```
# Install spaCy model (run once)
!python -m spacy download en_core_web_sm

# Import libraries
import pandas as pd
import spacy

# Load English model
nlp = spacy.load('en_core_web_sm')

# Load dataset (first 1000 rows for demonstration)
df = pd.read_csv('arxiv_data.csv', engine='python', nrows=1000)

# Preview dataset
print(df['summaries'].head())
```

```
Collecting en-core-web-sm==3.8.0
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-3.8
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.8/12.8 MB 80.7 MB/s eta 0:00:00
✓ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
⚠ Restart to reload dependencies
If you are in a Jupyter or Colab notebook, you may need to restart Python in
order to load all the package's dependencies. You can do this by selecting the
'Restart kernel' or 'Restart runtime' option.
0    Stereo matching is one of the widely used tech...
1    The recent advancements in artificial intellig...
2    In this paper, we proposed a novel mutual cons...
3    Consistency training has proven to be an advan...
4    To ensure safety in automated driving, the cor...
Name: summaries, dtype: object
```

```
# Apply spaCy pipeline to summaries
df['doc'] = df['summaries'].apply(lambda text: nlp(text if isinstance(text, str) else ""))

# Preview
print(df['doc'].head())
```

```
0    (Stereo, matching, is, one, of, the, widely, u...
1    (The, recent, advancements, in, artificial, in...
2    (In, this, paper, ,, we, proposed, a, novel, m...
3    (Consistency, training, has, proven, to, be, a...
4    (To, ensure, safety, in, automated, driving, ,...
Name: doc, dtype: object
```

```
# Tokenize each abstract
df['tokens'] = df['doc'].apply(lambda doc: [token.text for token in doc])

# Preview first 20 tokens of first summary
print(df['tokens'].iloc[0][:20])
```

```
 'inferring', 'depth', 'from', '\n', 'stereo', 'images', 'owing', 'to', 'its', 'robustness']
```

```
# Extract noun phrases
df['noun_phrases'] = df['doc'].apply(lambda doc: [chunk.text for chunk in doc.noun_chunks])

# Preview noun phrases of first abstract
```

```
print(df['noun_phrases'].iloc[0])
```

```
['Stereo matching', 'the widely used techniques', 'depth', 'stereo images', 'its robustness'
```

```
# Function to extract entities with specific labels
def extract_entities(doc, labels=None):
    if labels:
        return [(ent.text, ent.label_) for ent in doc.ents if ent.label_ in labels]
    else:
        return [(ent.text, ent.label_) for ent in doc.ents]

# Example: common entity types
entity_labels = ['ORG', 'DATE', 'PERSON', 'PRODUCT', 'GPE', 'EVENT']
df['entities'] = df['doc'].apply(lambda doc: extract_entities(doc, labels=entity_labels))

# Preview entities in first abstract
print(df['entities'].iloc[0])
```

```
[]
```

```
from spacy.matcher import Matcher

# Initialize matcher
matcher = Matcher(nlp.vocab)

# Define a pattern (ADJ + NOUN + NOUN)
pattern = [{"POS": "ADJ"}, {"POS": "NOUN"}, {"POS": "NOUN"}]
matcher.add("TECH_TERM_PATTERN", [pattern])

# Function to match patterns
def match_technical_terms(doc):
    matches = matcher(doc)
    return [doc[start:end].text for match_id, start, end in matches]

# Apply matcher
df['tech_terms'] = df['doc'].apply(match_technical_terms)

# Preview technical terms in first abstract
print(df['tech_terms'].iloc[0])
```

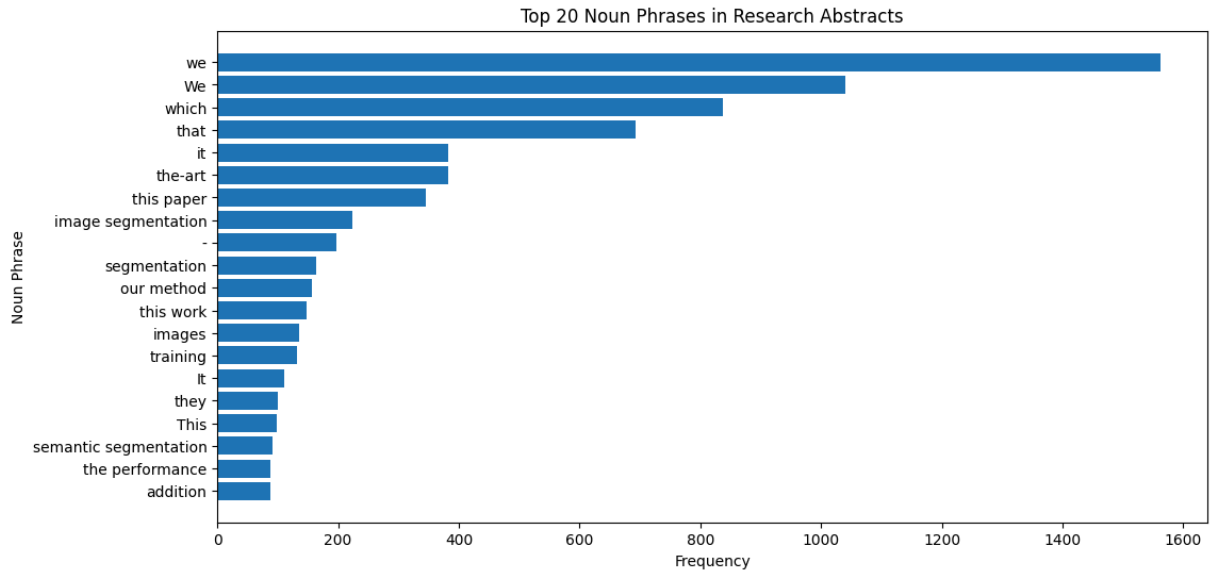
```
['deep neural network']
```

```
from collections import Counter
import matplotlib.pyplot as plt

# Flatten all noun phrases
all_noun_phrases = [np for sublist in df['noun_phrases'] for np in sublist]

# Count top 20 noun phrases
noun_freq = Counter(all_noun_phrases)
top_noun_phrases = noun_freq.most_common(20)

# Plot
plt.figure(figsize=(12,6))
plt.barh([x[0] for x in reversed(top_noun_phrases)], [x[1] for x in reversed(top_noun_phrase
plt.title("Top 20 Noun Phrases in Research Abstracts")
plt.xlabel("Frequency")
plt.ylabel("Noun Phrase")
plt.show()
```




```
# Flatten all entity labels
all_entities = [ent[1] for sublist in df['entities'] for ent in sublist]

# Count frequency
entity_freq = Counter(all_entities)

# Plot
plt.figure(figsize=(8,5))
plt.bar(entity_freq.keys(), entity_freq.values())
plt.title("Named Entity Frequency by Type")
plt.xlabel("Entity Type")
plt.ylabel("Count")
plt.show()
```