```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
data = pd.read_csv('/content/news.csv')
```

```
data.head()
```

	Unnamed: 0	**title**	**text**	**label**
0	8476	You Can Smell Hillary’s Fear	Daniel Greenfield, a Shillman Journalism Fello...	FAKE
1	10294	Watch The Exact Moment Paul Ryan Committed Pol...	Google Pinterest Digg Linkedin Reddit Stumbleu...	FAKE
2	3608	Kerry to go to Paris in gesture of sympathy	U.S. Secretary of State John F. Kerry said Mon...	REAL
3	10142	Bernie supporters on Twitter erupt in anger ag...	— Kaydee King (@KaydeeKing) November 9, 2016 T...	FAKE
4	875	The Battle of New York: Why This Primary Matters	It's primary day in New York and front-runners...	REAL

```
data.shape
```

```
(6335, 4)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6335 entries, 0 to 6334
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   Unnamed: 0  6335 non-null   int64 
 1   title       6335 non-null   object
 2   text        6335 non-null   object
 3   label       6335 non-null   object
dtypes: int64(1), object(3)
memory usage: 198.1+ KB
```

```
data.columns
```

```
Index(['Unnamed: 0', 'title', 'text', 'label'], dtype='object')
```

Task

Preprocess the dataset "news.csv" for a fake news detection task. This includes cleaning the text data, tokenizing it, encoding the labels, and splitting the data into training and testing sets.

Text preprocessing

Subtask:

Clean the text data by removing stopwords, punctuation, and special characters.

Reasoning: Define a function to clean the text data and apply it to the 'text' column.

```
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

nltk.download('punkt')
nltk.download('stopwords')

stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    tokens = word_tokenize(text)
    cleaned_tokens = [word for word in tokens if word not in stop_words]
    return ' '.join(cleaned_tokens)

data['cleaned_text'] = data['text'].apply(clean_text)
data.head()
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

	Unnamed: 0	title	text	label	cleaned_text
0	8476	You Can Smell Hillary’s Fear	Daniel Greenfield, a Shillman Journalism Fello...	FAKE	daniel greenfield shillman journalism fellow f...
1	10294	Watch The Exact Moment Paul Ryan Committed Pol...	Google Pinterest Digg Linkedin Reddit Stumbleu...	FAKE	google pinterest digg linkedin reddit stumbleu...
2	3608	Kerry to go to Paris in gesture of sympathy	U.S. Secretary of State John F. Kerry said Mon...	REAL	us secretary state john f kerry said monday st...
3	10142	Bernie supporters on Twitter erupt in anger ag...	— Kaydee King (@KaydeeKing) November 9, 2016 T...	FAKE	kaydee king kaydeeking november lesson tonight...
4	875	The Battle of New York: Why This Primary Matters	It's primary day in New York and front-runners...	REAL	primary day new york frontrunners hillary clin...

Reasoning: The previous attempt failed because 'punkt_tab' was not found. Download the required resource and then apply the cleaning function again.

```
import nltk
nltk.download('punkt_tab')

import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    tokens = word_tokenize(text)
    cleaned_tokens = [word for word in tokens if word not in stop_words]
    return ' '.join(cleaned_tokens)
```

```
data['cleaned_text'] = data['text'].apply(clean_text)
data.head()
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

	Unnamed: 0	title	text	label	cleaned_text
0	8476	You Can Smell Hillary's Fear	Daniel Greenfield, a Shillman Journalism Fello...	FAKE	daniel greenfield shillman journalism fellow f...
1	10294	Watch The Exact Moment Paul Ryan Committed Pol...	Google Pinterest Digg Linkedin Reddit Stumbleu...	FAKE	google pinterest digg linkedin reddit stumbleu...
2	3608	Kerry to go to Paris in gesture of sympathy	U.S. Secretary of State John F. Kerry said Mon...	REAL	us secretary state john f kerry said monday st...
3	10142	Bernie supporters on Twitter erupt in anger ag...	— Kaydee King (@KaydeeKing) November 9, 2016 T...	FAKE	kaydee king kaydeeking november lesson tonight...
4	875	The Battle of New York: Why This Primary Matters	It's primary day in New York and front-runners...	REAL	primary day new york frontrunners hillary clin...

Tokenization

Subtask:

Convert the cleaned text data into a numerical format using tokenization.

Reasoning: Convert the cleaned text data into a numerical format using tokenization.

```
from tensorflow.keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer()
tokenizer.fit_on_texts(data['cleaned_text'])
tokenized_text = tokenizer.texts_to_sequences(data['cleaned_text'])
```

Label encoding

Subtask:

Convert the 'FAKE' and 'REAL' labels into binary values (0 and 1).

Reasoning: Convert the 'FAKE' and 'REAL' labels into binary values (0 and 1) and store them in a new column named 'encoded_label'.

```
data['encoded_label'] = data['label'].map({'FAKE': 0, 'REAL': 1})
display(data.head())
```

	Unnamed: 0	title	text	label	cleaned_text	encoded_label
0	8476	You Can Smell Hillary’s Fear	Daniel Greenfield, a Shillman Journalism Fello...	FAKE	daniel greenfield shillman journalism fellow f...	0
1	10294	Watch The Exact Moment Paul Ryan Committed Pol...	Google Pinterest Digg Linkedin Reddit Stumbleu...	FAKE	google pinterest digg linkedin reddit stumbleu...	0
2	3608	Kerry to go to Paris in gesture of sympathy	U.S. Secretary of State John F. Kerry said Mon...	REAL	us secretary state john f kerry said monday st...	1
3	10142	Bernie supporters on Twitter erupt in anger ag...	— Kaydee King (@KaydeeKing) November 9, 2016 T...	FAKE	kaydee king kaydeeking november lesson tonight...	0
4	875	The Battle of New York: Why This Primary Matters	It's primary day in New York and front-runners...	REAL	primary day new york frontrunners hillary clin...	1

Data splitting

Subtask:

Split the preprocessed dataset into training and testing sets.

Reasoning: Split the tokenized text data and the encoded labels into training and testing sets.

```
X_train, X_test, y_train, y_test = train_test_split(tokenized_text, data['encoded_label'
```

Summary:

Data Analysis Key Findings

- Text data was cleaned by removing stopwords, punctuation, and special characters, storing the result in a new column `cleaned_text`.
- The cleaned text data was tokenized into sequences of integers using `tensorflow.keras.preprocessing.text.Tokenizer`.
- The categorical labels 'FAKE' and 'REAL' were successfully encoded into binary values 0 and 1, respectively, and stored in the `encoded_label` column.
- The preprocessed data was split into training and testing sets with a test size of 20%, resulting in `X_train`, `X_test`, `y_train`, and `y_test`.

Insights or Next Steps

- The preprocessed data is now ready for training a machine learning model for fake news detection.
- Padding the tokenized sequences to a fixed length would be a necessary next step before feeding the data into a neural network model.

Task

Generate word embeddings for the text data in the loaded dataset using either Word2Vec or TF-IDF, and prepare the data for a deep learning model.

Choose embedding method

Subtask:

Decide whether to use Word2Vec or TF-IDF for generating word embeddings.

Generate embeddings

Subtask:

Apply the chosen method to the cleaned text data to create word embeddings.

Reasoning: Apply the chosen method (Word2Vec) to the cleaned text data to create word embeddings by tokenizing the cleaned text, training a Word2Vec model, and generating document embeddings by averaging word embeddings.

Reasoning: The previous command failed because the `gensim` library was not installed. Install the `gensim` library and then re-execute the code to generate word embeddings.

```
!pip install gensim
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: numpy<2.0,>=1.18.5 in /usr/local/lib/python3.1
Requirement already satisfied: scipy<1.14.0,>=1.7.0 in /usr/local/lib/python3.
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.12
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/dist-packag
```

Reasoning: Now that gensim is installed, re-execute the code to generate word embeddings for the cleaned text data.

Reasoning: The previous command failed because the `data` DataFrame was not available in the current session. Since the data was loaded and cleaned in previous cells, the DataFrame should be available. Re-executing the same code block should work as the environment should now have the `data` DataFrame loaded.

```
from gensim.models import Word2Vec

tokenized_cleaned_text = [text.split() for text in data['cleaned_text']]

word2vec_model = Word2Vec(sentences=tokenized_cleaned_text, vector_size=100, window=5, m

def get_document_embedding(text_tokens, model):
    embeddings = [model.wv[word] for word in text_tokens if word in model.wv]
    if embeddings:
        return np.mean(embeddings, axis=0)
    else:
        return np.zeros(model.vector_size)

document_embeddings = [get_document_embedding(tokens, word2vec_model) for tokens in toke
```

Prepare data for model

Subtask:

Convert the word embeddings into a format suitable for a deep learning model, which may involve padding sequences.

Reasoning: Convert the document embeddings list to a NumPy array and ensure its shape is suitable for a deep learning model input.

```
document_embeddings_array = np.array(document_embeddings)
print(document_embeddings_array.shape)
```

```
(6335, 100)
```

Summary:

Data Analysis Key Findings

- Word2Vec was chosen over TF-IDF for generating word embeddings due to its ability to capture semantic relationships, which is considered more beneficial for fake news detection.
- Word2Vec embeddings with a dimensionality of 100 were successfully generated for each document by averaging the word vectors within the document.
- The generated document embeddings were converted into a NumPy array with a shape of (6335, 100), making them suitable for direct input into a deep learning model.

Insights or Next Steps

- The document embeddings are now ready to be used as features for training a deep learning model for fake news detection.
- Consider exploring different Word2Vec parameters (e.g., vector size, window size) or alternative embedding techniques like GloVe or FastText in future iterations to potentially improve model performance.

Task

Define and compile a Sequential Deep Learning Model using TensorFlow with an Embedding layer, LSTM layer, and Dense layers for binary classification, and prepare the data for training.

Define model architecture

Subtask:

Define the Sequential Deep Learning Model with an Embedding layer, LSTM layer, and Dense layers.

Reasoning: Define the Sequential Deep Learning Model with an Embedding layer, LSTM layer, and Dense layers.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences

vocab_size = len(tokenizer.word_index) + 1
max_sequence_length = max(len(seq) for seq in tokenized_cleaned_text)

model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=100, input_length=max_sequence_leng
model.add(LSTM(128))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ embedding (Embedding)           │ ?                      │   0 (unbuilt) │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ lstm (LSTM)                     │ ?                      │   0 (unbuilt) │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ ?                      │   0 (unbuilt) │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ ?                      │   0 (unbuilt) │
└─────────────────────────────────┴────────────────────────┴───────────────┘

 Total params: 0 (0.00 B)
 Trainable params: 0 (0.00 B)
 Non-trainable params: 0 (0.00 B)
```

Compile model

Subtask:

Compile the model by specifying the optimizer, loss function, and metrics.

Reasoning: Compile the defined Sequential model by specifying the optimizer, loss function, and metrics as instructed.

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Prepare for training

Subtask:

Prepare the data for training the model, potentially involving padding the sequences.

Reasoning: Pad the training and testing sequences and convert them and the labels to NumPy arrays of type float32 for model training.

```
X_train_padded = pad_sequences(X_train, maxlen=max_sequence_length, padding='post')
X_test_padded = pad_sequences(X_test, maxlen=max_sequence_length, padding='post')

X_train_padded = np.array(X_train_padded).astype('float32')
X_test_padded = np.array(X_test_padded).astype('float32')
y_train = np.array(y_train).astype('float32')
y_test = np.array(y_test).astype('float32')
```

Summary:

Data Analysis Key Findings

- A Sequential Deep Learning Model was successfully defined with an Embedding layer, an LSTM layer, and two Dense layers for binary classification.
- The model was compiled using the 'adam' optimizer, 'binary_crossentropy' loss function, and 'accuracy' as the evaluation metric.
- The training and testing input sequences were padded to a uniform length using the 'post' padding strategy and converted to 'float32' NumPy arrays, along with the target labels.

Insights or Next Steps

- The defined and compiled model is now ready to be trained using the prepared padded sequence data and corresponding labels.
- After training, the model's performance on the test set should be evaluated using metrics like accuracy and loss to assess its effectiveness in binary classification.

Task

Evaluate the trained deep learning model for fake news detection using accuracy, precision, recall, and F1-score.

Evaluate model

Subtask:

Evaluate the trained model on the test set.

Reasoning: Evaluate the trained deep learning model on the test set to get the loss and accuracy.

```
loss, accuracy = model.evaluate(X_test_padded, y_test)
print(f'Test Loss: {loss:.4f}')
print(f'Test Accuracy: {accuracy:.4f}')
```

```
40/40 ━━━━━━━━━━━━━━━━━━━━ 167s 4s/step - accuracy: 0.5136 - loss: 0.6931
Test Loss: 0.6932
Test Accuracy: 0.4957
```

Make predictions

Subtask:

Use the trained model to make predictions on the test set.

Reasoning: Use the trained model to make predictions on the test set and convert the probability predictions into binary class predictions.

```
y_prob = model.predict(X_test_padded)
y_pred = (y_prob > 0.5).astype('int32')
```

```
40/40 ━━━━━━━━━━━━━━━━━━━━ 164s 4s/step
```

Calculate metrics

Subtask:

Calculate accuracy, precision, recall, and F1-score based on the predictions and actual labels.

Reasoning: Calculate and print the accuracy, precision, recall, and F1-score based on the predicted and actual labels.

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

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1-score: {f1:.4f}')
```

```
Accuracy: 0.4957
Precision: 0.0000
```

```
Recall: 0.0000
F1-score: 0.0000
```

Summarize results

Subtask:

Summarize the evaluation results.

Summary:

Data Analysis Key Findings

- The model achieved a test accuracy of approximately 49.57% and a test loss of 0.6932.
- The precision, recall, and F1-score for the positive class (assuming it's fake news) are all 0.0000.

Insights or Next Steps

- The current model's performance is no better than random guessing for this binary classification task. The zero values for precision, recall, and F1-score suggest a significant issue, potentially class imbalance or a model that consistently predicts the negative class.
- Investigate potential reasons for the poor performance, including class distribution in the training data, model architecture complexity, training parameters, and data preprocessing steps. Consider techniques like handling class imbalance, using different model architectures, or hyperparameter tuning.

Task

Test the trained fake news detection model on new, unseen news articles.

Provide new data

Subtask:

Get the new unseen news articles you want to test the model on.

Reasoning: Create a list containing the text of new, unseen news articles to test the trained model.

```
new_articles = [
    "Breaking news: Scientists discover a new planet capable of supporting life.",
    "Fake news alert: A celebrity claims to have a cure for cancer that is being suppres
    "Analysis of recent economic trends shows steady growth.",
    "Satirical article: Talking dogs elected to local government in small town.",
    "Expert opinion on the impact of climate change on global weather patterns.",
    "Misleading headline: 'Doctors hate this one weird trick for losing weight.'"
]
```

Preprocess new data

Subtask:

Apply the same preprocessing steps (cleaning, tokenization, padding) to the new data as you did for the training data.

Reasoning: Apply the same preprocessing steps (cleaning, tokenization, padding) to the new data as you did for the training data.

```
_new_articles = [clean_text(article) for article in new_articles]
d_new_articles = tokenizer.texts_to_sequences(cleaned_new_articles)
dded = pad_sequences(tokenized_new_articles, maxlen=max_sequence_length, padding='post')
dded = np.array(X_new_padded).astype('float32')
_new_padded.shape)
```

```
(6, 10371)
```

Make predictions

Subtask:

Use the trained model to predict whether each new article is fake or real.

Reasoning: Use the trained model to make predictions on the preprocessed new data and convert the probabilities to binary predictions.

```
y_new_prob = model.predict(X_new_padded)
y_new_pred = (y_new_prob > 0.5).astype('int32')
print(y_new_pred)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
[[0]
 [0]
 [0]
 [0]
 [0]
 [0]]
```

Interpret results

Subtask:

Examine the model's predictions for the new articles.

Reasoning: Create a list of tuples containing the original new article text and its predicted label, then iterate through the list and print each original article with its predicted label.

```
predicted_articles = []
for i in range(len(new_articles)):
    predicted_articles.append((new_articles[i], y_new_pred[i][0]))

for article, label in predicted_articles:
    print(f"Original Article: {article}")
    print(f"Predicted Label: {'REAL' if label == 1 else 'FAKE'}")
    print("-" * 50)
```

```
nal Article: Breaking news: Scientists discover a new planet capable of suppor
cted Label: FAKE
--------------------------------------------
nal Article: Fake news alert: A celebrity claims to have a cure for cancer tha
cted Label: FAKE
--------------------------------------------
nal Article: Analysis of recent economic trends shows steady growth.
cted Label: FAKE
--------------------------------------------
nal Article: Satirical article: Talking dogs elected to local government in sm
cted Label: FAKE
--------------------------------------------
nal Article: Expert opinion on the impact of climate change on global weather
cted Label: FAKE
--------------------------------------------
nal Article: Misleading headline: 'Doctors hate this one weird trick for losin
cted Label: FAKE
--------------------------------------------
```

Summary:

Data Analysis Key Findings

- A list of six new, unseen news articles was successfully created.
- The new articles were preprocessed using the same steps as the training data, including cleaning, tokenization, and padding to a maximum sequence length of 10371. The resulting padded data had a shape of (6, 10371).
- The trained fake news detection model was used to predict the probability of each article being real. These probabilities were then converted to binary predictions (0 for fake, 1 for real) using a threshold of 0.5.

- The model predicted all six of the new articles as "FAKE".

Insights or Next Steps

- The model's prediction of "FAKE" for all articles, including seemingly real news and analysis, suggests that the model may not be performing accurately on this new data or could be overfitted.
- Further analysis of the model's performance on a larger, more diverse test set with known labels is needed to properly evaluate its effectiveness.