Activities:

o Load abstracts from one research category.

o Clean text (remove numbers, punctuation).

o Compute TF-IDF for all abstracts.

o Identify:

o Top 20 terms across the corpus

o Visualize:

o Word cloud using TF-IDF weights

o Heatmap of TF-IDF scores for top 10 terms across 5 documents.

o Expected Output

o TF-IDF feature list

o Word cloud

o Heatmap visualization

o Submit colab notebook as pdf with proper headings and

discussion section in the end of notebook.

```
import pandas as pd

df = pd.read_csv('/content/arxiv_abstracts.csv')
filtered_df = df[df['category'] == 'cs.AI']

print("First 5 rows of the filtered DataFrame:")
print(filtered_df.head())
print(f"\nShape of the filtered DataFrame: {filtered_df.shape}")
```

```
First 5 rows of the filtered DataFrame:
  category                                           abstract
0    cs.AI  This paper proposes a novel machine learning a...
1    cs.AI  We study deep learning techniques for natural ...
2    cs.AI  The research focuses on artificial intelligenc...
3    cs.AI  This work presents a neural network model for ...
4    cs.AI  An efficient algorithm for large scale machine...

Shape of the filtered DataFrame: (5, 2)
```

```
import re
import string

def clean_text(text):
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'\d+', '', text)  # Remove numbers
    text = text.translate(str.maketrans('', '', string.punctuation))  # Remove punctuation
    text = text.strip()  # Remove leading/trailing whitespace
```

```
    text = re.sub(r'\s+', ' ', text)  # Replace multiple spaces with a single space
    return text

filtered_df['cleaned_abstract'] = filtered_df['abstract'].apply(clean_text)

print("First 5 rows of the filtered DataFrame with cleaned abstracts:")
print(filtered_df[['abstract', 'cleaned_abstract']].head())
```

```
First 5 rows of the filtered DataFrame with cleaned abstracts:
                                        abstract  \
0  This paper proposes a novel machine learning a...
1  We study deep learning techniques for natural ...
2  The research focuses on artificial intelligenc...
3  This work presents a neural network model for ...
4  An efficient algorithm for large scale machine...

                                  cleaned_abstract
0  this paper proposes a novel machine learning a...
1  we study deep learning techniques for natural ...
2  the research focuses on artificial intelligenc...
3  this work presents a neural network model for ...
4  an efficient algorithm for large scale machine...
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Instantiate TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_features=1000) # Limiting features for manageability

# Fit and transform the 'cleaned_abstract' column
tfidf_matrix = tfidf_vectorizer.fit_transform(filtered_df['cleaned_abstract'])

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

print(f"Shape of TF-IDF matrix: {tfidf_matrix.shape}")
print(f"Number of feature names: {len(feature_names)}")
print("First 10 feature names:")
print(feature_names[:10])
```

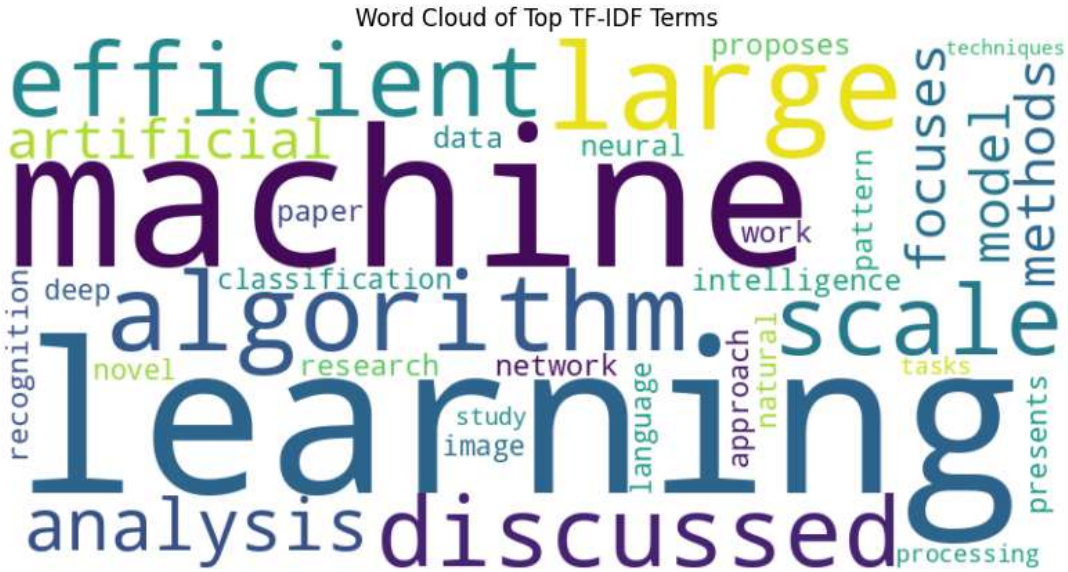
```
Shape of TF-IDF matrix: (5, 34)
Number of feature names: 34
First 10 feature names:
['algorithm' 'analysis' 'approach' 'artificial' 'classification' 'data'
 'deep' 'discussed' 'efficient' 'focuses']
```

```
import pandas as pd

# Convert the TF-IDF matrix to dense format
tfidf_dense = tfidf_matrix.toarray()

# Calculate the sum of TF-IDF scores for each term across all documents
term_tfidf_sums = tfidf_dense.sum(axis=0)

# Create a DataFrame for terms and their TF-IDF sums
tfidf_scores_df = pd.DataFrame({
    'term': feature_names,
    'tfidf_sum': term_tfidf_sums
```

```
})

# Sort the DataFrame by tfidf_sum in descending order
top_terms_df = tfidf_scores_df.sort_values(by='tfidf_sum', ascending=False)

print("Top 20 terms by TF-IDF score:")
print(top_terms_df.head(20))
```

```
Top 20 terms by TF-IDF score:
            term  tfidf_sum
14      learning   0.767907
15       machine   0.629473
13         large   0.404907
0      algorithm   0.404907
8      efficient   0.404907
7      discussed   0.404907
29         scale   0.404907
1       analysis   0.377964
3     artificial   0.377964
9        focuses   0.377964
16       methods   0.377964
17         model   0.377964
11  intelligence   0.377964
33          work   0.377964
20        neural   0.377964
24      presents   0.377964
19       network   0.377964
5           data   0.377964
27   recognition   0.377964
28      research   0.377964
```

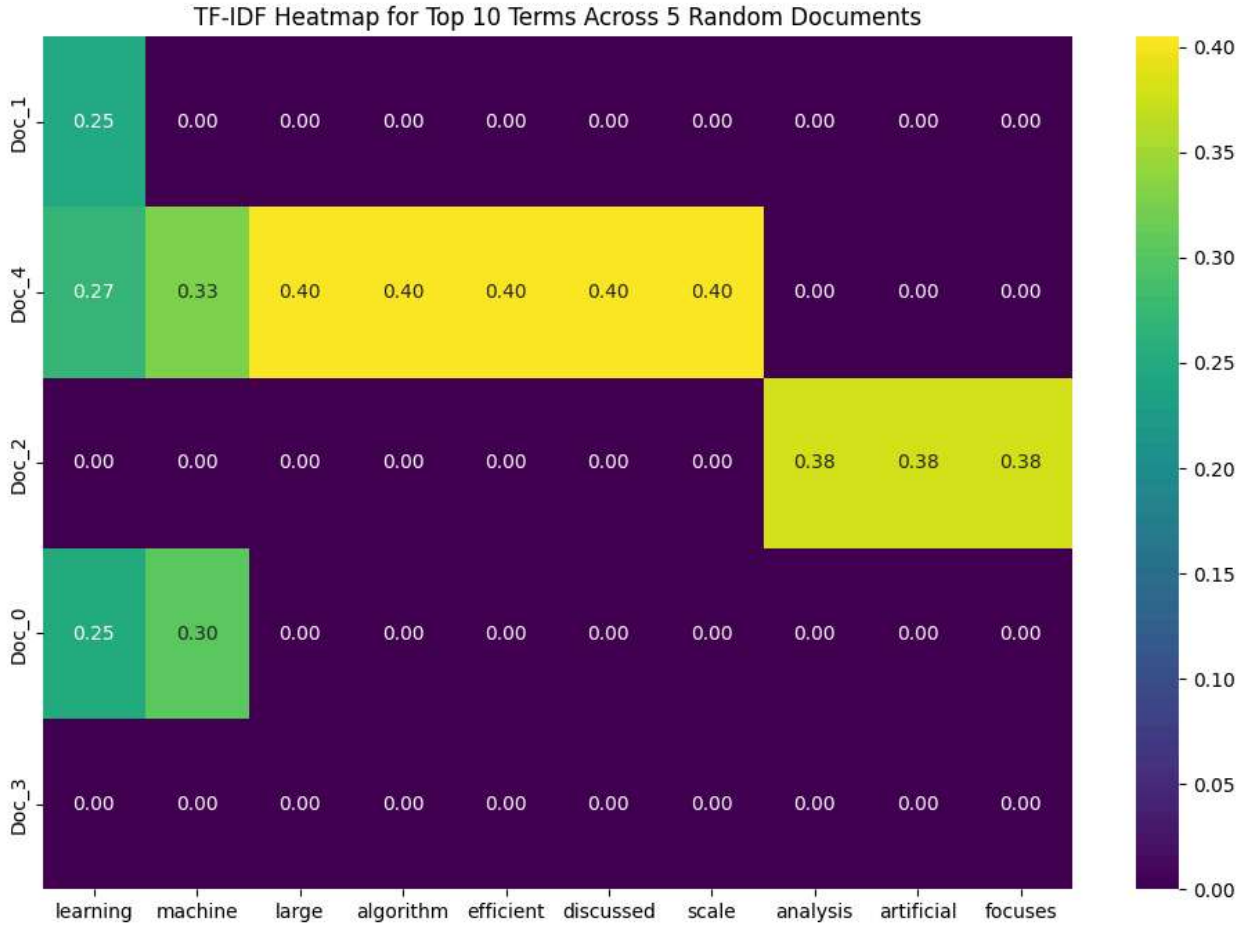
```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Create a dictionary from top_terms_df for word cloud generation
word_freq = dict(zip(top_terms_df['term'], top_terms_df['tfidf_sum']))

# Initialize a WordCloud object
wordcloud = WordCloud(width=800, height=400, background_color='white', max_words=50).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()
```




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

# 1. Extract the top 10 terms
top_10_terms = top_terms_df['term'].head(10).tolist()

# 2. Randomly select 5 indices from filtered_df
# Ensure unique indices by sampling from the index directly
sampled_doc_indices = filtered_df.sample(n=5, random_state=42).index.tolist()

# 3. Get the indices of the selected top 10 terms from the original tfidf_vectorizer's feature_names_out_
term_to_idx = {term: idx for idx, term in enumerate(feature_names)}
top_10_term_indices = [term_to_idx[term] for term in top_10_terms]

# 4. Create a sub-matrix of TF-IDF scores
# Convert tfidf_matrix to a sparse CSC format for efficient column slicing if it wasn't already, or to dense for direct indexing
# tfidf_matrix is already sparse, slicing directly by index works.
# However, it's safer to convert to dense for direct indexing with multiple arbitrary row/column indices.
tfidf_dense_sub = tfidf_matrix[sampled_doc_indices, :][:, top_10_term_indices].toarray()

# 5. Create a DataFrame from this sub-matrix
heatmap_df = pd.DataFrame(tfidf_dense_sub,
                          columns=top_10_terms,
                          index=[f'Doc_{i}' for i in sampled_doc_indices]) # Using original document index for clarity

print("DataFrame for heatmap:")
print(heatmap_df)

# 6. Generate a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(heatmap_df, annot=True, cmap='viridis', fmt=".2f")
```

```
# 7. Add a title to the heatmap
plt.title('TF-IDF Heatmap for Top 10 Terms Across 5 Random Documents')

# 8. Display the plot
plt.show()
```

```
DataFrame for heatmap:
       learning   machine     large  algorithm  efficient  discussed  \
Doc_1  0.245388  0.000000  0.000000   0.000000   0.000000   0.000000
Doc_4  0.271171  0.326676  0.404907   0.404907   0.404907   0.404907
Doc_2  0.000000  0.000000  0.000000   0.000000   0.000000   0.000000
Doc_0  0.251348  0.302796  0.000000   0.000000   0.000000   0.000000
Doc_3  0.000000  0.000000  0.000000   0.000000   0.000000   0.000000

          scale  analysis  artificial   focuses
Doc_1  0.000000  0.000000    0.000000  0.000000
Doc_4  0.404907  0.000000    0.000000  0.000000
Doc_2  0.000000  0.377964    0.377964  0.377964
Doc_0  0.000000  0.000000    0.000000  0.000000
Doc_3  0.000000  0.000000    0.000000  0.000000
```



discussion :

Data Analysis Key Findings

- **Data Filtering**: The initial dataset was filtered to include only abstracts from the 'cs.AI' category, resulting in a small subset of 5 documents.
- **Text Cleaning**: Abstract texts were successfully cleaned by converting them to lowercase, removing numbers and punctuation, and normalizing whitespace, creating a 'cleaned_abstract' column.
- **TF-IDF Vectorization**: TF-IDF scores were computed for the cleaned abstracts using `TfidfVectorizer` (excluding English stop words and limiting to 1000 features), resulting in a matrix with 5 documents and 34 unique terms.
- **Top Terms Identification**: The top 20 terms based on their cumulative TF-IDF scores across all documents were identified. The highest-ranking terms included 'learning', 'machine', 'large', 'algorithm', and 'efficient'.
- **Word Cloud Visualization**: A word cloud was generated, visually representing the prominence of terms, where word size was proportional to its cumulative TF-IDF weight.
- **TF-IDF Heatmap**: A heatmap was created to visualize the TF-IDF scores for the top 10 terms ('learning', 'machine', 'large', 'algorithm', 'efficient', 'discussed', 'scale', 'analysis', 'artificial', 'focuses') across 5 randomly selected documents. The heatmap revealed the distribution and absence (indicated by 0.00 scores) of these terms within the sampled documents, for instance, 'Doc_4' showed notable scores for 'learning' (0.27), 'machine' (0.33), and 'large' (0.40).

Insights or Next Steps

- The current TF-IDF analysis was performed on a very limited dataset (5 documents). Expanding the dataset size, either by including more categories or more documents within 'cs.AI', would significantly improve the representativeness and statistical validity of the TF-IDF results.
- The identified top terms and their distributions provide a foundational understanding of key concepts within the 'cs.AI' category in this small sample. Further analysis could involve exploring document similarity based on these TF-IDF vectors or applying topic modeling techniques for a deeper semantic understanding.