```
import pandas as pd

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

# Inspect the dataframe
print(df.head())
print(df.info())
```

```
                                              titles  \
0  Survey on Semantic Stereo Matching / Semantic ...
1  FUTURE-AI: Guiding Principles and Consensus Re...
2  Enforcing Mutual Consistency of Hard Regions f...
3  Parameter Decoupling Strategy for Semi-supervi...
4  Background-Foreground Segmentation for Interio...

                                           summaries  \
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...

                        terms
0          ['cs.CV', 'cs.LG']
1  ['cs.CV', 'cs.AI', 'cs.LG']
2          ['cs.CV', 'cs.AI']
3                   ['cs.CV']
4          ['cs.CV', 'cs.LG']
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51774 entries, 0 to 51773
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   titles     51774 non-null  object
 1   summaries  51774 non-null  object
 2   terms      51774 non-null  object
dtypes: object(3)
memory usage: 1.2+ MB
None
```

```
import pandas as pd
import spacy
from spacy.matcher import Matcher
from collections import Counter
import matplotlib.pyplot as plt
import seaborn as sns

# Load a small sample for demonstration
df_sample = pd.read_csv('arxiv_data.csv').head(500)

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

# 1. Processing text using nlp.pipe for efficiency
# We only need specific components to speed up processing
```

```
# Noun chunks need 'parser', entities need 'ner'
# We'll keep them but use nlp.pipe
print("Processing text...")
docs = list(nlp.pipe(df_sample['summaries'], batch_size=50))

# 2. Extract Noun Phrases and Named Entities
noun_phrases = []
entities = []
entity_labels = []

for doc in docs:
    # Noun Phrases (Noun Chunks)
    # We'll filter for longer phrases or specific properties if needed
    noun_phrases.extend([chunk.text.lower().strip() for chunk in doc.noun_chunk

    # Named Entities
    for ent in doc.ents:
        entities.append(ent.text)
        entity_labels.append(ent.label_)

# 3. Rule-based Matcher for Technical Terms
matcher = Matcher(nlp.vocab)
# Patterns for common technical terms
patterns = [
    [{"LOWER": "deep"}, {"LOWER": "learning"}],
    [{"LOWER": "neural"}, {"LOWER": "network"}],
    [{"LOWER": "neural"}, {"LOWER": "networks"}],
    [{"LOWER": "artificial"}, {"LOWER": "intelligence"}],
    [{"LOWER": "machine"}, {"LOWER": "learning"}],
    [{"LOWER": "computer"}, {"LOWER": "vision"}],
    [{"POS": "ADJ", "OP": "*"}, {"POS": "NOUN"}, {"LOWER": "segmentation"}]
]
matcher.add("TECH_TERM", patterns)

tech_matches = []
for doc in docs:
    matches = matcher(doc)
    for match_id, start, end in matches:
        span = doc[start:end]
        tech_matches.append(span.text.lower())

# 4. Frequency Analysis
top_noun_phrases = Counter(noun_phrases).most_common(10)
top_entities = Counter(entities).most_common(10)
entity_type_counts = Counter(entity_labels)
top_tech_terms = Counter(tech_matches).most_common(10)

# Output Data
print("\n--- Top Noun Phrases ---")
for np, count in top_noun_phrases:
    print(f"{np}: {count}")

print("\n--- Top Named Entities ---")
for ent, count in top_entities:
    print(f"{ent}: {count}")
```

```
print("\n--- Matcher Identified Technical Terms ---")
for term, count in top_tech_terms:
    print(f"{term}: {count}")

# 5. Visualizations
# Top Noun Phrases Plot
plt.figure(figsize=(10, 6))
np_df = pd.DataFrame(top_noun_phrases, columns=['Noun Phrase', 'Count'])
sns.barplot(data=np_df, x='Count', y='Noun Phrase', palette='viridis')
plt.title('Top 10 Frequent Noun Phrases')
plt.tight_layout()
plt.savefig('top_noun_phrases.png')

# Entity Frequency Chart
plt.figure(figsize=(10, 6))
ent_df = pd.DataFrame(entity_type_counts.most_common(10), columns=['Entity Type
sns.barplot(data=ent_df, x='Count', y='Entity Type', palette='magma')
plt.title('Frequency of Entity Types')
plt.tight_layout()
plt.savefig('entity_type_frequency.png')

print("\nVisualizations saved.")
```

```
Processing text...

--- Top Noun Phrases ---
this paper: 195
image segmentation: 148
our method: 115
this work: 98
semantic segmentation: 70
medical image segmentation: 58
the performance: 52
our approach: 51
the proposed method: 44
the network: 40

--- Top Named Entities ---
two: 195
three: 70
first: 70
CNN: 64
3D: 62
one: 54
U-Net: 51
2D: 45
1: 41
2: 38

--- Matcher Identified Technical Terms ---
image segmentation: 527
deep learning: 135
medical image segmentation: 122
neural networks: 105
neural network: 67
computer vision: 51
```

```
import spacy
try:
    nlp = spacy.load('en_core_web_sm')
    print("sm loaded")
except:
    try:
        nlp = spacy.load('en_core_web_md')
        print("md loaded")
    except:
        print("No model found, using blank")
        nlp = spacy.blank('en')
  sns.barplot(data=ent_df, x='Count', y='Entity Type', palette='magma')
sm loaded
Visualizations saved.
```

```
import pandas as pd
import re
from collections import Counter

df = pd.read_csv('arxiv_data.csv')
sample_texts = df['summaries'].head(100).tolist()

# Mocking noun phrase extraction (simple words + noun)
# This is just to give the user a preview
noun_phrases = []
for text in sample_texts:
```