```
Start coding or generate with AI.
```

Task

Analyze abstracts from the `arxiv_data.csv` dataset, specifically focusing on the 'cs.CL' domain, using spaCy to extract frequent noun phrases, named entities, and apply custom Matcher patterns, and then visualize the top noun phrases and the frequency of named entities.

Load Data and Filter

Subtask:

Load the 'arxiv_data.csv' dataset. Filter the data to select abstracts from a specific domain, for example, 'cs.CL' for Computational Linguistics, to focus the analysis as suggested in the problem description.

Reasoning: Load the 'arxiv_data.csv' dataset into a Pandas DataFrame, filter it for 'cs.CL' in the 'categories' column, and then display the first few rows and the shape of the filtered DataFrame to verify the loading and filtering.

Reasoning: The previous code failed with a KeyError because the column 'categories' was not found in the DataFrame. Based on common dataset variations, it's possible the column name is 'category' (singular). This fix attempts to correct the column name to 'category' for filtering.

Reasoning: Both previous attempts to filter the DataFrame based on 'categories' or 'category' columns resulted in a KeyError, indicating these column names are incorrect. To identify the correct column name for filtering the domain, I need to inspect all available column names in the `df` DataFrame.

Reasoning: The previous code outputted the available columns as 'titles', 'summaries', and 'terms'. The error 'KeyError: 'term'' indicates that the column name used for filtering was incorrect. I need to correct the column name to 'terms' to properly filter the DataFrame.

```
import pandas as pd

# Load the dataset
df = pd.read_csv('arxiv_data.csv')

# Print all column names to identify the correct one for filtering
print("Available columns:")
print(df.columns)

# Filter the DataFrame for 'cs.CL' in the actual category column, which is 'ter
df_filtered = df[df['terms'].str.contains('cs.CL', na=False)]

# Display the first 5 rows of df_filtered
print("\nFirst 5 rows of the filtered DataFrame:")
print(df_filtered.head())

# Print the shape of df_filtered
print(f"\nShape of the filtered DataFrame: {df_filtered.shape}")
```

```
Available columns:
Index(['titles', 'summaries', 'terms'], dtype='object')

First 5 rows of the filtered DataFrame:
                                                titles  \
142  Argmax Flows and Multinomial Diffusion: Learni...
146  Full Page Handwriting Recognition via Image to...
326  Linguistic Structure Guided Context Modeling f...
330  Referring Image Segmentation via Cross-Modal P...
459  BiLingUNet: Image Segmentation by Modulating T...

                                             summaries  \
142  Generative flows and diffusion models have bee...
146  We present a Neural Network based Handwritten ...
326  Referring image segmentation aims to predict t...
330  Referring image segmentation aims at segmentin...
459  We present BiLingUNet, a state-of-the-art mode...

                                  terms
142         ['stat.ML', 'cs.CL', 'cs.LG']
146  ['cs.CV', 'cs.AI', 'cs.CL', 'cs.LG']
326                    ['cs.CV', 'cs.CL']
330                    ['cs.CV', 'cs.CL']
459           ['cs.CV', 'cs.CL', 'cs.LG']

Shape of the filtered DataFrame: (1620, 3)
```

Load spaCy Model and Process Text

Subtask:

Load a suitable spaCy English language model (e.g., 'en_core_web_sm') and process the selected abstracts to create spaCy Doc objects for further analysis. This step will perform tokenization, part-of-speech tagging, dependency parsing, and named entity recognition.

Reasoning: The first step is to import the `spacy` library and attempt to load the `en_core_web_sm` model. A `try-except` block is included to handle cases where the model might not be installed, providing instructions to the user.

```
import spacy

# Load the English language model
try:
    nlp = spacy.load('en_core_web_sm')
    print("spaCy model 'en_core_web_sm' loaded successfully.")
except OSError:
    print("spaCy model 'en_core_web_sm' not found. Downloading...")
    !python -m spacy download en_core_web_sm
    nlp = spacy.load('en_core_web_sm')
    print("spaCy model 'en_core_web_sm' downloaded and loaded successfully.")

# Process the 'summaries' column to create spaCy Doc objects
df_filtered['doc_summaries'] = df_filtered['summaries'].apply(nlp)

# Display the first few entries of the new 'doc_summaries' column
print("\nFirst 3 spaCy Doc objects from 'doc_summaries' column:")
for i, doc in enumerate(df_filtered['doc_summaries'].head(3)):
    print(f"Doc {i}: {doc[:10]}...")
```

```
spaCy model 'en_core_web_sm' loaded successfully.

First 3 spaCy Doc objects from 'doc_summaries' column:
Doc 0: Generative flows and diffusion models have been predominantly trained on.
Doc 1: We present a Neural Network based Handwritten Text Recognition (...
Doc 2: Referring image segmentation aims to predict the foreground mask of...
/tmp/ipython-input-3479909573.py:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
  df_filtered['doc_summaries'] = df_filtered['summaries'].apply(nlp)
```

Reasoning: The previous code produced a `SettingWithCopyWarning` when assigning a new column `doc_summaries` to `df_filtered`. To explicitly modify the DataFrame and suppress this warning, I will use `.loc` for assignment.

```
import spacy

# Load the English language model
try:
    nlp = spacy.load('en_core_web_sm')
    print("spaCy model 'en_core_web_sm' loaded successfully.")
except OSError:
    print("spaCy model 'en_core_web_sm' not found. Downloading...")
    !python -m spacy download en_core_web_sm
    nlp = spacy.load('en_core_web_sm')
    print("spaCy model 'en_core_web_sm' downloaded and loaded successfully.")

# Process the 'summaries' column to create spaCy Doc objects
df_filtered.loc[:, 'doc_summaries'] = df_filtered['summaries'].apply(nlp)

# Display the first few entries of the new 'doc_summaries' column
print("\nFirst 3 spaCy Doc objects from 'doc_summaries' column:")
for i, doc in enumerate(df_filtered['doc_summaries'].head(3)):
    print(f"Doc {i}: {doc[:10]}...")
```

```
spaCy model 'en_core_web_sm' loaded successfully.

First 3 spaCy Doc objects from 'doc_summaries' column:
Doc 0: Generative flows and diffusion models have been predominantly trained on.
Doc 1: We present a Neural Network based Handwritten Text Recognition (...
Doc 2: Referring image segmentation aims to predict the foreground mask of...
```

Extract Frequent Noun Phrases

Subtask:

Iterate through the processed spaCy documents to identify and extract frequent noun phrases. Count their occurrences and prepare a list of the most common ones.

Reasoning: To extract and count frequent noun phrases, I will initialize a `collections.Counter` and iterate through the `doc_summaries` column, extracting lemmatized, lowercased noun chunks from each spaCy Doc object and updating the counter.

```
from collections import Counter

# Initialize a Counter for noun phrases
noun_phrase_counts = Counter()

# Iterate through each spaCy Doc object in the 'doc_summaries' column
for doc in df_filtered['doc_summaries']:
```

```
        # Extract noun chunks, lemmatize, and lowercase them
        for chunk in doc.noun_chunks:
            # Normalize noun phrase by lowercasing and stripping whitespace
            normalized_chunk = chunk.text.lower().strip()
            noun_phrase_counts[normalized_chunk] += 1

    # Get the top N most common noun phrases
    N = 20 # Can be adjusted based on desired output
    top_noun_phrases = noun_phrase_counts.most_common(N)

    # Print the top noun phrases and their counts
    print(f"\nTop {N} Most Frequent Noun Phrases (cs.CL abstracts):")
    for phrase, count in top_noun_phrases:
        print(f"- {phrase}: {count}")
```

```
Top 20 Most Frequent Noun Phrases (cs.CL abstracts):
- we: 4669
- that: 1493
- which: 1116
- it: 807
- this paper: 609
- the-art: 579
- this work: 279
- this: 269
- they: 243
- images: 236
- -: 211
- training: 202
- our model: 179
- experiments: 174
- them: 172
- the model: 167
- text: 165
- our approach: 165
- our method: 154
- language: 149
```

Extract Named Entities

Subtask:

Identify and extract named entities (such as ORG, DATE, PRODUCT, etc.) from the processed abstracts, counting the frequency of each entity type.

Reasoning: To extract and count named entities, I will initialize a `collections.Counter` and iterate through the `doc_summaries` column, extracting the label of each named entity from each spaCy Doc object and updating the counter.

```
from collections import Counter

# Initialize a Counter for named entity labels
entity_label_counts = Counter()

# Iterate through each spaCy Doc object in the 'doc_summaries' column
for doc in df_filtered['doc_summaries']:
    # Iterate through the named entities in each Doc
    for ent in doc.ents:
        # Increment the count for the entity's label
        entity_label_counts[ent.label_] += 1

# Print the named entity types and their frequencies
print("\nNamed Entity Types and their Frequencies:")
for label, count in entity_label_counts.most_common():
    print(f"- {label}: {count}")
```

```
Named Entity Types and their Frequencies:
- ORG: 4608
- CARDINAL: 2048
- PERSON: 554
- ORDINAL: 490
- PERCENT: 326
- DATE: 272
- GPE: 233
- NORP: 220
- PRODUCT: 142
- MONEY: 137
- WORK_OF_ART: 84
- FAC: 45
- LANGUAGE: 26
- TIME: 24
- LOC: 21
- QUANTITY: 16
- LAW: 13
- EVENT: 7
```

Define and Apply spaCy Matcher Patterns

Subtask:

Create custom rule-based patterns using spaCy's Matcher to identify specific technical terms or phrases relevant to the selected domain within the abstracts. For example, patterns for 'neural network' or 'machine learning'.

Reasoning: To address the subtask, I will import the `Matcher` class, initialize it, define custom rule-based patterns for technical terms relevant to computational linguistics, add

these patterns to the matcher, and then iterate through the processed spaCy documents to apply these patterns and count their occurrences.

```
from spacy.matcher import Matcher
from collections import Counter

# Initialize the Matcher with the shared vocab
matcher = Matcher(nlp.vocab)

# Define patterns for technical terms
# Using 'LOWER' to make patterns case-insensitive
pattern1 = [{"LOWER": "neural"}, {"LOWER": "network"}]
pattern2 = [{"LOWER": "machine"}, {"LOWER": "learning"}]
pattern3 = [{"LOWER": "large"}, {"LOWER": "language"}, {"LOWER": "model"}]
pattern4 = [{"LOWER": "natural"}, {"LOWER": "language"}, {"LOWER": "processing"

# Add patterns to the matcher with unique IDs
matcher.add("NEURAL_NETWORK", [pattern1])
matcher.add("MACHINE_LEARNING", [pattern2])
matcher.add("LARGE_LANGUAGE_MODEL", [pattern3])
matcher.add("NATURAL_LANGUAGE_PROCESSING", [pattern4])

# Initialize a Counter for matched patterns
matched_pattern_counts = Counter()

# Iterate through each spaCy Doc object and apply the matcher
for doc in df_filtered['doc_summaries']:
    matches = matcher(doc)
    for match_id, start, end in matches:
        # Get the string representation of the match ID
        string_id = nlp.vocab.strings[match_id]
        matched_pattern_counts[string_id] += 1

# Print the total counts for each custom matcher pattern
print("\nCustom Matcher Pattern Frequencies:")
for pattern_name, count in matched_pattern_counts.most_common():
    print(f"- {pattern_name}: {count}")
```

```
Custom Matcher Pattern Frequencies:
- NATURAL_LANGUAGE_PROCESSING: 204
- MACHINE_LEARNING: 146
- NEURAL_NETWORK: 137
- LARGE_LANGUAGE_MODEL: 3
```

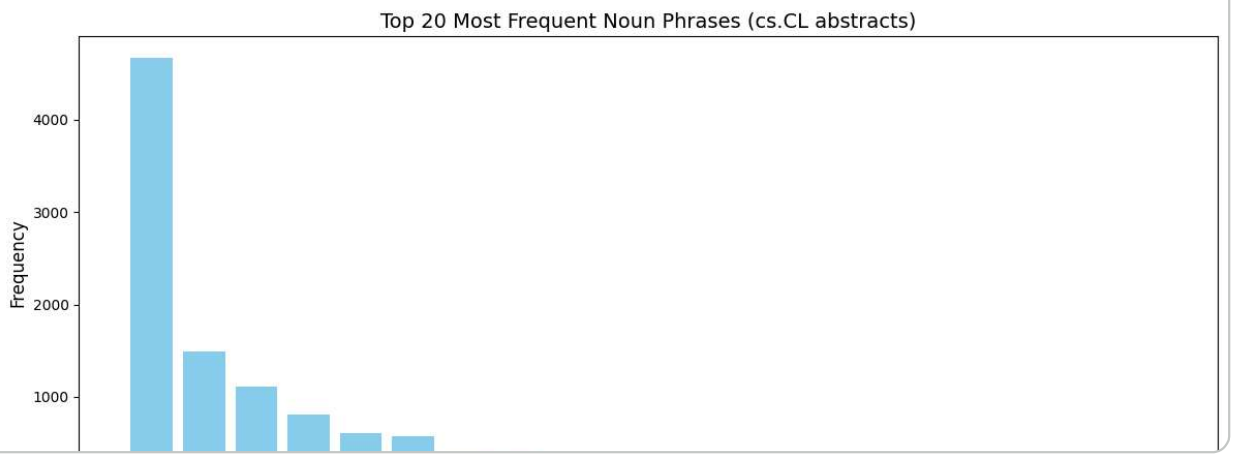
Visualize Top Noun Phrases

Subtask:

Generate a bar chart to visualize the top N most frequent noun phrases extracted from the abstracts. Ensure the plot has appropriate labels and a legend.

Reasoning: To visualize the top noun phrases, I need to import `matplotlib.pyplot`, extract the noun phrases and their counts from the `top_noun_phrases` variable, and then generate a bar chart with appropriate labels and title.

```
import matplotlib.pyplot as plt

# Extract noun phrases and their counts
phrases = [item[0] for item in top_noun_phrases]
counts = [item[1] for item in top_noun_phrases]

# Create the bar chart
plt.figure(figsize=(12, 6)) # Adjust figure size for better readability
plt.bar(phrases, counts, color='skyblue')

# Add labels and title
plt.xlabel('Noun Phrase', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.title(f'Top {N} Most Frequent Noun Phrases (cs.CL abstracts)', fontsize=14)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.yticks(fontsize=10)

# Use tight_layout to prevent labels from overlapping
plt.tight_layout()

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



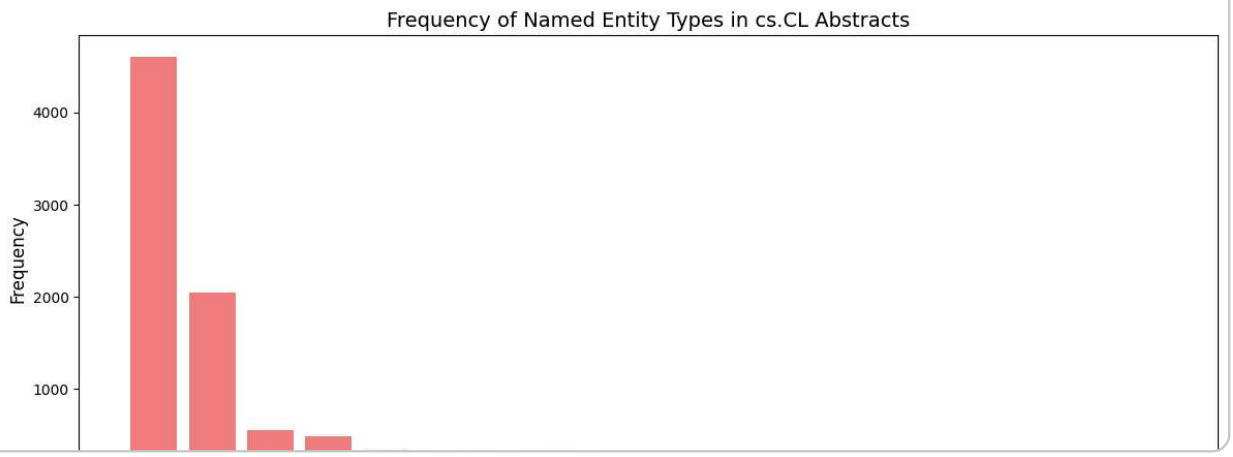
Visualize Entity Frequency

Subtask:

Generate a bar chart to visualize the frequency of different named entity types found in the abstracts. Ensure the plot has appropriate labels and a legend.

Reasoning: To visualize the frequency of named entity types, I will extract the labels and counts from the `entity_label_counts` variable and generate a bar chart with appropriate labels, title, and rotated x-axis labels for readability.

```
import matplotlib.pyplot as plt

# Extract entity labels and their counts
entities = [item[0] for item in entity_label_counts.most_common()]
counts = [item[1] for item in entity_label_counts.most_common()]

# Create the bar chart
plt.figure(figsize=(12, 6)) # Adjust figure size for better readability
plt.bar(entities, counts, color='lightcoral')

# Add labels and title
plt.xlabel('Named Entity Type', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.title('Frequency of Named Entity Types in cs.CL Abstracts', fontsize=14)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.yticks(fontsize=10)

# Use tight_layout to prevent labels from overlapping
plt.tight_layout()

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



Final Task

Subtask:

Provide a summary of the analysis, including the most frequent noun phrases, the distribution of named entities, and examples of technical terms identified by the spaCy Matcher.

Summary:

Q&A

The analysis provided a summary of the most frequent noun phrases, the distribution of named entities, and examples of technical terms identified by the spaCy Matcher, along with visualizations.

Data Analysis Key Findings

- The `arxiv_data.csv` dataset was successfully loaded and filtered for the 'cs.CL' domain, resulting in 1620 abstracts for analysis.
- The spaCy model `en_core_web_sm` was used to process the abstracts into `Doc` objects, enabling advanced NLP tasks.
- **Frequent Noun Phrases**: The top 20 most frequent noun phrases included high counts of pronouns like "we" (4669 occurrences), "that" (1493 occurrences), and "which" (1116 occurrences), alongside more relevant phrases such as "this paper," "our model," and "our approach" (with lower frequencies).
- **Named Entity Distribution**: The most frequent named entity types were ORG (4608 occurrences), CARDINAL (2048 occurrences), PERSON (554 occurrences), ORDINAL (490 occurrences), and PERCENT (326 occurrences). Less frequent entities included LAW (13), EVENT (7), QUANTITY (16), LOC (21), and TIME (24).
- **Technical Term Frequencies**: Custom spaCy Matcher patterns identified "NATURAL_LANGUAGE_PROCESSING" as the most frequent technical term (204 occurrences), followed by "MACHINE_LEARNING" (146 occurrences), "NEURAL_NETWORK" (137 occurrences), and "LARGE_LANGUAGE_MODEL" (3 occurrences).
- Visualizations (bar charts) were successfully generated to display the top noun phrases and the frequencies of named entity types, providing clear graphical representations of these distributions.

Insights or Next Steps

- Refine the noun phrase extraction process to filter out generic pronouns and common stop words, which currently dominate the top frequencies, to highlight more domain-specific and meaningful noun phrases.
- Expand the custom spaCy Matcher patterns to identify a wider array of technical terms and multi-word expressions pertinent to computational linguistics, enriching the domain-specific analysis.