O

November 16, 2024

```
[ ]: import pandas as pd
     import nltk
     import matplotlib.pyplot as plt
     import seaborn as sns
     from wordcloud import WordCloud
     from sklearn.feature_extraction.text import TfidfVectorizer
     from nltk.sentiment.vader import SentimentIntensityAnalyzer
     from sklearn.decomposition import LatentDirichletAllocation
     from sklearn.feature_extraction.text import CountVectorizer

     # Fix for the formatargspec issue in Python 3.11+
     from inspect import getfullargspec
     import sklearn
     sklearn.utils.fixes.signature = getfullargspec

     # Step 1: Load Data
         df = pd.read_excel('V:/master/Englisch/presenatation/real madrid.xlsx',␣
      ↪sheet_name='Sheet1')
```

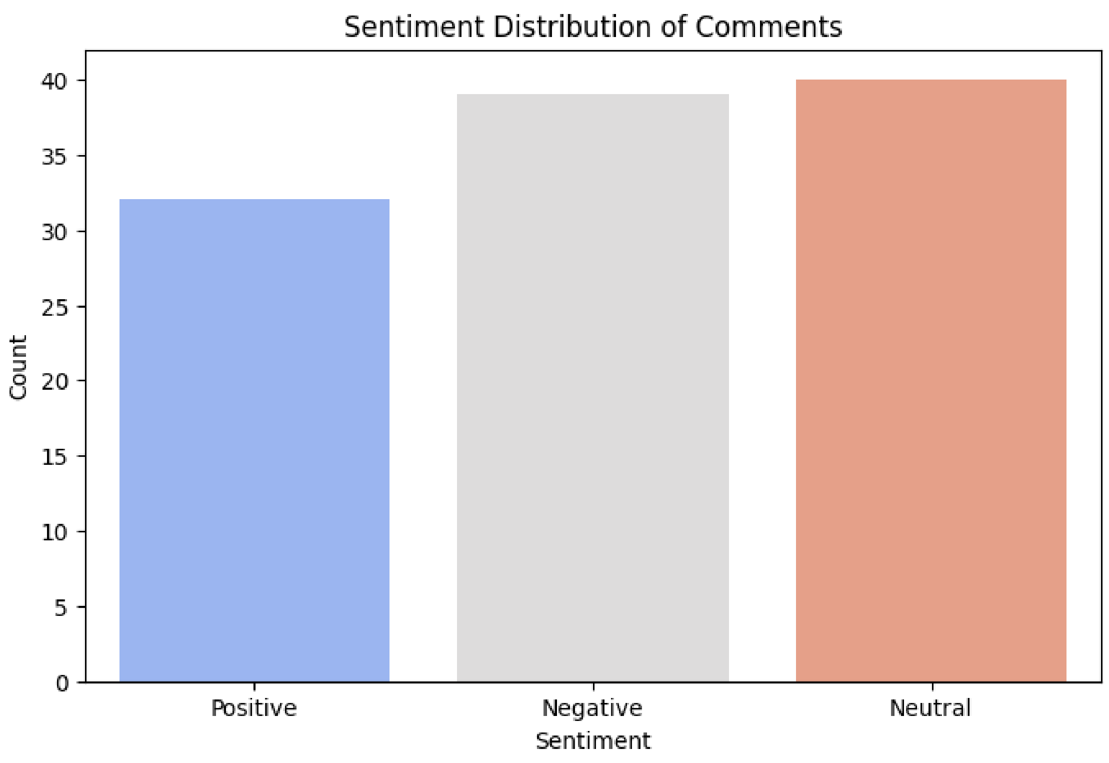
```
[ ]: import pandas as pd
     import numpy as np
     import nltk
     import re
     import matplotlib.pyplot as plt
     import seaborn as sns
     from wordcloud import WordCloud
     from nltk.sentiment import SentimentIntensityAnalyzer
     from nltk.corpus import stopwords
     from nltk.tokenize import word_tokenize
     from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
     from sklearn.decomposition import LatentDirichletAllocation
     from collections import Counter
     import plotly.express as px

     # Ensure NLTK data is available
     nltk.download('stopwords')
     nltk.download('punkt')
     nltk.download('vader_lexicon')
```

```
# Step 1: Preprocessing Function
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'\W+', ' ', text)
    text = re.sub(r'http\S+', '', text)
    text = text.strip()
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]
    return ' '.join(tokens)

# Apply preprocessing
df['Cleaned_Comment'] = df['Comment'].apply(preprocess_text)

# Step 2: Sentiment Analysis using VADER
sia = SentimentIntensityAnalyzer()
df['Sentiment'] = df['Cleaned_Comment'].apply(lambda x: sia.
 ↪polarity_scores(x)['compound'])
df['Sentiment_Label'] = df['Sentiment'].apply(lambda x: 'Positive' if x > 0␣
 ↪else ('Negative' if x < 0 else 'Neutral'))

# Visualization 1: Sentiment Distribution
plt.figure(figsize=(8, 5))
sns.countplot(x='Sentiment_Label', data=df, palette='coolwarm')
plt.title('Sentiment Distribution')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.show()

# Visualization 2: Sentiment Over Time
df['Published At'] = pd.to_datetime(df['Published At'])
df['Date'] = df['Published At'].dt.date
sentiment_over_time = df.groupby('Date')['Sentiment'].mean()
plt.figure(figsize=(12, 6))
plt.plot(sentiment_over_time.index, sentiment_over_time.values, marker='o',␣
 ↪color='blue')
plt.title('Average Sentiment Over Time')
plt.xlabel('Date')
plt.ylabel('Average Sentiment')
plt.grid(True)
plt.show()

# Visualization 3: Word Frequency Analysis
all_words = ' '.join(df['Cleaned_Comment'])
word_freq = Counter(all_words.split())
```

```
common_words = pd.DataFrame(word_freq.most_common(20), columns=['Word',␣
↪'Frequency'])
plt.figure(figsize=(10, 6))
sns.barplot(x='Frequency', y='Word', data=common_words, palette='viridis')
plt.title('Top 20 Most Frequent Words')
plt.show()

# Visualization 4: WordCloud
wordcloud = WordCloud(width=800, height=400, background_color='white').
↪generate(all_words)
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('WordCloud of Comments')
plt.show()

# Visualization 5: Bigrams Analysis
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(ngram_range=(2, 2), stop_words='english')
bigrams = vectorizer.fit_transform(df['Cleaned_Comment'])
bigram_freq = Counter(vectorizer.get_feature_names_out())
common_bigrams = pd.DataFrame(bigram_freq.most_common(10), columns=['Bigram',␣
↪'Frequency'])
plt.figure(figsize=(10, 6))
sns.barplot(x='Frequency', y='Bigram', data=common_bigrams, palette='rocket')
plt.title('Top 10 Bigrams')
plt.show()

# Step 3: Topic Modeling using LDA
count_vectorizer = CountVectorizer(max_df=0.9, min_df=2, stop_words='english')
X_count = count_vectorizer.fit_transform(df['Cleaned_Comment'])
lda = LatentDirichletAllocation(n_components=5, random_state=42)
lda.fit(X_count)

# Display Topics
print("\nLDA Topics:")
for index, topic in enumerate(lda.components_):
    print(f"\nTopic {index + 1}:")
    print([count_vectorizer.get_feature_names_out()[i] for i in topic.
↪argsort()[-10:]])

# Visualization 6: Correlation Heatmap of Topics
topics_df = pd.DataFrame(lda.transform(X_count), columns=[f'Topic {i+1}' for i␣
↪in range(lda.n_components)])
plt.figure(figsize=(10, 8))
sns.heatmap(topics_df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap of Topics')
```

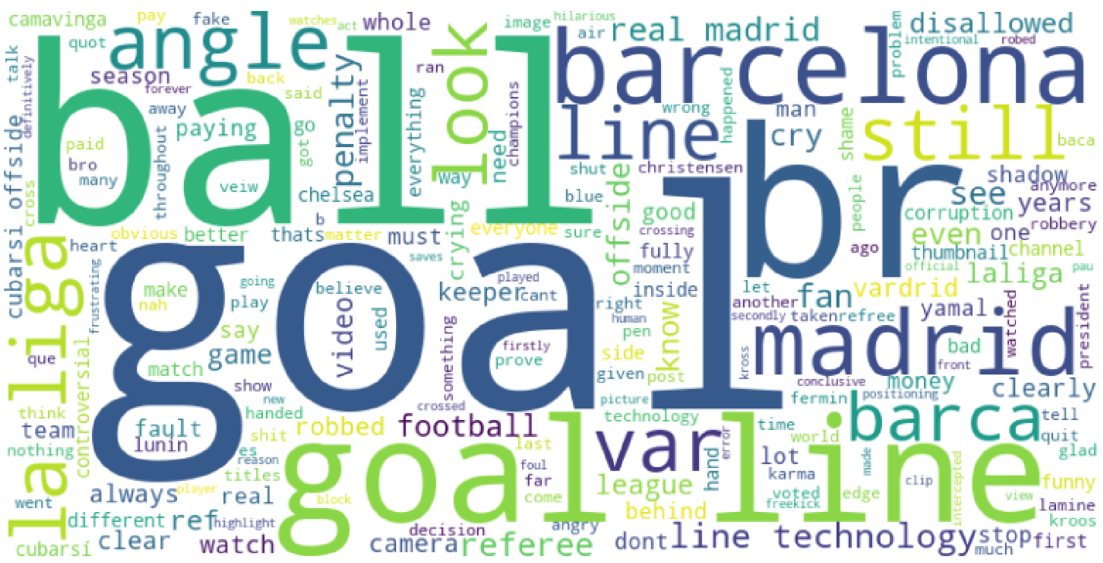
```python
plt.show()

# Visualization 7: Most Active Authors
top_authors = df['Author'].value_counts().head(10)
plt.figure(figsize=(10, 6))
sns.barplot(x=top_authors.values, y=top_authors.index, palette='Set2')
plt.title('Top 10 Most Active Authors')
plt.xlabel('Number of Comments')
plt.show()

# Save the processed dataset
df.to_csv('extended_analysis_comments.csv', index=False)
print("\nExtended Analysis Complete. Data saved as 'extended_analysis_comments.
 ↪csv'.")
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\user\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\user\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\user\AppData\Roaming\nltk_data…
[nltk_data]   Package vader_lexicon is already up-to-date!
C:\Users\user\AppData\Local\Temp\ipykernel_10368\1676910924.py:45:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x='Sentiment_Label', data=df, palette='coolwarm')
```

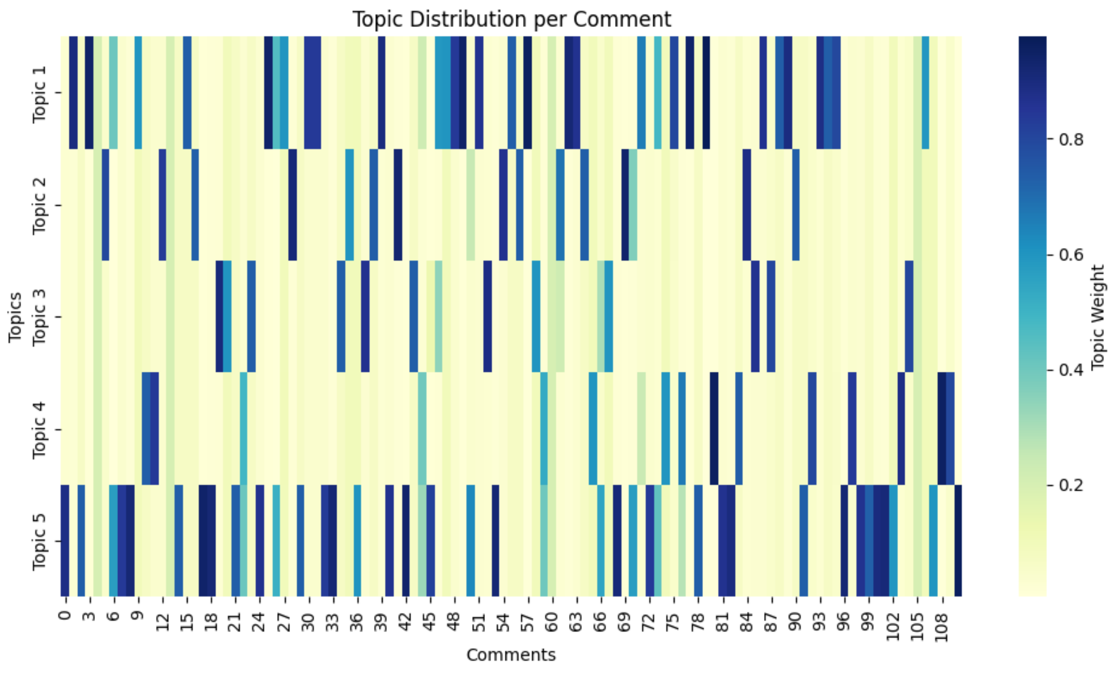


```
Top 20 Keywords by TF-IDF:
goal          17.571025
39            15.914069
line          11.454574
ball          10.192744
barcelona      8.189518
madrid         7.460681
offside        7.223973
br             6.717789
var            5.584390
still          5.424040
real           5.229244
angle          4.883931
technology     4.355280
barca          4.345673
look           4.060758
la             3.676009
liga           3.676009
penalty        3.660830
see            3.419021
also           2.968599
dtype: float64
```

WordCloud of Comments



```
LDA Topics:
Topic 1:
['game', 'ball', 'angle', 'technology', 'la', 'liga', 'real', 'madrid', 'line',
'goal']
Topic 2:
['real', 'clearly', '39', 'controversial', 'crying', 'penalty', 'br', 'madrid',
'var', 'goal']
Topic 3:
['chelsea', 'bro', 'team', 'paying', 'football', 'br', 'match', 'referee', '39',
'barcelona']
Topic 4:
['better', 'yamal', 'corruption', 'barca', 'good', 'different', 'years', 'goal',
'br', 'robbed']
Topic 5:
['league', 'video', 'br', 'disallowed', 'look', 'offside', 'ball', 'line',
'goal', '39']
```



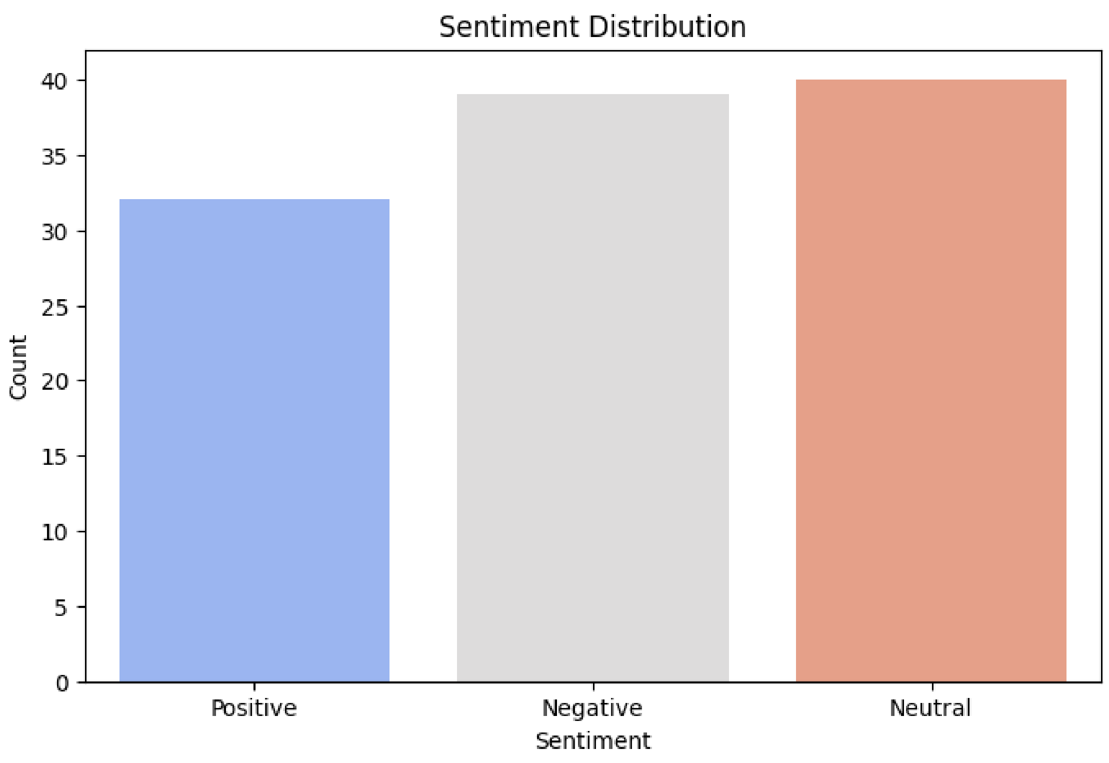
```
Analysis Complete. Processed data saved as 'processed_comments_analysis.csv'.
```

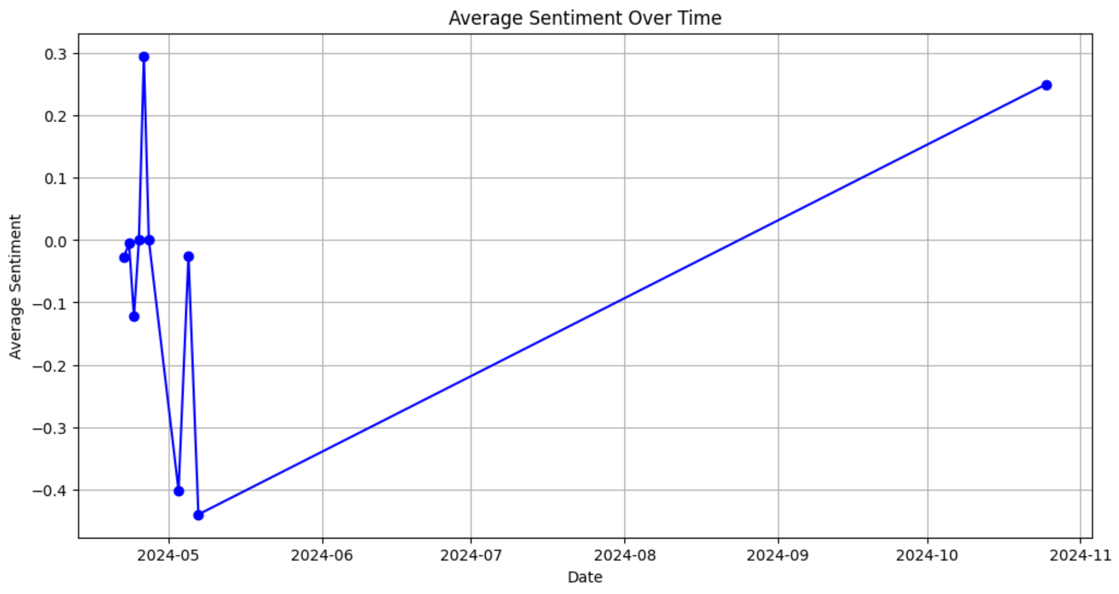
[40]:
```python
import pandas as pd
import numpy as np
import nltk
import re
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
from nltk.sentiment import SentimentIntensityAnalyzer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
from collections import Counter
import plotly.express as px

# Ensure NLTK data is available
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('vader_lexicon')

# Step 1: Preprocessing Function
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'\W+', ' ', text)
    text = re.sub(r'http\S+', '', text)
    text = text.strip()
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]
    return ' '.join(tokens)

# Apply preprocessing
df['Cleaned_Comment'] = df['Comment'].apply(preprocess_text)

# Step 2: Sentiment Analysis using VADER
sia = SentimentIntensityAnalyzer()
df['Sentiment'] = df['Cleaned_Comment'].apply(lambda x: sia.
  ↪polarity_scores(x)['compound'])
df['Sentiment_Label'] = df['Sentiment'].apply(lambda x: 'Positive' if x > 0␣
  ↪else ('Negative' if x < 0 else 'Neutral'))

# Visualization 1: Sentiment Distribution
plt.figure(figsize=(8, 5))
sns.countplot(x='Sentiment_Label', data=df, palette='coolwarm')
plt.title('Sentiment Distribution')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.show()

# Visualization 2: Sentiment Over Time
df['Published At'] = pd.to_datetime(df['Published At'])
df['Date'] = df['Published At'].dt.date
sentiment_over_time = df.groupby('Date')['Sentiment'].mean()
plt.figure(figsize=(12, 6))
plt.plot(sentiment_over_time.index, sentiment_over_time.values, marker='o',␣
  ↪color='blue')
plt.title('Average Sentiment Over Time')
plt.xlabel('Date')
plt.ylabel('Average Sentiment')
plt.grid(True)
plt.show()

# Visualization 3: Word Frequency Analysis
all_words = ' '.join(df['Cleaned_Comment'])
word_freq = Counter(all_words.split())
common_words = pd.DataFrame(word_freq.most_common(20), columns=['Word',␣
  ↪'Frequency'])
plt.figure(figsize=(10, 6))
sns.barplot(x='Frequency', y='Word', data=common_words, palette='viridis')
```

```
plt.title('Top 20 Most Frequent Words')
plt.show()

# Visualization 4: WordCloud
wordcloud = WordCloud(width=800, height=400, background_color='white').
  ↪generate(all_words)
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('WordCloud of Comments')
plt.show()

# Visualization 5: Bigrams Analysis
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(ngram_range=(2, 2), stop_words='english')
bigrams = vectorizer.fit_transform(df['Cleaned_Comment'])
bigram_freq = Counter(vectorizer.get_feature_names_out())
common_bigrams = pd.DataFrame(bigram_freq.most_common(10), columns=['Bigram',␣
  ↪'Frequency'])
plt.figure(figsize=(10, 6))
sns.barplot(x='Frequency', y='Bigram', data=common_bigrams, palette='rocket')
plt.title('Top 10 Bigrams')
plt.show()

# Step 3: Topic Modeling using LDA
count_vectorizer = CountVectorizer(max_df=0.9, min_df=2, stop_words='english')
X_count = count_vectorizer.fit_transform(df['Cleaned_Comment'])
lda = LatentDirichletAllocation(n_components=5, random_state=42)
lda.fit(X_count)

# Display Topics
print("\nLDA Topics:")
for index, topic in enumerate(lda.components_):
    print(f"\nTopic {index + 1}:")
    print([count_vectorizer.get_feature_names_out()[i] for i in topic.
  ↪argsort()[-10:]])

# Visualization 6: Correlation Heatmap of Topics
topics_df = pd.DataFrame(lda.transform(X_count), columns=[f'Topic {i+1}' for i␣
  ↪in range(lda.n_components)])
plt.figure(figsize=(10, 8))
sns.heatmap(topics_df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap of Topics')
plt.show()

# Visualization 7: Most Active Authors
top_authors = df['Author'].value_counts().head(10)
```

```
plt.figure(figsize=(10, 6))
sns.barplot(x=top_authors.values, y=top_authors.index, palette='Set2')
plt.title('Top 10 Most Active Authors')
plt.xlabel('Number of Comments')
plt.show()

# Save the processed dataset
df.to_csv('extended_analysis_comments.csv', index=False)
print("\nExtended Analysis Complete. Data saved as 'extended_analysis_comments.
 ↪csv'.")
```

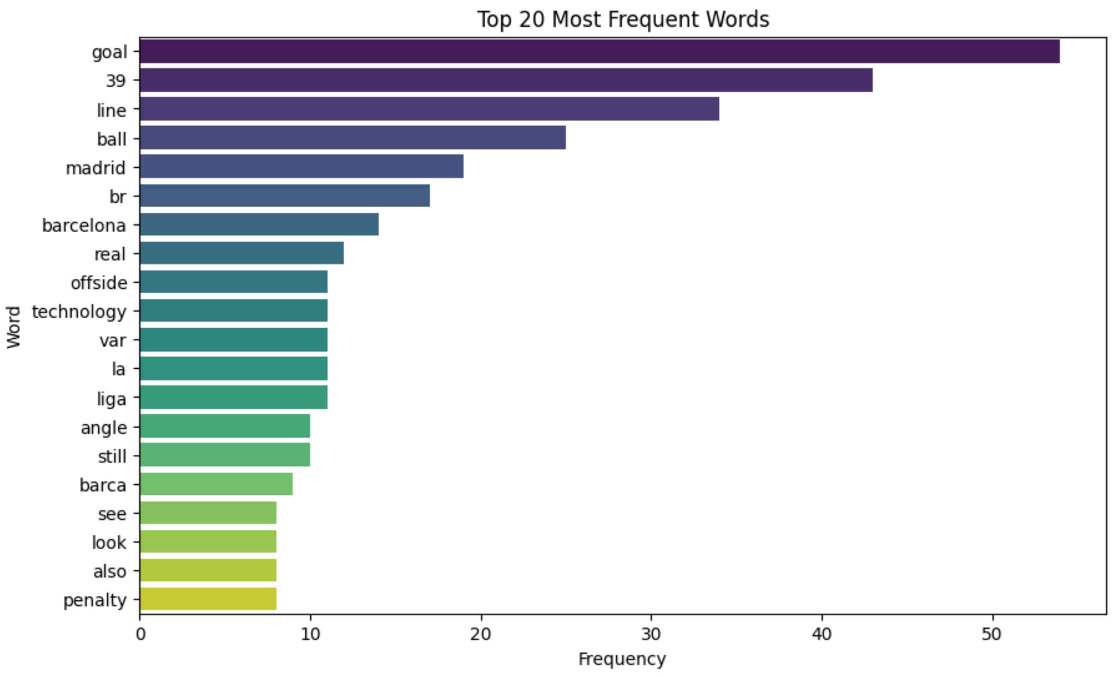
```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\user\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\user\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\user\AppData\Roaming\nltk_data…
[nltk_data]   Package vader_lexicon is already up-to-date!
C:\Users\user\AppData\Local\Temp\ipykernel_10368\2458968829.py:42:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x='Sentiment_Label', data=df, palette='coolwarm')
```

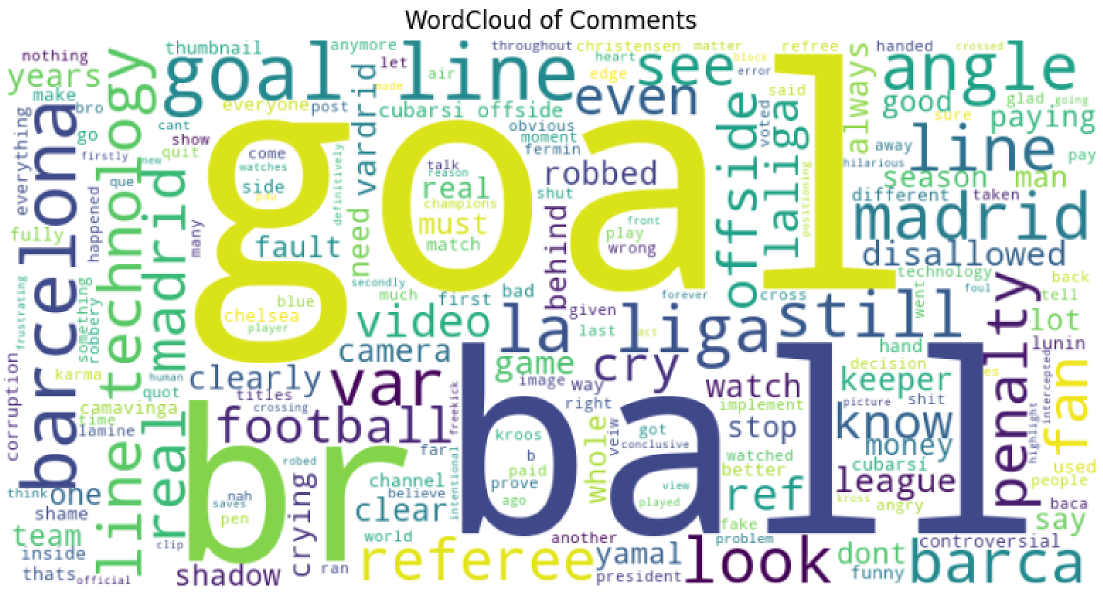




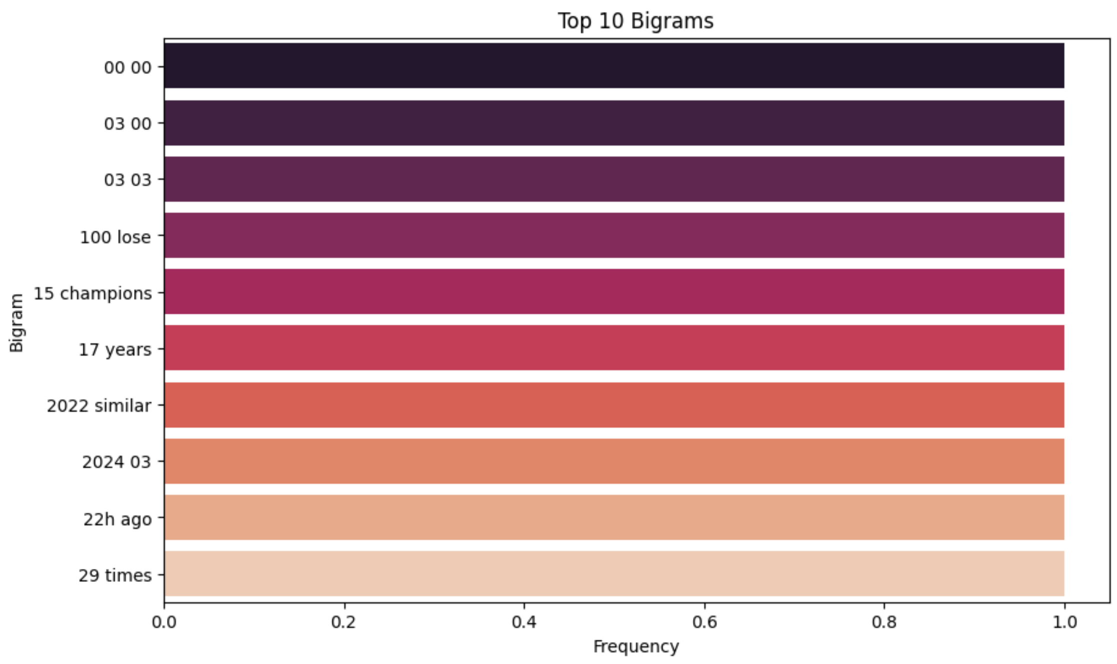
```
C:\Users\user\AppData\Local\Temp\ipykernel_10368\2458968829.py:65:
FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='Frequency', y='Word', data=common_words, palette='viridis')
```

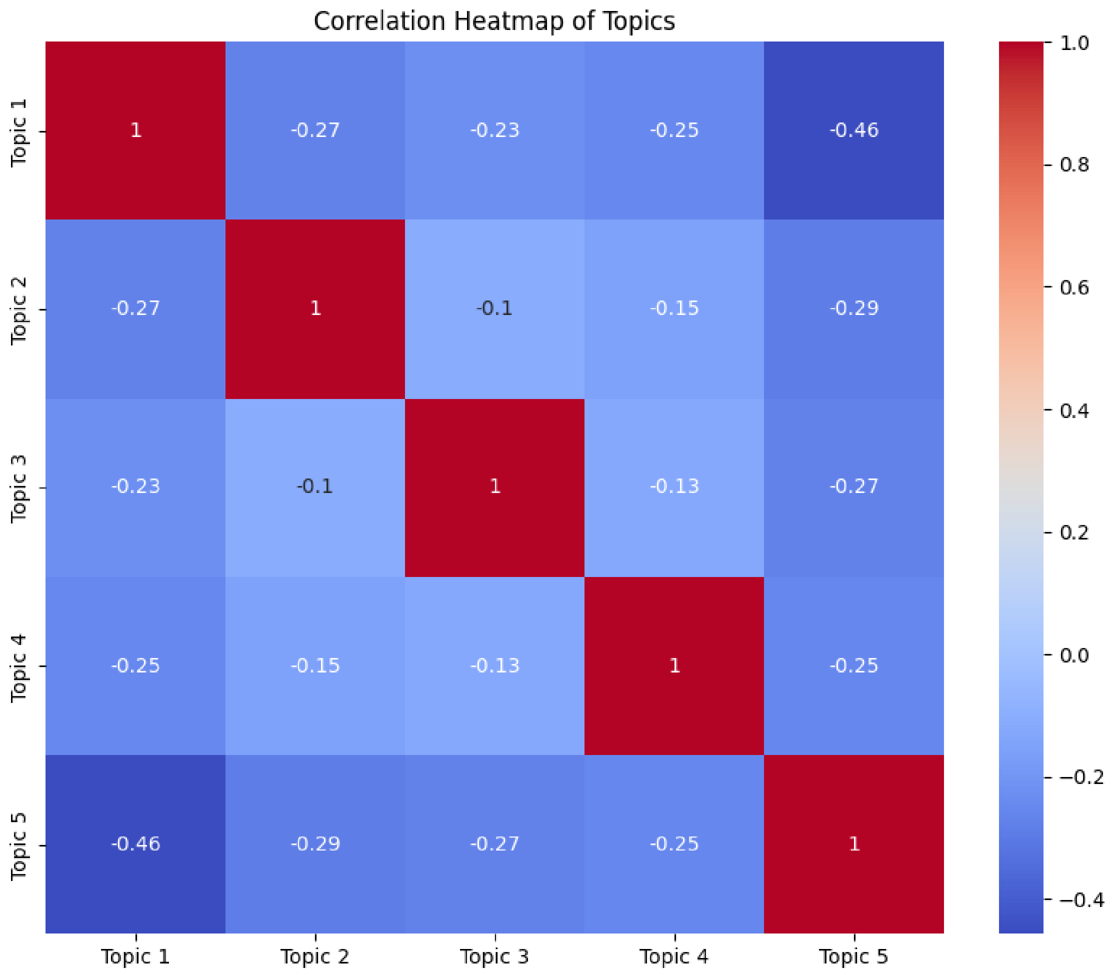




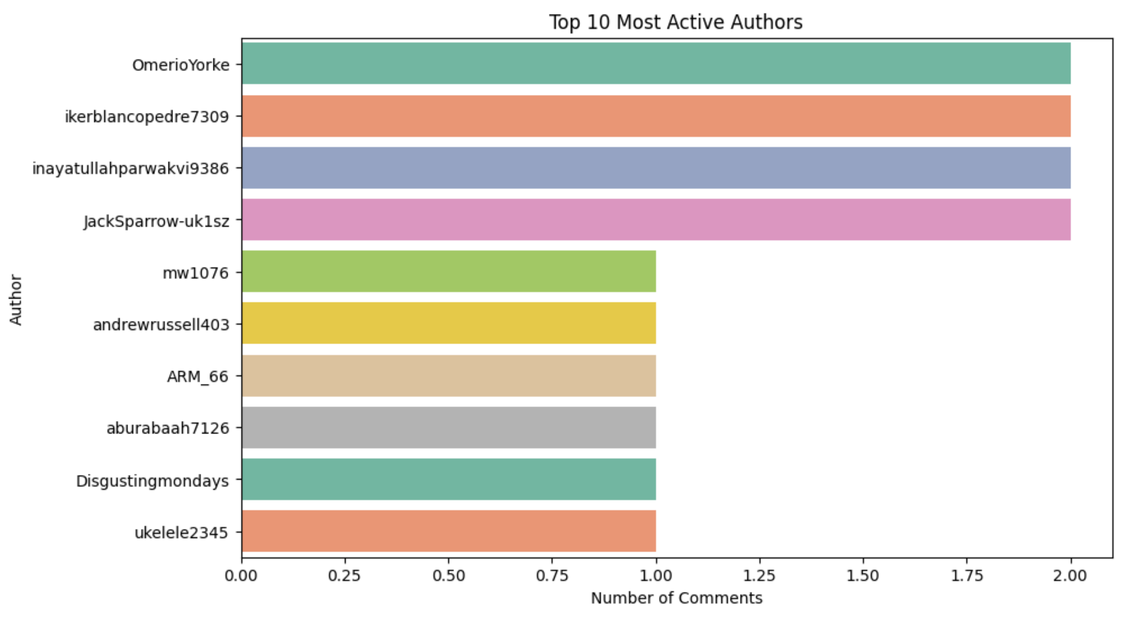
```
C:\Users\user\AppData\Local\Temp\ipykernel_10368\2458968829.py:84:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='Frequency', y='Bigram', data=common_bigrams, palette='rocket')
```



```
LDA Topics:

Topic 1:
['game', 'ball', 'angle', 'technology', 'la', 'liga', 'real', 'madrid', 'line',
'goal']

Topic 2:
['real', 'clearly', '39', 'controversial', 'crying', 'penalty', 'br', 'madrid',
'var', 'goal']

Topic 3:
['chelsea', 'bro', 'team', 'paying', 'football', 'br', 'match', 'referee', '39',
'barcelona']

Topic 4:
['better', 'yamal', 'corruption', 'barca', 'good', 'different', 'years', 'goal',
'br', 'robbed']

Topic 5:
['league', 'video', 'br', 'disallowed', 'look', 'offside', 'ball', 'line',
'goal', '39']
```



```
C:\Users\user\AppData\Local\Temp\ipykernel_10368\2458968829.py:110:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=top_authors.values, y=top_authors.index, palette='Set2')
```



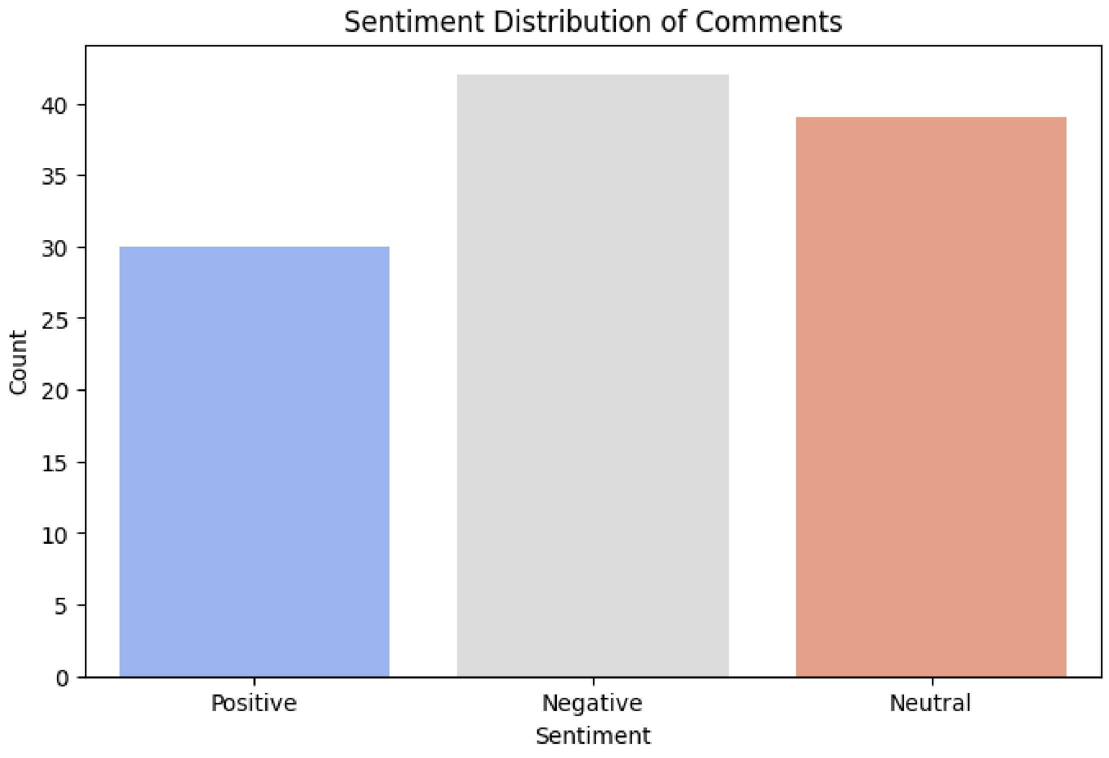
```
Extended Analysis Complete. Data saved as 'extended_analysis_comments.csv'.
```

[44]:
```
import spacy
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from spacy import displacy
from nltk.sentiment import SentimentIntensityAnalyzer
from wordcloud import WordCloud

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

# Load dataset from Excel
df = pd.read_excel('V:/master/Englisch/presenatation/real madrid.xlsx',␣
 ↪sheet_name='Sheet1')

# Check if 'Comment' column exists and preprocess the data
if 'Comment' not in df.columns:
    print("Error: 'Comment' column not found in the dataset.")
else:
    # Ensure all comments are strings
    df['Comment'] = df['Comment'].apply(lambda x: str(x) if not isinstance(x,␣
 ↪str) else x)

    # Preprocess and analyze text data with spaCy
```

```
def preprocess_with_spacy(text):
    # Process the text through spaCy's NLP pipeline
    doc = nlp(text)

    # Extract named entities
    entities = [(ent.text, ent.label_) for ent in doc.ents]

    # Return the processed text and named entities
    return doc, entities

# Apply spaCy NLP pipeline to each comment
df['Doc'], df['Named_Entities'] = zip(*df['Comment'].
↪apply(preprocess_with_spacy))

# Sentiment Analysis using VADER
sia = SentimentIntensityAnalyzer()
df['Sentiment'] = df['Comment'].apply(lambda x: sia.
↪polarity_scores(x)['compound'])
df['Sentiment_Label'] = df['Sentiment'].apply(lambda x: 'Positive' if x > 0␣
↪else ('Negative' if x < 0 else 'Neutral'))

# Visualization 1: Sentiment Distribution
plt.figure(figsize=(8, 5))
sns.countplot(x='Sentiment_Label', data=df, palette='coolwarm')
plt.title('Sentiment Distribution of Comments')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.show()

# Visualization 2: Named Entities Visualization with spaCy
def plot_named_entities(doc):
    # Visualize entities in the text
    displacy.render(doc, style='ent', jupyter=True)

# Visualize named entities in the first comment
plot_named_entities(df['Doc'][0])

# Visualizing Entity Types Distribution
# Count the frequency of entity types
entity_labels = {}
for entities in df['Named_Entities']:
    for _, label in entities:
        if label in entity_labels:
            entity_labels[label] += 1
        else:
            entity_labels[label] = 1
```

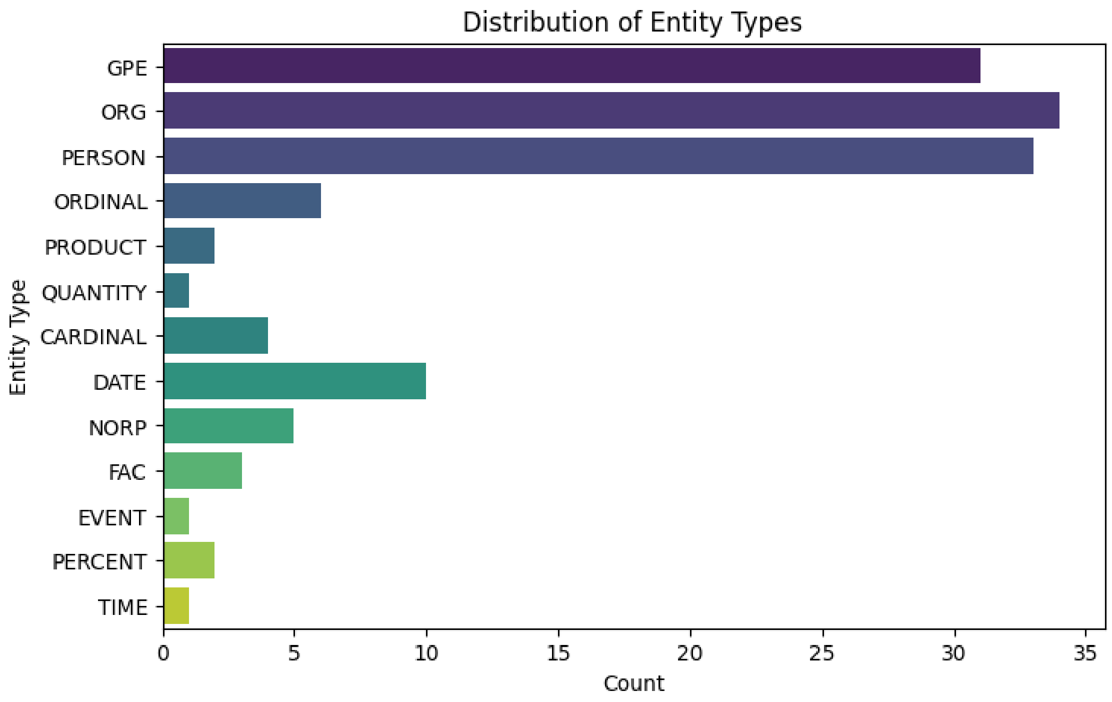
```
  # Visualization 3: Entity Types Distribution using barplot
  plt.figure(figsize=(8, 5))
  sns.barplot(x=list(entity_labels.values()), y=list(entity_labels.keys()),␣
↪palette='viridis')
  plt.title('Distribution of Entity Types')
  plt.xlabel('Count')
  plt.ylabel('Entity Type')
  plt.show()

  # Word Cloud Generation
  text_for_wordcloud = ' '.join(df['Comment'])
  wordcloud = WordCloud(background_color='white', width=800, height=400).
↪generate(text_for_wordcloud)
  plt.figure(figsize=(10, 6))
  plt.imshow(wordcloud, interpolation='bilinear')
  plt.axis('off')
  plt.title('WordCloud of Comments')
  plt.show()

  # Save the processed dataset for further analysis
  df.to_csv('processed_comments_analysis.csv', index=False)
  print("Analysis Complete. Processed data saved as␣
↪'processed_comments_analysis.csv'.")
```

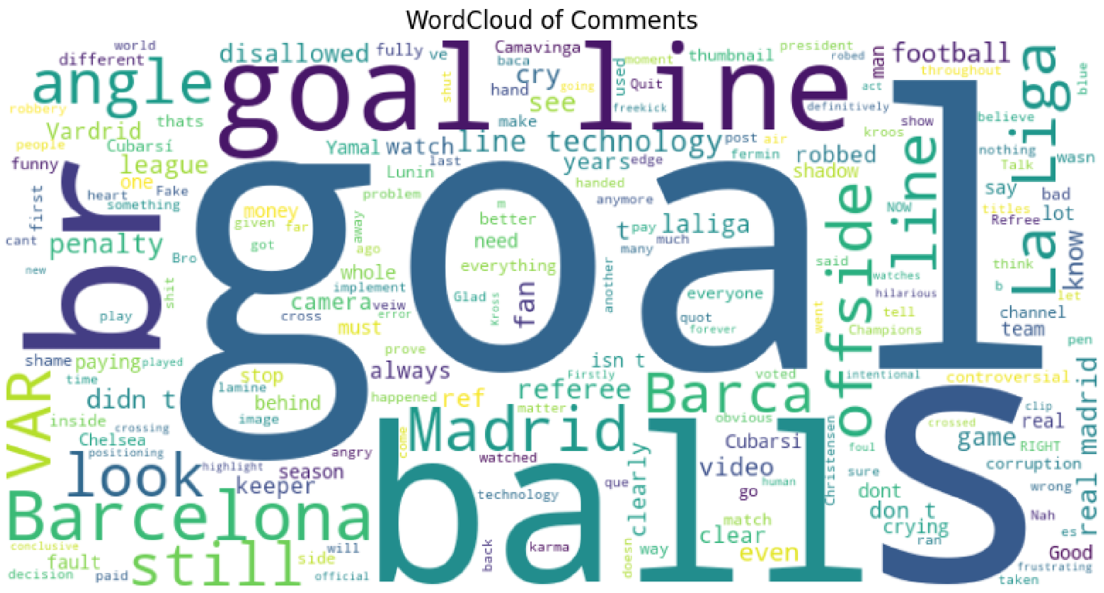
```
C:\Users\user\AppData\Local\Temp\ipykernel_10368\3253552271.py:43:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x='Sentiment_Label', data=df, palette='coolwarm')
```



```
<IPython.core.display.HTML object>
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_10368\3253552271.py:69:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=list(entity_labels.values()), y=list(entity_labels.keys()),
palette='viridis')
```





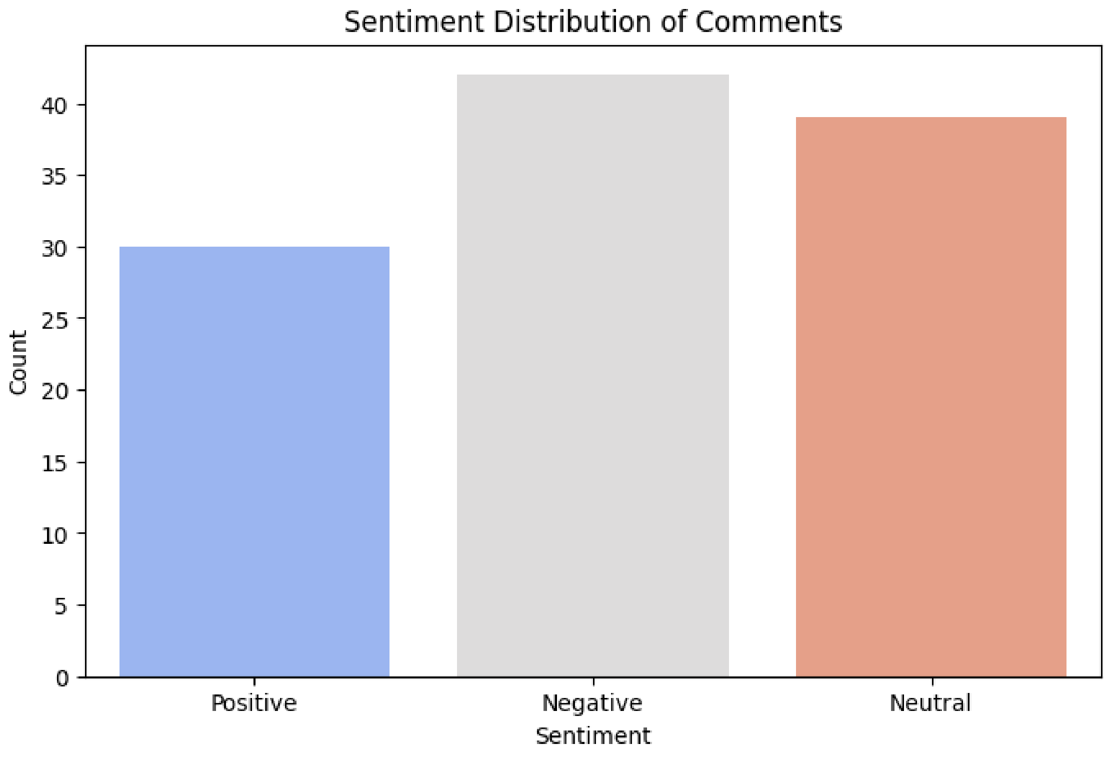
```
Analysis Complete. Processed data saved as 'processed_comments_analysis.csv'.
```

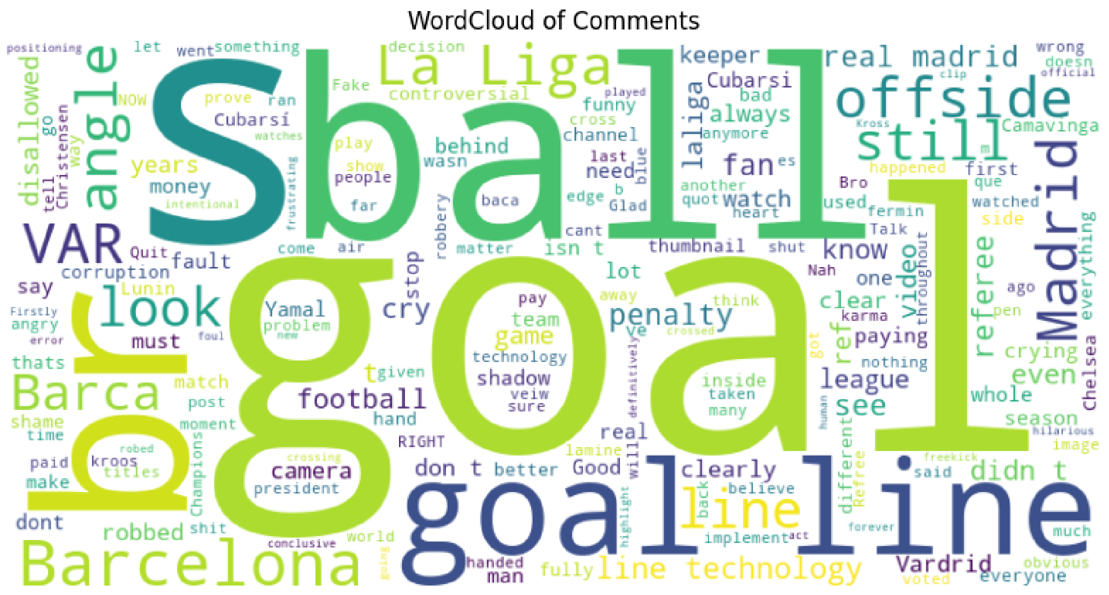
[45]:
```python
import spacy
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from spacy import displacy
from nltk.sentiment import SentimentIntensityAnalyzer
from wordcloud import WordCloud
from sklearn.decomposition import LatentDirichletAllocation as LDA
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.preprocessing import LabelEncoder
from nltk.util import ngrams
import numpy as np
import re
from sklearn.metrics import silhouette_score

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

# Load dataset
df = pd.read_excel('V:/master/Englisch/presenatation/real madrid.xlsx',␣
↪sheet_name='Sheet1')

# Ensure all comments are strings
df['Comment'] = df['Comment'].apply(lambda x: str(x) if not isinstance(x, str)␣
↪else x)

# Sentiment Analysis using VADER
sia = SentimentIntensityAnalyzer()
df['Sentiment'] = df['Comment'].apply(lambda x: sia.
↪polarity_scores(x)['compound'])
df['Sentiment_Label'] = df['Sentiment'].apply(lambda x: 'Positive' if x > 0␣
↪else ('Negative' if x < 0 else 'Neutral'))

# Visualize Sentiment Distribution
plt.figure(figsize=(8, 5))
sns.countplot(x='Sentiment_Label', data=df, palette='coolwarm')
plt.title('Sentiment Distribution of Comments')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.show()

# Preprocess and Extract Named Entities using spaCy
def preprocess_with_spacy(text):
    doc = nlp(text)
    entities = [(ent.text, ent.label_) for ent in doc.ents]
    return doc, entities

df['Doc'], df['Named_Entities'] = zip(*df['Comment'].
↪apply(preprocess_with_spacy))
```

```
# Word Cloud Generation
text_for_wordcloud = ' '.join(df['Comment'])
wordcloud = WordCloud(background_color='white', width=800, height=400).
  ↪generate(text_for_wordcloud)
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('WordCloud of Comments')
plt.show()

# **Advanced Topic Modeling with LDA**
# Create a TF-IDF Vectorizer and apply LDA
tfidf = TfidfVectorizer(stop_words='english', max_features=1000)
X_tfidf = tfidf.fit_transform(df['Comment'])

lda = LDA(n_components=5, random_state=42)
lda_topics = lda.fit_transform(X_tfidf)

# Display LDA topics
def get_lda_topics(model, feature_names, n_words=10):
    topics = []
    for topic_idx, topic in enumerate(model.components_):
        topic_words = [feature_names[i] for i in topic.argsort()[:-n_words - 1:
  ↪-1]]
        topics.append(topic_words)
    return topics

lda_topics_words = get_lda_topics(lda, tfidf.get_feature_names_out())
for i, topic in enumerate(lda_topics_words):
    print(f"Topic {i+1}: {', '.join(topic)}")

# Visualize LDA Topic Distribution
topic_distribution = pd.DataFrame(lda_topics, columns=[f'Topic {i+1}' for i in␣
  ↪range(5)])
topic_distribution['Dominant_Topic'] = topic_distribution.idxmax(axis=1)
sns.countplot(x='Dominant_Topic', data=topic_distribution, palette='coolwarm')
plt.title('Distribution of Dominant Topics')
plt.xlabel('Topic')
plt.ylabel('Count')
plt.show()

# **Advanced N-Gram Analysis (Bigrams and Trigrams)**
def get_ngrams(text, n=2):
    words = re.findall(r'\w+', text.lower())
    return list(ngrams(words, n))
```

```python
bigrams = df['Comment'].apply(lambda x: get_ngrams(x, n=2))
trigrams = df['Comment'].apply(lambda x: get_ngrams(x, n=3))

# Flatten bigrams and trigrams
flat_bigrams = [item for sublist in bigrams for item in sublist]
flat_trigrams = [item for sublist in trigrams for item in sublist]

# Display most frequent bigrams and trigrams
from collections import Counter
bigram_freq = Counter(flat_bigrams)
trigram_freq = Counter(flat_trigrams)

print("Top 10 Bigrams:", bigram_freq.most_common(10))
print("Top 10 Trigrams:", trigram_freq.most_common(10))

# **Clustering Comments using K-Means**
kmeans = KMeans(n_clusters=5, random_state=42)
df['Cluster'] = kmeans.fit_predict(X_tfidf)

# Visualize clustering results
plt.figure(figsize=(8, 5))
sns.countplot(x='Cluster', data=df, palette='Set2')
plt.title('Clustering of Comments')
plt.xlabel('Cluster')
plt.ylabel('Count')
plt.show()

# **Silhouette Score for Clustering Quality**
sil_score = silhouette_score(X_tfidf, df['Cluster'])
print(f"Silhouette Score: {sil_score:.3f}")

# Save the processed dataset
df.to_csv('extended_analysis_comments.csv', index=False)
print("Extended Analysis Complete. Data saved as 'extended_analysis_comments.
 ↪csv'.")
```

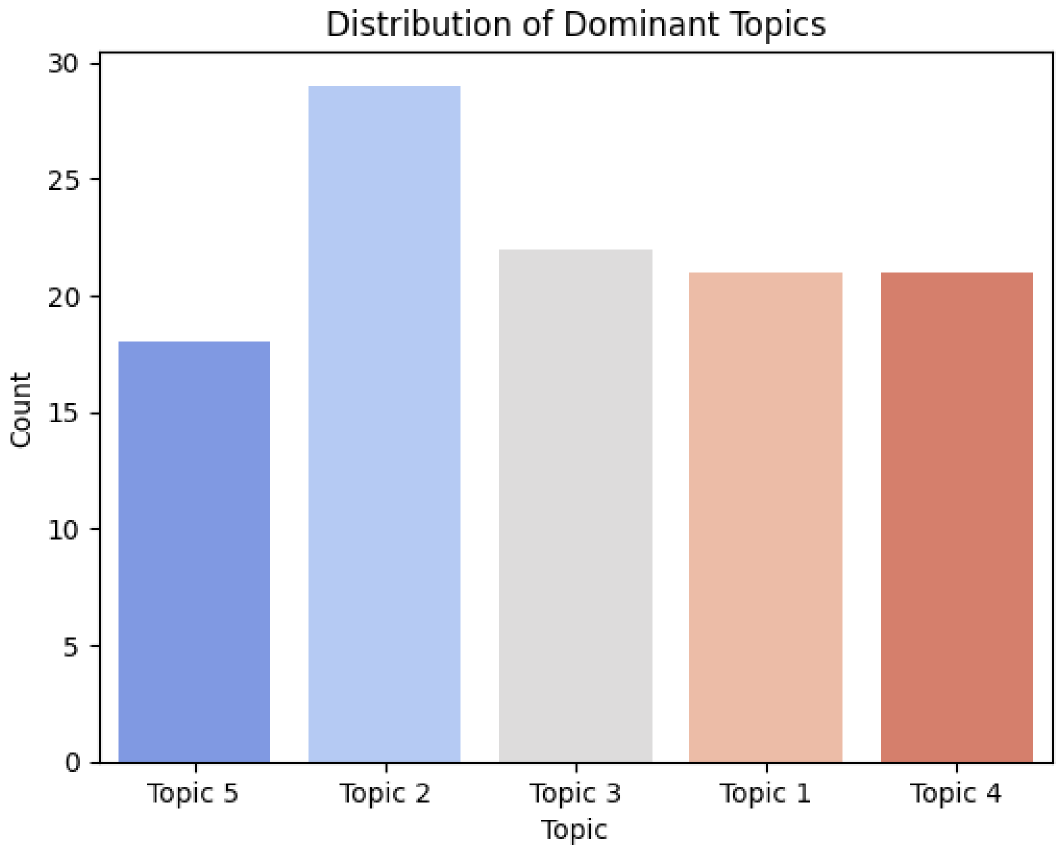
```
C:\Users\user\AppData\Local\Temp\ipykernel_10368\2614709392.py:33:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x='Sentiment_Label', data=df, palette='coolwarm')
```





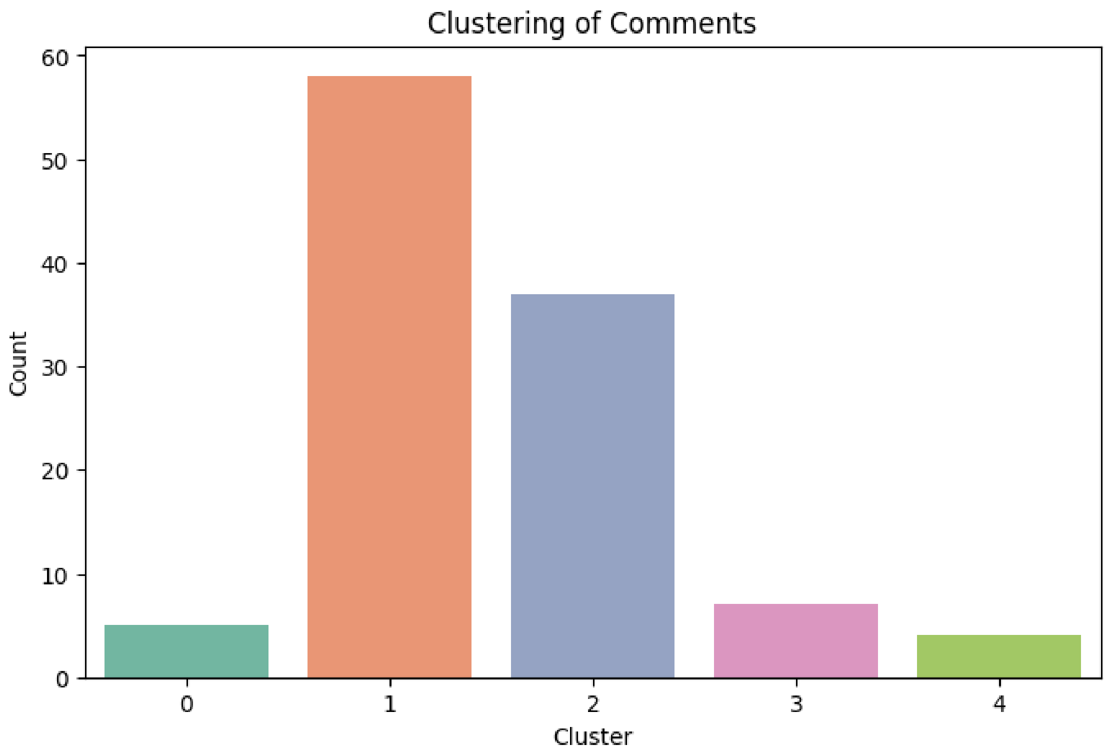
```
Topic 1: goal, offside, 39, baby, vardridd, cubarsí, technology, liga, la, line
Topic 2: goal, 39, barcelona, line, var, madrid, dont, ball, shadow, real
Topic 3: offside, cubarsi, match, 39, football, madrid, br, man, quit, chelsea
Topic 4: goal, 39, line, don, br, technology, laliga, vardrid, crying, barcelona
Topic 5: barca, ball, inside, line, robbed, camera, better, convincing, vardrid,
won
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_10368\2614709392.py:79:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x='Dominant_Topic', data=topic_distribution, palette='coolwarm')
```



```
Top 10 Bigrams: [(('39', 's'), 22), (('the', 'ball'), 19), (('a', 'goal'), 18),
(('the', 'line'), 17), (('goal', 'line'), 15), (('it', 'was'), 12), (('of',
'the'), 11), (('the', 'goal'), 11), (('la', 'liga'), 11), (('39', 't'), 10)]
Top 10 Trigrams: [(('goal', 'line', 'technology'), 8), (('it', '39', 's'), 8),
(('the', 'goal', 'line'), 7), (('was', 'a', 'goal'), 7), (('it', 'was', 'a'),
6), (('didn', '39', 't'), 5), (('not', 'a', 'goal'), 5), (('look', 'at', 'the'),
4), (('of', 'the', 'goal'), 4), (('of', 'the', 'ball'), 4)]
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_10368\2614709392.py:111:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x='Cluster', data=df, palette='Set2')
```



```
Silhouette Score: 0.021
Extended Analysis Complete. Data saved as 'extended_analysis_comments.csv'.
```

[51]:
```python
from nltk.sentiment import SentimentIntensityAnalyzer

# Initialize the SentimentIntensityAnalyzer
sia = SentimentIntensityAnalyzer()

# Function to get sentiment label based on compound score
def get_sentiment_label(compound_score):
    if compound_score >= 0.05:
        return 'Positive'
    elif compound_score <= -0.05:
        return 'Negative'
    else:
        return 'Neutral'

# Apply sentiment analysis on the comments and store the result in a new column
df['Sentiment_Score'] = df['Comment'].apply(lambda x: sia.
 ↪polarity_scores(x)['compound'])
df['Sentiment'] = df['Sentiment_Score'].apply(get_sentiment_label)

# Check the first few rows to ensure the new columns are added
print(df.head())
```

```
                Author                                             Comment  \
0     toniilievski3934  Even on the official video you can clearly see…
1      DrJaswin_Dsouza  real madrid would have paid a lot of money to …
2  SohaybMirouad-rx8sv                                  Look at the shadow
3     andrewrussell403  The thumbnail is hilarious. It is being used t…
4               mw1076  It was not a human error. However, it was an i…

           Published At  Sentiment_Score Sentiment
0  2024-10-25T17:08:17Z           0.2500  Positive
1  2024-05-07T16:28:02Z          -0.5777  Negative
2  2024-05-05T17:27:08Z           0.0000   Neutral
3  2024-05-05T01:35:33Z          -0.3400  Negative
4  2024-05-03T14:03:20Z           0.3089  Positive
```

[52]:
```
df['Sentiment_Num'] = df['Sentiment'].map({'Positive': 1, 'Neutral': 0,␣
 ↪'Negative': -1})
```