```
import pandas as pd

# Load the uploaded dataset
df= pd.read_csv("fake_reviews_dataset.csv")

# Display the first few rows of the dataset for inspection
df.head()
```

	category	rating	text	label
0	Home_and_Kitchen	5.0	Love this! Well made, sturdy, and very comfor...	1
1	Home_and_Kitchen	5.0	love it, a great upgrade from the original. I...	1
2	Home_and_Kitchen	5.0	This pillow saved my back. I love the look and...	1
3	Home_and_Kitchen	1.0	Missing information on how to use it, but it i...	1
4	Home_and_Kitchen	5.0	Very nice set. Good quality. We have had the s...	1

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

```
#ANOTHERR


import re
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer


# Download necessary NLTK data files if you haven't
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

# Initialize the Lemmatizer and stopwords list
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    # Lowercasing
    text = text.lower()

    # Remove special characters and numbers (optional depending on the context)
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    # Tokenize text
    words = text.split()

    # Remove stop words and apply lemmatization
    words = [lemmatizer.lemmatize(word) for word in words if word not in stop_words]

    return ' '.join(words)
df['processed_text'] = df['text'].apply(preprocess_text)

# Show the original and processed text side by side
print(df[['text', 'processed_text']].head())
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
                                                text  \
0  Love this!  Well made, sturdy, and very comfor...
1  love it, a great upgrade from the original.  I...
2  This pillow saved my back. I love the look and...
3  Missing information on how to use it, but it i...
4  Very nice set. Good quality. We have had the s...

                                      processed_text
0  love well made sturdy comfortable love itvery ...
1   love great upgrade original ive mine couple year
2            pillow saved back love look feel pillow
3        missing information use great product price
4                nice set good quality set two month
```

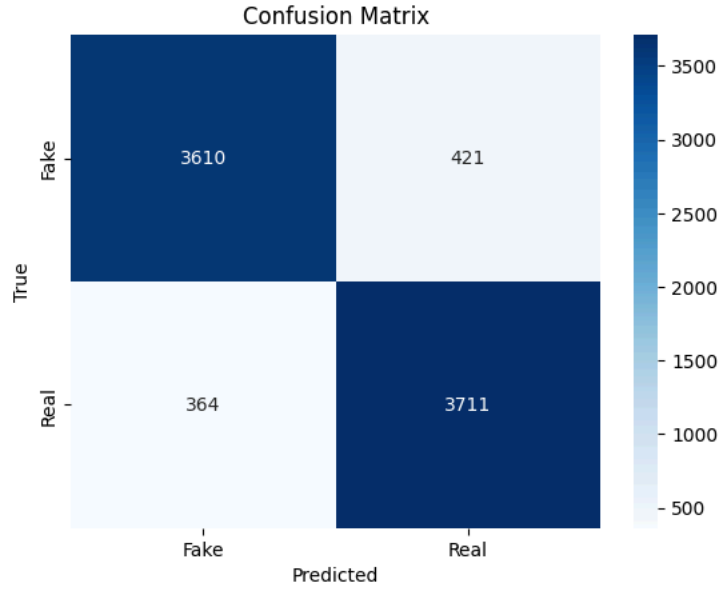
```
# Initialize the TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=5000, ngram_range=(1, 2))

# Apply preprocessing to the reviews and transform using TF-IDF
processed_reviews = df['text'].apply(preprocess_text)
X = tfidf_vectorizer.fit_transform(processed_reviews)


from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.regularizers import l2
from keras.callbacks import EarlyStopping
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

# Assuming df is already loaded and has columns 'text' and 'label'

# Apply TF-IDF vectorization to the reviews
tfidf_vectorizer = TfidfVectorizer(max_features=5000, ngram_range=(1, 2))  # You can tweak ngram_range here if necessary
X_tfidf = tfidf_vectorizer.fit_transform(df['text'])  # Replace 'text' with your text column
y = df['label']  # Replace 'label' with your label column

# Split data into training and validation sets
X_train_tfidf, X_val_tfidf, y_train, y_val = train_test_split(X_tfidf, y, test_size=0.2, random_state=42)

# Build the model
model = Sequential()

# ---1. Input Layer with L2 Regularization and Dropout (to avoid overfitting)---
model.add(Dense(512, activation='relu', input_dim=X_tfidf.shape[1], kernel_regularizer=l2(0.01)))  # Added L2 regularization
model.add(Dropout(0.5))  # Dropout regularization to avoid overfitting

# ---2. Hidden Layer with L2 Regularization and Dropout (to avoid overfitting)---
model.add(Dense(256, activation='relu', kernel_regularizer=l2(0.01)))  # Added L2 regularization
model.add(Dropout(0.5))  # Dropout regularization to avoid overfitting

# Output layer (no changes needed here for overfitting)
model.add(Dense(1, activation='sigmoid'))  # Binary classification

# ---3. Compile the model---
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# ---4. Early Stopping to Avoid Overfitting (stop training when no improvement)---
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

# ---5. Train the model with early stopping to avoid overfitting---
model.fit(X_train_tfidf, y_train, epochs=5, batch_size=32, validation_data=(X_val_tfidf, y_val), callbacks=[early_stopping])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` arg
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/5
1014/1014 ━━━━━━━━━━━━━━━━━━━━ 52s 50ms/step - accuracy: 0.7937 - loss: 1.5170 - val_accuracy: 0.8897 - val_loss: 0.6090
Epoch 2/5
1014/1014 ━━━━━━━━━━━━━━━━━━━━ 80s 48ms/step - accuracy: 0.8821 - loss: 0.6045 - val_accuracy: 0.8979 - val_loss: 0.5595
Epoch 3/5
1014/1014 ━━━━━━━━━━━━━━━━━━━━ 83s 49ms/step - accuracy: 0.8892 - loss: 0.5695 - val_accuracy: 0.8959 - val_loss: 0.5456
Epoch 4/5
1014/1014 ━━━━━━━━━━━━━━━━━━━━ 48s 47ms/step - accuracy: 0.8910 - loss: 0.5495 - val_accuracy: 0.9033 - val_loss: 0.5214
Epoch 5/5
1014/1014 ━━━━━━━━━━━━━━━━━━━━ 51s 50ms/step - accuracy: 0.8909 - loss: 0.5384 - val_accuracy: 0.9032 - val_loss: 0.5034
<keras.src.callbacks.history.History at 0x7e712eb6e9b0>
```

```
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Make predictions on the validation set
y_pred = (model.predict(X_val_tfidf) > 0.5).astype(int)  # Convert probabilities to binary labels

# Confusion Matrix
cm = confusion_matrix(y_val, y_pred)

# Plot confusion matrix
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Fake", "Real"], yticklabels=["Fake", "Real"])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

# Classification Report
print("Classification Report")
```

```
print(classification_report(y_val, y_pred, target_names=['Fake', 'Real']))
```

```
254/254 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step
```




```
Classification Report
              precision    recall  f1-score   support

        Fake       0.91      0.90      0.90      4031
        Real       0.90      0.91      0.90      4075

    accuracy                           0.90      8106
   macro avg       0.90      0.90      0.90      8106
weighted avg       0.90      0.90      0.90      8106
```

```
# Vectorize the entire dataset (not just X_test)
X_all_tfidf = tfidf_vectorizer.transform(df['text']).toarray()

# Predict using the trained model
all_predictions = model.predict(X_all_tfidf)

# Map predictions to 'Real' or 'Fake'
df['predicted_label'] = ['Real' if pred > 0.5 else 'Fake' for pred in all_predictions]

# Save the output to a CSV file with the updated label
df.to_csv('predicted_reviews_all_with_labels.csv', index=False)
```

```
1267/1267 ━━━━━━━━━━━━━━━━━━━━ 10s 8ms/step
```