Text Similarity Analysis and Categorization

1. **Text Pre-processing**:

   - Converts text to lowercase.
   - Removes punctuation.
   - Removes extra spaces and stop words (using NLTK's English stop word list).

2. **Similarity Calculations**:

   - **Semantic Similarity**: Calculated using Sentence Transformer, which captures the meaning of the sentences and compares them.
   - **Jaccard Similarity**: Compares the intersection and union of words between the two sentences.
   - **Cosine Similarity**: Uses TF-IDF vectorization to calculate the cosine similarity between sentence vectors.
   - **Mean Similarity**: Averages the three similarity scores to provide an overall measure of similarity.

3. **Categorization**:

   - Based on the semantic similarity percentage, sentences are categorized into five classes: "Matched", "Need Review", "Moderate Review", "Significant Review", and "Not Matched."

4. **User Inputs**:

   - **Home**: Provides an overview of the different similarity metrics.
   - **Upload Data**: Allows users to upload an Excel file containing sentence pairs for batch processing.
   - **Manual Input**: Users can enter sentences manually and get the similarity scores and categorizations.

5. **Excel Output**:

   - After processing, the similarity metrics and categorizations are saved in an Excel file that users can download directly.

Text Similarity Analysis and Categorization

Explanation of the provided Streamlit application code, focusing on its functionality and key points:

- **pandas:** For data manipulation and handling Excel files.
- **streamlit:** For creating the web application interface.
- **sentence_transformers:** For semantic similarity computation.
- **sklearn:** For TF-IDF vectorization and cosine similarity.
- **re:** For regular expressions to clean text.
- **nltk:** For natural language processing tasks (like removing stopwords).
- **BytesIO:** For handling file operations in memory without writing to disk.

NLTK Download

```
nltk.download('stopwords')
```

- Downloads the stop words list from NLTK for text pre-processing.

Load Pre-trained Model

```
model = SentenceTransformer('paraphrase-MiniLM-L6-v2')
```

- Loads a pre-trained sentence transformer model for semantic similarity computation.

Text Pre-processing Function:

```
def preprocess_text(text):
    text = text.lower()  # Convert text to lowercase
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    text = re.sub(r'\s+', ' ', text).strip()  # Remove extra spaces
    stop_words = set(stopwords.words('english'))  # Set of stopwords
    text = ' '.join([word for word in text.split() if word not in stop_words])  #
Remove stopwords
    return text
```

Text Similarity Analysis and Categorization

- **Lowercase:** Standardizes text for comparison.
- **Remove Punctuation:** Cleans the text.
- **Remove Extra Spaces:** Ensures clean formatting.
- **Stop word Removal:** Removes common words that do not add meaning.

Similarity Calculation Function:

```
def calculate_similarities(sentence1, sentence2):
    sentence1 = preprocess_text(sentence1)  # Preprocess the first sentence
    sentence2 = preprocess_text(sentence2)  # Preprocess the second sentence
# Semantic Similarity (STS) with SentenceTransformer
    embedding1 = model.encode(sentence1, convert_to_tensor=True)  # Encode first sentence
    embedding2 = model.encode(sentence2, convert_to_tensor=True)  # Encode second
sentence
    semantic_similarity = util.pytorch_cos_sim(embedding1, embedding2).item()
# Jaccard Similarity
    set1 = set(sentence1.split())  # Create a set of words from first sentence
    set2 = set(sentence2.split())  # Create a set of words from second sentence
    jaccard_similarity = len(set1.intersection(set2)) / len(set1.union(set2))  #
Calculate Jaccard similarity
# Cosine Similarity (TF-IDF)
    vectorizer = TfidfVectorizer().fit_transform([sentence1, sentence2])  #
Vectorize sentences
    vectors = vectorizer.toarray()  # Convert to array
    cosine_sim = cosine_similarity(vectors)[0, 1]  # Calculate cosine similarity
    return semantic_similarity, jaccard_similarity, cosine_sim
```

Text Similarity Analysis and Categorization

- **Pre-process Sentences:** Cleans the input sentences.
- **Semantic Similarity:** Uses the transformer model to compute meaning-based similarity.
- **Jaccard Similarity:** Measures the similarity based on shared words.
- **Cosine Similarity:** Uses TF-IDF vectorization for vector-based comparison.

Categorization Function:

```
def categorize_semantic_similarity(percentage):
        # Adjusting thresholds to reflect more logical segmentation
        if percentage >= 86:
            return "Matched"
        elif 70 <= percentage < 85.99:
            return "Need Review"
        elif 50 <= percentage < 70:
            return "Moderate Review"
        elif 25 <= percentage < 50:
            return "Significant Review"
        else:
            return "Not Matched"
```

- **Categorize:** The similarity percentage into logical groups to provide qualitative assessments of similarity.

Text Similarity Analysis and Categorization

Main Function:

```
def main():
    st.title("Text Similarity Analysis and Categorization")  # Title of the app
    # Main layout with two columns
    col1, col2 = st.columns(2)
    with col1:
        st.header("Navigation")  # Navigation header
        options = [
            "Home", "Upload Data", "Manual Input"  # Options for navigation
        choice = st.radio("Go to", options)  # Radio buttons for navigation choice
    if choice == "Home":
        # Home Page Content
        st.markdown("""
        <h2 style='font-size:28px;'>Semantic Similarity</h2>
, unsafe_allow_html=True)  # Displays information about similarity metrics


    elif choice == "Upload Data":
# Upload Excel file
        uploaded_file = st.file_uploader("Choose an Excel file", type="xlsx")
        if uploaded_file:
# Read the Excel file
            df = pd.read_excel(uploaded_file)  # Load data into a DataFrame
# Rename columns based on your file structure
            df.columns = ['Sentence1', 'Sentence2']  # Set expected column names
# Calculate all similarity metrics
            similarities = df.apply(lambda row: calculate_similarities(row['Sentence1'],
row['Sentence2']), axis=1)
            df[['Semantic Similarity', 'Jaccard Similarity', 'Cosine Similarity']] =
pd.DataFrame(similarities.tolist(), index=df.index)
# Convert Semantic Similarity to percentage
            df['Semantic Similarity (%)'] = df['Semantic Similarity'] * 100
# Categorize deviation based on the Semantic Similarity percentage
```

```
        df['Semantic Deviation'] = df['Semantic Similarity
(%)'].apply(categorize_semantic_similarity)
# Calculate the mean of Semantic Similarity, Jaccard Similarity, and Cosine Similarity
        df['Mean Similarity'] = df[['Semantic Similarity', 'Jaccard Similarity', 'Cosine
Similarity']].mean(axis=1)
# Convert Mean Similarity to percentage
        df['Mean Similarity (%)'] = df['Mean Similarity'] * 100
# Format Mean Similarity and Semantic Similarity as percentages with two decimal
places
        df['Mean Similarity (%)'] = df['Mean Similarity (%)'].apply(lambda x: f'{x:.2f}%')
        df['Semantic Similarity (%)'] = df['Semantic Similarity (%)'].apply(lambda x:
f'{x:.2f}%')
# Reorder columns
        df = df[['Sentence1', 'Sentence2', 'Semantic Similarity', 'Jaccard Similarity',
            'Cosine Similarity', 'Mean Similarity', 'Mean Similarity (%)',
            'Semantic Similarity (%)', 'Semantic Deviation']]
# Display the dataframe with similarity scores and deviation
        st.subheader("Similarity Results:")
        st.write(df)  # Show results in a table
# Save the DataFrame to an in-memory buffer
        output = BytesIO()
        df.to_excel(output, index=False, engine='openpyxl')  # Save DataFrame to an Excel
file
        output.seek(0)  # Rewind the buffer
# Download the result as an Excel file
        st.download_button(
            label="Download Result as Excel",
            data=output,
            file_name='semantic_similarity_results_with_deviation.xlsx',
            mime='application/vnd.openxmlformats-officedocument.spreadsheetml.sheet'
        )
    else:
        st.warning("Please upload an Excel file to proceed.")
```

```
    elif choice == "Manual Input":
        st.subheader("Manual Input for Sentence Similarity")
        sentence1 = st.text_input("Enter the first sentence:")  # Input for first sentence
        sentence2 = st.text_input("Enter the second sentence:")  # Input for second sentence
        if st.button("Calculate Similarity"):
            if sentence1 and sentence2:
                # Ensure the input isn't just whitespace
                if sentence1.strip() and sentence2.strip():
                    # Calculate similarity scores
                    semantic_similarity, jaccard_similarity, cosine_similarity_score =
calculate_similarities(sentence1, sentence2)
# Convert to percentage
                    semantic_similarity_pct = semantic_similarity * 100
                    mean_similarity = (semantic_similarity + jaccard_similarity +
cosine_similarity_score) / 3
                    mean_similarity_pct = mean_similarity * 100
# Categorize based on Semantic Similarity percentage
                    deviation_category = categorize_semantic_similarity(semantic_similarity_pct)
# Display results
                    st.write(f"**Semantic Similarity:** {semantic_similarity_pct:.2f}%")
                    st.write(f"**Jaccard Similarity:** {jaccard_similarity:.2f}")
                    st.write(f"**Cosine Similarity:** {cosine_similarity_score:.2f}")
                    st.write(f"**Mean Similarity:** {mean_similarity_pct:.2f}%")
                    st.write(f"**Semantic Deviation Category:** {deviation_category}")
                else:
                    st.warning("Please enter valid sentences (non-whitespace).")
            else:
                st.warning("Please enter both sentences to calculate similarity.")
```

- **Title and Navigation:** Sets up the main title and navigation options.
- **Home Page:** Displays information about semantic similarity.
- **Upload Data:** Allows users to upload an Excel file for batch processing.

- **Data Frame Handling:** Loads and renames columns, processes similarities, and computes metrics.
- **Display and Download:** Shows results and provides an option to download them as an Excel file.
- **Manual Input:** Users can enter sentences manually to calculate similarities in real-time.

Run the Application:

```
if __name__ == "__main__":
    main()  # Execute the main function
```

- This runs the application when the script is executed, allowing the Streamlit interface to launch.

Summary Points:

- **User-Friendly:** Offers both manual input and file upload options for ease of use.
- **Similarity Metrics:** Computes and displays various similarity metrics for comparative analysis.
- **Categorization:** Provides qualitative assessments based on semantic similarity.
- **Downloadable Results:** Allows users to download results in a structured format for further analysis.