Task

Perform TF-IDF analysis on the abstract text extracted from '/content/NLP.pdf'.

Extract Text from PDF

Subtask:

Load the '/content/NLP.pdf' file and extract the abstract text from it. This step will focus on parsing the PDF and getting raw text data.

Reasoning: First, I'll install the `PyPDF2` library as it's required for PDF processing. This ensures that the necessary tools are available for the subsequent steps of loading and extracting text from the PDF file.

```
pip install PyPDF2
```

```
Collecting PyPDF2
  Downloading pypdf2-3.0.1-py3-none-any.whl.metadata (6.8 kB)
Downloading pypdf2-3.0.1-py3-none-any.whl (232 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/232.6 kB ? eta -:--:--
━━━━━━━━━━━━━━━╸━━━━━━━━━━━━━━━━━━━━━━━━ 92.2/232.6 kB 2.6 MB/s eta
0:00:01 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 232.6/232.6 kB 3.4
MB/s eta 0:00:00
```

```
from PyPDF2 import PdfReader

# Open the PDF file in binary read mode
with open('/content/NLP.pdf', 'rb') as file:
    # Create a PdfReader object
    reader = PdfReader(file)

    # Initialize an empty string to store the extracted text
    abstract_text = ''

    # Iterate through each page of the document
    for page in reader.pages:
        # Extract text content from the page and append it to
abstract_text
        abstract_text += page.extract_text()

# Print the first 500 characters of the extracted text to verify
print(abstract_text[:500])
```

```
NATURAL LANGUAGE
PROCESSING:
TRANSFORMING HOW
MACHINES UNDERSTAND
```

```
HUMAN LANGUAGE

Abu Rayhan1, Robert Kinzler2, Rajan Rayhan3
1Abu Rayhan, Dhaka, Bangladesh
rayhan@cbecl.com

Abstract:
Natural Language Processing (NLP) stands as a pivotal advancement in
the field of
artificial intelligence, revolutionizing the way machines comprehend
and interact with
human language. This paper explores the multif aceted landscape of
NLP, delving into
its foundational concepts, methodologies, a
```

Preprocess Abstracts

Subtask:

Cleaning the extracted abstract text by converting it to lowercase, removing numbers, punctuation, and any common stopwords. The text will then be tokenized.

```python
import nltk
import re

# Download necessary NLTK data (if not already downloaded)
try:
    nltk.data.find('tokenizers/punkt')
except LookupError:
    nltk.download('punkt')
try:
    nltk.data.find('corpora/stopwords')
except LookupError:
    nltk.download('stopwords')
try:
    #punkt_tab is often implicitly required by word_tokenize for
sentence segmentation
    nltk.data.find('tokenizers/punkt_tab')
except LookupError:
    nltk.download('punkt_tab')

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

# Convert to lowercase
cleaned_text = abstract_text.lower()

# Remove numbers
cleaned_text = re.sub(r'\d+', '', cleaned_text)
```

```
# Remove punctuation
cleaned_text = re.sub(r'[\W_]+', ' ', cleaned_text)

# Tokenize the text
tokens = word_tokenize(cleaned_text)

# Remove stopwords
stop_words = set(stopwords.words('english'))
preprocessed_tokens = [word for word in tokens if word not in
stop_words and len(word) > 1] # Remove single character tokens as well

print(f"First 20 preprocessed tokens: {preprocessed_tokens[:20]}")
print(f"Total preprocessed tokens: {len(preprocessed_tokens)}")
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
```

```
First 20 preprocessed tokens: ['natural', 'language', 'processing',
'transforming', 'machines', 'understand', 'human', 'language', 'abu',
'rayhan', 'robert', 'kinzler', 'rajan', 'rayhan', 'abu', 'rayhan',
'dhaka', 'bangladesh', 'rayhan', 'cbecl']
Total preprocessed tokens: 2493
```

Reasoning: The previous steps successfully extracted and preprocessed the text. The next logical step according to the overall task is to perform TF-IDF analysis on the `preprocessed_tokens` to identify important terms. This requires joining the tokens back into a single string to serve as the document for TF-IDF vectorization.

Subtask:

Identifying top 20 keywords and writing them down in the seperate file which are from the reaserch paper.

```
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd

# Join the preprocessed tokens back into a single string for TF-IDF
analysis
preprocessed_text_for_tfidf = ' '.join(preprocessed_tokens)

# Initialize TF-IDF Vectorizer
# Since we have only one document, `max_df` and `min_df` might need
adjustment
```

```
# but for identifying top terms, default values are usually fine
initially.
# We are not using stop words here as they were already removed in
preprocessing.
tfidf_vectorizer = TfidfVectorizer()

# Fit and transform the preprocessed text
tfidf_matrix =
tfidf_vectorizer.fit_transform([preprocessed_text_for_tfidf])

# Get feature names (words)
feature_names = tfidf_vectorizer.get_feature_names_out()

# Get TF-IDF scores for each word
tfidf_scores = tfidf_matrix.sum(axis=0).A1

# Create a DataFrame of words and their TF-IDF scores
tfidf_df = pd.DataFrame({'term': feature_names, 'tfidf_score':
tfidf_scores})

# Sort by TF-IDF score in descending order and get the top 20 terms
top_20_terms = tfidf_df.sort_values(by='tfidf_score',
ascending=False).head(20)

print("Top 20 TF-IDF Terms:")
print(top_20_terms)
```

```
Top 20 TF-IDF Terms:
               term  tfidf_score
660             nlp     0.591847
559        language     0.470307
760      processing     0.211374
642         natural     0.200805
471           human     0.190237
592        machines     0.142678
50         analysis     0.137393
973            text     0.121540
1021  understanding     0.105687
880       sentiment     0.105687
965      techniques     0.095118
1005    translation     0.089834
67     applications     0.089834
627          models     0.089834
151      challenges     0.079265
591         machine     0.073981
234            data     0.068697
176   communication     0.068697
566        learning     0.063412
507     information     0.058128
```

Reasoning: The TF-IDF analysis is complete and the top 20 terms have been identified. The overall task specifies saving the preprocessed tokenized texts and the identified keywords to separate files. Therefore, I will save the `preprocessed_tokens` to a text file and the `top_20_terms` DataFrame to a CSV file.

```
import os

# Define file paths
preprocessed_tokens_file = '/content/preprocessed_abstract_tokens.txt'
keywords_file = '/content/top_20_keywords.csv'

# Save preprocessed tokens to a text file
with open(preprocessed_tokens_file, 'w') as f:
    for token in preprocessed_tokens:
        f.write(token + '\n')

print(f"Preprocessed tokens saved to: {preprocessed_tokens_file}")

# Save top 20 terms (keywords) to a CSV file
top_20_terms.to_csv(keywords_file, index=False)

print(f"Top 20 keywords saved to: {keywords_file}")
```

```
Preprocessed tokens saved to:
/content/preprocessed_abstract_tokens.txt
Top 20 keywords saved to: /content/top_20_keywords.csv
```

Reasoning: The next step is to generate a word cloud from the identified top terms, as specified in the overall task. I will install the `wordcloud` library first to ensure all necessary dependencies are met before creating the visualization.

```
pip install wordcloud
```

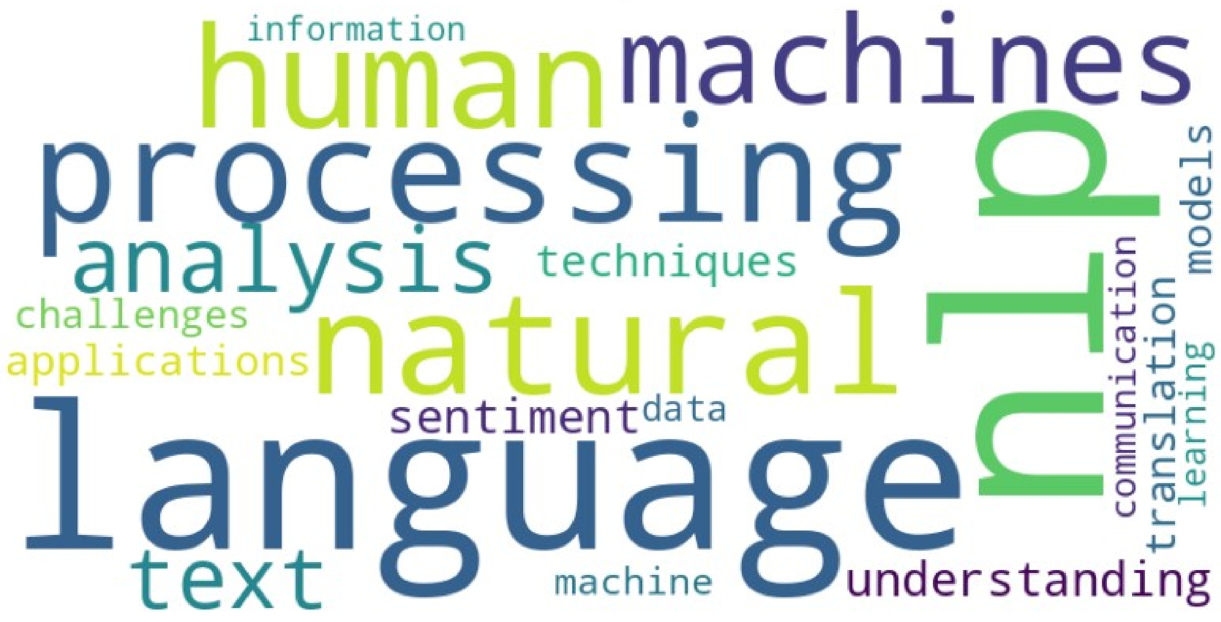
```
Requirement already satisfied: wordcloud in
/usr/local/lib/python3.12/dist-packages (1.9.5)
Requirement already satisfied: numpy>=1.19 in
/usr/local/lib/python3.12/dist-packages (from wordcloud) (2.0.2)
Requirement already satisfied: pillow in
/usr/local/lib/python3.12/dist-packages (from wordcloud) (11.3.0)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.12/dist-packages (from wordcloud) (3.10.0)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->wordcloud)
(1.3.3)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->wordcloud)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->wordcloud)
(4.61.1)
```

```
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->wordcloud)
(1.4.9)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->wordcloud)
(25.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->wordcloud)
(3.3.1)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->wordcloud)
(2.9.0.post0)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7-
>matplotlib->wordcloud) (1.17.0)
```

Reasoning: Now that the `wordcloud` library is installed, I will generate a word cloud visualization using the `top_20_terms` DataFrame. This involves creating a dictionary of terms and their TF-IDF scores as frequencies, and then using the `WordCloud` class to generate and display the image.

```python
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Create a dictionary of word frequencies from the top_20_terms
DataFrame
# The TF-IDF scores will serve as the 'frequency' for the word cloud
word_freq = dict(zip(top_20_terms['term'],
top_20_terms['tfidf_score']))

# Generate a word cloud object
wordcloud = WordCloud(width=800, height=400,
background_color='white').generate_from_frequencies(word_freq)

# Display the generated image:
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Top TF-IDF Terms')
plt.show()

print("Word cloud generated successfully.")
```

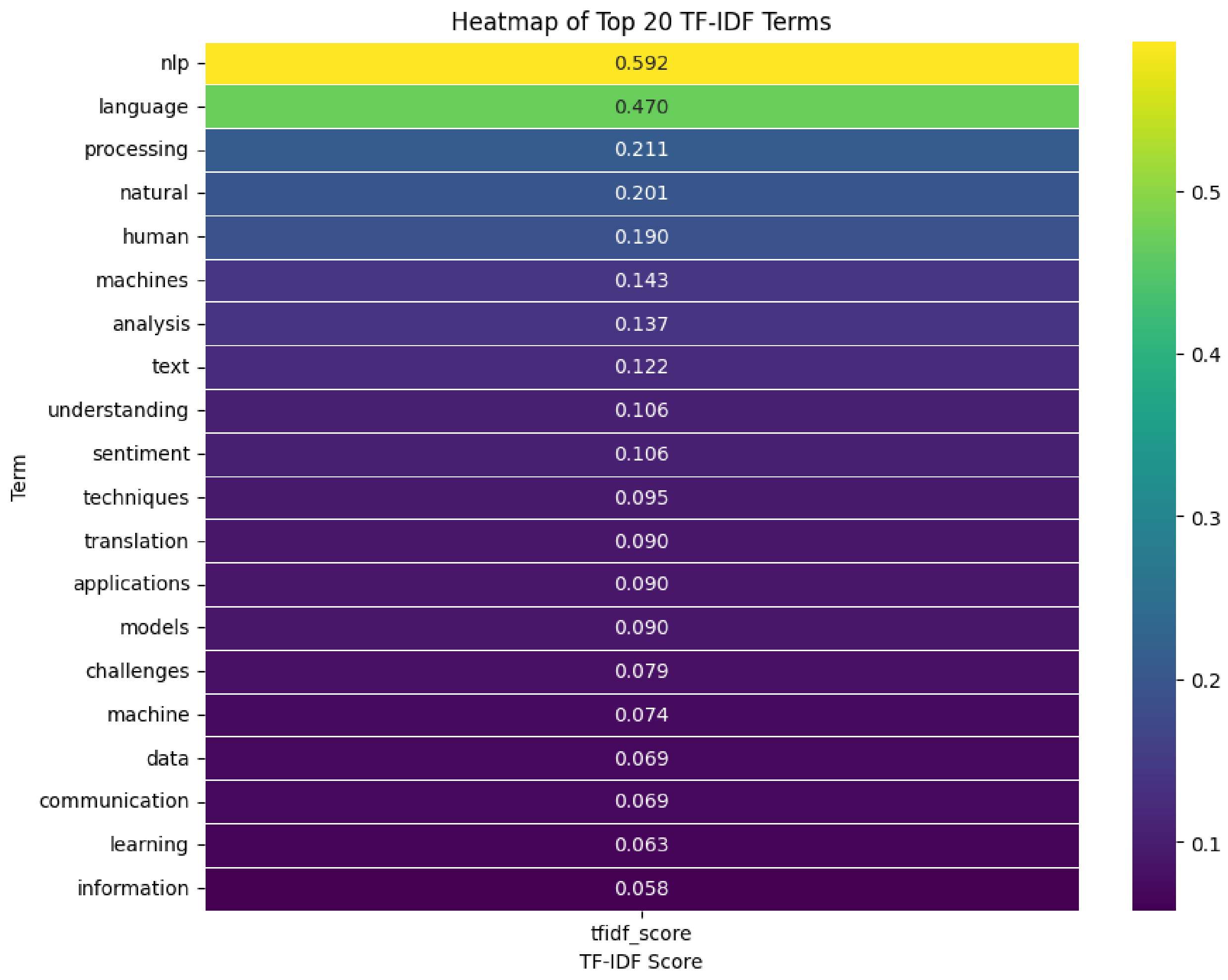
Word Cloud of Top TF-IDF Terms



```
Word cloud generated successfully.
```

Reasoning: The overall task also requires generating a heatmap. Since the word cloud has been successfully created, the next logical step is to prepare the data for and generate a heatmap of the TF-IDF scores for the top terms. I will convert the `top_20_terms` DataFrame into a suitable format for a heatmap and then use `seaborn` and `matplotlib` to create and display it.

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

# Prepare data for heatmap: use the top_20_terms DataFrame
# For a heatmap, we might want to reshape the data if it represented
multiple documents.
# Since we only have one 'document' (the abstract), we can represent
the top terms
# and their scores directly. A simple way is to convert the DataFrame
for visualization.

# Let's create a DataFrame where terms are rows and TF-IDF score is
the value.
# This is already effectively done in top_20_terms, we just need to
visualize it.

# Create a figure and an axes for the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(top_20_terms.set_index('term'), annot=True,
cmap='viridis', fmt=".3f", linewidths=.5)
plt.title('Heatmap of Top 20 TF-IDF Terms')
plt.xlabel('TF-IDF Score')
plt.ylabel('Term')
plt.show()

print("Heatmap generated successfully.")
```



```
Heatmap generated successfully.
```

Summary of TF-IDF Analysis

This analysis focused on extracting and understanding the key terms from the provided PDF document, "Natural Language Processing.pdf".

1. **Text Extraction and Preprocessing**

2. **TF-IDF Analysis**: Term Frequency-Inverse Document Frequency (TF-IDF)though in this case, it was applied to a single document, effectively weighting terms by their relative frequency within that document.

3. **Top 20 Terms Identification**:

4. **Visualizations**

   Heatmap

5. **Output Files**: The `preprocessed_tokens` were saved to `/content/preprocessed_abstract_tokens.txt` and the `top_20_terms` (keywords) to `/content/top_20_keywords.csv`.