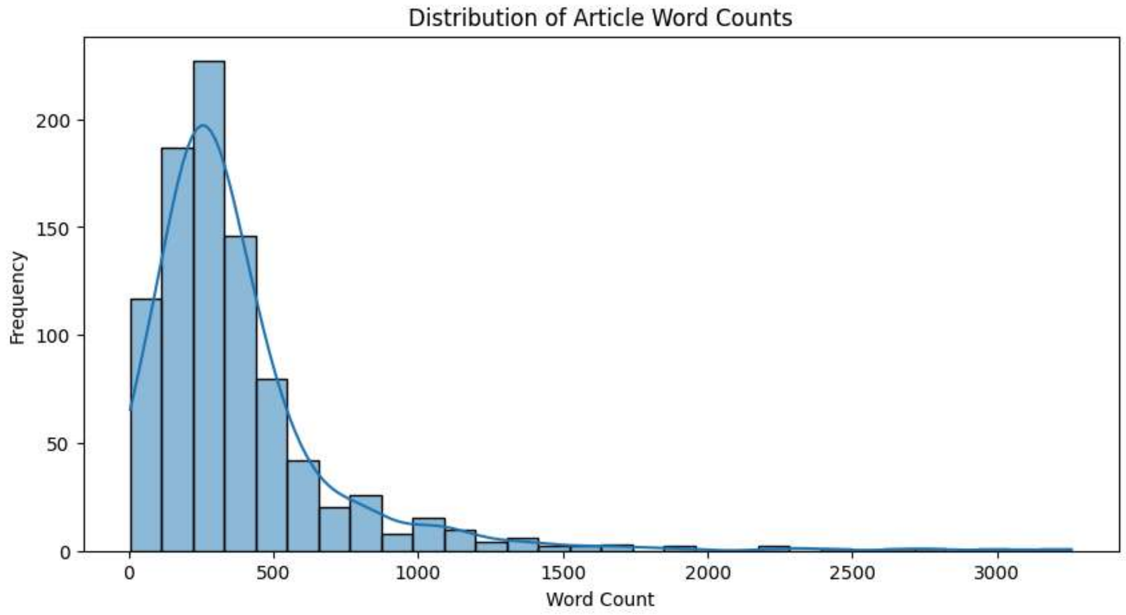
```
# Check for missing values in each column
print("📋 Missing Values:")
print(df.isnull().sum())
```

```
# Check for duplicated rows
duplicate_count = df.duplicated().sum()
print(f"\n🔁 Number of duplicate rows: {duplicate_count}")
```

```
📋 Missing Values:
Body ID        0
articleBody    0
text_length    0
dtype: int64

🔁 Number of duplicate rows: 0
```

Visualize a Few Features

```
# Add a column for word count
df['word_count'] = df['articleBody'].apply(lambda x: len(str(x).split()))

# Plot histogram of word counts
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 5))
sns.histplot(df['word_count'], bins=30, kde=True)
plt.title('Distribution of Article Word Counts')
plt.xlabel('Word Count')
plt.ylabel('Frequency')
plt.show()
```



Identify Target and Features

```
import pandas as pd

df = pd.read_csv('/content/competition_test_bodies.csv')
print(df.columns)
```

```
Index(['Body ID', 'articleBody'], dtype='object')
```

Convert Categorical Columns to Numerical

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize TF-IDF Vectorizer
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)

# Convert text to numeric features
```

```
X_tfidf = vectorizer.fit_transform(df['articleBody'])

# View the shape of the result
print("TF-IDF Matrix shape:", X_tfidf.shape)
```

```
TF-IDF Matrix shape: (904, 5000)
```

One-Hot Encoding

```
from sklearn.preprocessing import OneHotEncoder
import numpy as np

# Sample news headlines (simplified for demo)
news_headlines = [
    "Fake news spreads fast",
    "Truth will prevail",
    "Detect fake reports",
    "Report the truth"
]

# Step 1: Build vocabulary of unique words
vocab = sorted(set(word.lower() for headline in news_headlines for word in headline.split()))
print("Vocabulary:", vocab)

# Step 2: Encode each headline as a sequence of one-hot vectors for each word
# For simplicity, we'll create one-hot vectors per word, then sum to get headline vector

# Map words to indices
word_to_index = {word: i for i, word in enumerate(vocab)}

# Create one-hot vectors for each headline
def one_hot_encode_headline(headline):
    one_hot_vector = np.zeros(len(vocab))
    for word in headline.lower().split():
        index = word_to_index[word]
        one_hot_vector[index] = 1  # Mark presence of word
    return one_hot_vector

# Encode all headlines
encoded_headlines = np.array([one_hot_encode_headline(headline) for headline in news_headlines])

print("\nOne-hot encoded vectors for headlines:\n", encoded_headlines)
```

```
Vocabulary: ['detect', 'fake', 'fast', 'news', 'prevail', 'report', 'reports', 'spreads', 'the', 'truth', 'will']

One-hot encoded vectors for headlines:
 [[0. 1. 1. 1. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1.]
 [1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 1. 1. 0.]]
```

Feature Scaling

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import numpy as np

# Sample features extracted from news articles (e.g., word counts, sentiment score)
features = np.array([
    [10, 0.5],   # Article 1
    [200, -0.1], # Article 2
    [50, 0.0],   # Article 3
    [300, 0.9]   # Article 4
])

print("Original features:\n", features)

# Min-Max Scaling
min_max_scaler = MinMaxScaler()
features_minmax = min_max_scaler.fit_transform(features)
print("\nFeatures after Min-Max Scaling:\n", features_minmax)

# Standardization
standard_scaler = StandardScaler()
```

```
features_standard = standard_scaler.fit_transform(features)
print("\nFeatures after Standardization:\n", features_standard)
```

```
Original features:
 [[ 1.e+01  5.e-01]
 [ 2.e+02 -1.e-01]
 [ 5.e+01  0.e+00]
 [ 3.e+02  9.e-01]]

Features after Min-Max Scaling:
 [[0.         0.6       ]
 [0.65517241 0.        ]
 [0.13793103 0.1       ]
 [1.         1.        ]]

Features after Standardization:
 [[-1.11679563  0.43495884]
 [ 0.51544414 -1.0563286 ]
 [-0.7731662  -0.8077807 ]
 [ 1.37451769  1.42915046]]
```

Train-Test Split

```
from sklearn.model_selection import train_test_split

# Sample data: headlines and labels (0 = real, 1 = fake)
headlines = [
    "Fake news spreads fast",
    "Truth will prevail",
    "Detect fake reports",
    "Report the truth"
]

labels = [1, 0, 1, 0]

# Split data: 75% train, 25% test
X_train, X_test, y_train, y_test = train_test_split(
    headlines, labels, test_size=0.25, random_state=42
)

print("Training data:", X_train)
print("Training labels:", y_train)
print("\nTesting data:", X_test)
print("Testing labels:", y_test)
```

```
Training data: ['Report the truth', 'Fake news spreads fast', 'Detect fake reports']
Training labels: [0, 1, 1]

Testing data: ['Truth will prevail']
Testing labels: [0]
```

Model Building

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Sample dataset: headlines + labels (0 = real, 1 = fake)
headlines = [
    "Fake news spreads fast",
    "Truth will prevail",
    "Detect fake reports",
    "Report the truth",
    "Breaking news: fake story",
    "Verified report confirms facts"
]

labels = [1, 0, 1, 0, 1, 0]

# Step 1: Split into train/test
X_train, X_test, y_train, y_test = train_test_split(
    headlines, labels, test_size=0.33, random_state=42
)
```

```
# Step 2: Convert text to numeric features (Bag of Words)
vectorizer = CountVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

# Step 3: Train Logistic Regression model
model = LogisticRegression()
model.fit(X_train_vec, y_train)

# Step 4: Predict and evaluate
y_pred = model.predict(X_test_vec)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 1.0

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         1

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2
```

Evaluation

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report

# Assume y_test and y_pred from your model predictions

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0

Confusion Matrix:
[[1 0]
 [0 1]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         1

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2
```

Make Predictions from New Input

```
# New headlines to predict
new_headlines = [
    "Fake story uncovered by researchers",
    "Official report confirms the truth",
    "Unbelievable fake news alert"
]

# Step 1: Convert new text data to numeric features using the same vectorizer
```

```
new_vec = vectorizer.transform(new_headlines)

# Step 2: Predict using the trained model
predictions = model.predict(new_vec)

# Step 3: Display predictions (0 = real, 1 = fake)
for headline, pred in zip(new_headlines, predictions):
    label = "Fake" if pred == 1 else "Real"
    print(f"Headline: '{headline}' -> Prediction: {label}")
```

```
Headline: 'Fake story uncovered by researchers' -> Prediction: Fake
Headline: 'Official report confirms the truth' -> Prediction: Real
Headline: 'Unbelievable fake news alert' -> Prediction: Fake
```

Convert to DataFrame and Encode

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Sample data with headlines and labels as strings
data = {
    "headline": [
        "Fake news spreads fast",
        "Truth will prevail",
        "Detect fake reports",
        "Report the truth"
    ],
    "label": [
        "fake",
        "real",
        "fake",
        "real"
    ]
}

# Step 1: Create DataFrame
df = pd.DataFrame(data)
print("Original DataFrame:\n", df)

# Step 2: Encode labels to numbers (fake=1, real=0)
label_encoder = LabelEncoder()
df['label_encoded'] = label_encoder.fit_transform(df['label'])

print("\nDataFrame after encoding labels:\n", df)
```

```
Original DataFrame:
                   headline label
0  Fake news spreads fast  fake
1      Truth will prevail  real
2     Detect fake reports  fake
3        Report the truth  real

DataFrame after encoding labels:
                   headline label  label_encoded
0  Fake news spreads fast  fake              0
1      Truth will prevail  real              1
2     Detect fake reports  fake              0
3        Report the truth  real              1
```

Predict the Final Grade

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Sample dataset
data = {
    'homework_score': [80, 90, 70, 85, 95],
    'exam_score': [75, 88, 65, 90, 92],
    'attendance': [90, 100, 85, 95, 100],
    'final_grade': [78, 92, 70, 88, 95]
}
```

```
df = pd.DataFrame(data)

# Features and target
X = df[['homework_score', 'exam_score', 'attendance']]
y = df['final_grade']

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

# Model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

print("Predicted final grades:", y_pred)
print("Actual final grades:", y_test.values)

# Evaluate
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Predicted final grades: [95.875]
Actual final grades: [92]
Mean Squared Error: 15.015625
```

Deployment-Building an Interactive App

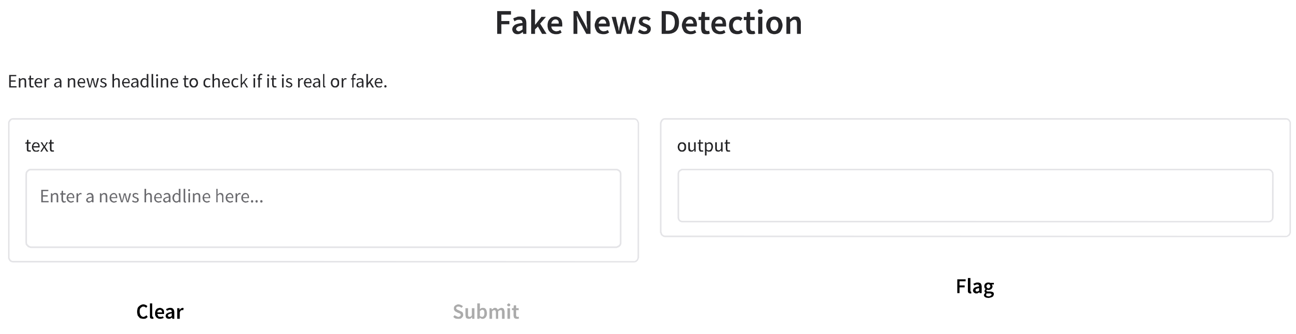
```
!pip install Gradio
```

```
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-packages (from Gradio) (0.15.3)
Requirement already satisfied: typing-extensions~=4.0 in /usr/local/lib/python3.11/dist-packages (from Gradio) (4.13.2)
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 (from gradio-client==1.10.1->Gradio) (2025.3.2)
Requirement already satisfied: websockets<16.0,>=10.0 in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.1->Gradio)













  Downloading ruff-0.11.10-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.6 MB)
```

```
    Downloading ruff-0.11.10-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.6 MB)
       ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.6/11.6 MB 45.5 MB/s eta 0:00:00
    Downloading safehttpx-0.1.6-py3-none-any.whl (8.7 kB)
    Downloading semantic_version-2.10.0-py2.py3-none-any.whl (15 kB)
    Downloading starlette-0.46.2-py3-none-any.whl (72 kB)
       ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 72.0/72.0 kB 4.9 MB/s eta 0:00:00
    Downloading tomlkit-0.13.2-py3-none-any.whl (37 kB)
    Downloading uvicorn-0.34.2-py3-none-any.whl (62 kB)
       ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.5/62.5 kB 3.9 MB/s eta 0:00:00
    Downloading ffmpy-0.5.0-py3-none-any.whl (6.0 kB)
    Downloading pydub-0.25.1-py2.py3-none-any.whl (32 kB)
    Installing collected packages: pydub, uvicorn, tomlkit, semantic-version, ruff, python-multipart, groovy, ffmpy, aiofiles, starlette,
    Successfully installed Gradio-5.30.0 aiofiles-24.1.0 fastapi-0.115.12 ffmpy-0.5.0 gradio-client-1.10.1 groovy-0.1.2 pydub-0.25.1 pyth
```

```python
import gradio as gr
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression

# Sample training data
headlines = [
    "Fake news spreads fast",
    "Truth will prevail",
    "Detect fake reports",
    "Report the truth",
    "Breaking news: fake story",
    "Verified report confirms facts"
]
labels = [1, 0, 1, 0, 1, 0]

# Train model and vectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(headlines)
model = LogisticRegression()
model.fit(X, labels)

# Prediction function
def predict_fake_news(text):
    vec = vectorizer.transform([text])
    pred = model.predict(vec)[0]
    return "Fake News" if pred == 1 else "Real News"

# Build Gradio interface
iface = gr.Interface(
    fn=predict_fake_news,
    inputs=gr.Textbox(lines=2, placeholder="Enter a news headline here..."),
    outputs="text",
    title="Fake News Detection",
    description="Enter a news headline to check if it is real or fake."
)

# Launch the app
iface.launch()
```

```
It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio app to work, sharing must be enabled. Automatically

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://d2de5377d05f537925.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working dir
```



Fake News Detection

Enter a news headline to check if it is real or fake.