Step 1: Installing Required Libraries

```
!pip install pandas matplotlib seaborn nltk wordcloud
import nltk
from wordcloud import WordCloud
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('punkt_tab')
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-packages (0.13.2)
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packages (3.9.1)
Requirement already satisfied: wordcloud in /usr/local/lib/python3.11/dist-packages (1.9.4)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.1)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages (from nltk) (8.1.8)
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from nltk) (4.67.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas)
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
True
```

Step 2: Load the JSON into a DataFrame

```
import pandas as pd

# Load JSON file
file_path = '/content/sarcasm_data.json'
with open(file_path, 'r') as f:
    data = pd.read_json(f, orient='index')

# Reset index to make IDs a column
data = data.reset_index().rename(columns={'index': 'id'})
print(data.head())  # Check first 5 rows
```

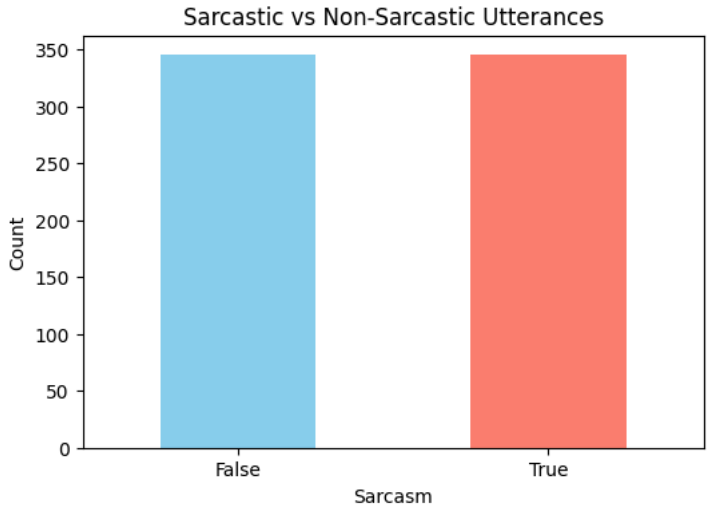
```
     id                                          utterance  speaker  \
0   160  It's just a privilege to watch your mind at work.  SHELDON
1   170  I don't think I'll be able to stop thinking ab...    PENNY
2   180  Since it's not bee season, you can have my epi...  SHELDON
3   190  Lois Lane is falling, accelerating at an initi...  SHELDON
4  1105  I'm just inferring this is a couch because the...  SHELDON

                                        context  \
0  [I never would have identified the fingerprint...
1  [This is one of my favorite places to kick bac...
2  [Here we go. Pad thai, no peanuts., But does i...
3  [A marathon? How many Superman movies are ther...
4  [Great Caesar's ghost, look at this place., So...

                              context_speakers show  sarcasm
0                           [LEONARD, SHELDON]  BBT     True
1  [HOWARD, PENNY, HOWARD, HOWARD, HOWARD, PENNY,...  BBT     True
2                       [LEONARD, HOWARD, LEONARD]  BBT    False
3  [PENNY, SHELDON, PENNY, SHELDON, SHELDON, PENN...  BBT    False
4  [SHELDON, LEONARD, SHELDON, SHELDON, SHELDON, ...  BBT     True
```

Step 3: Class Distribution Analysis

```
import matplotlib.pyplot as plt

# Count sarcastic vs non-sarcastic
class_counts = data['sarcasm'].value_counts()
```

```
print("Class Distribution:\n", class_counts)

# Plot
plt.figure(figsize=(6, 4))
class_counts.plot(kind='bar', color=['skyblue', 'salmon'])
plt.title('Sarcastic vs Non-Sarcastic Utterances')
plt.xlabel('Sarcasm')
plt.ylabel('Count')
plt.xticks(ticks=[0, 1], labels=['False', 'True'], rotation=0)
plt.show()

# Percentage
percent_sarcasm = (class_counts[True] / len(data)) * 100
print(f"Percentage Sarcastic: {percent_sarcasm:.2f}%")
```

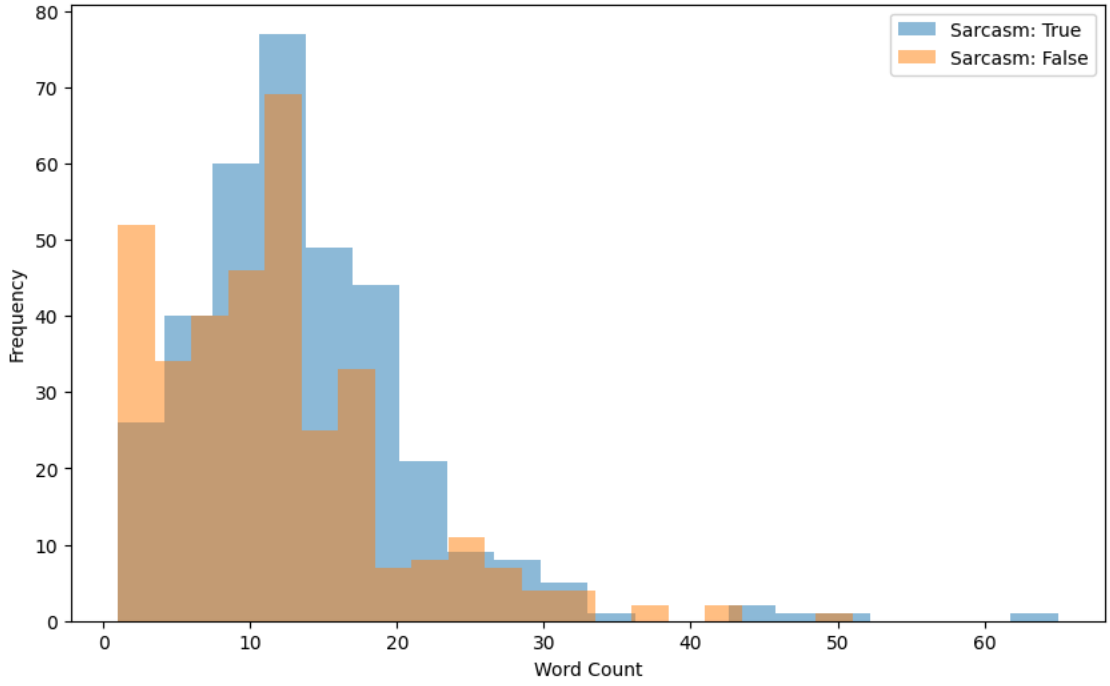
```
Class Distribution:
 sarcasm
True     345
False    345
Name: count, dtype: int64
```




```
Percentage Sarcastic: 50.00%
```

Step 4: Text Length Analysis

```
# Add word count and character count columns
data['word_count'] = data['utterance'].apply(lambda x: len(str(x).split()))
data['char_count'] = data['utterance'].apply(len)

# Plot word count distribution
plt.figure(figsize=(10, 6))
for label in [True, False]:
    subset = data[data['sarcasm'] == label]
    plt.hist(subset['word_count'], bins=20, alpha=0.5, label=f'Sarcasm: {label}')
plt.title('Word Count Distribution by Sarcasm')
plt.xlabel('Word Count')
plt.ylabel('Frequency')
plt.legend()
plt.show()

# Summary stats
print("Word Count Stats:\n", data.groupby('sarcasm')['word_count'].describe())
```



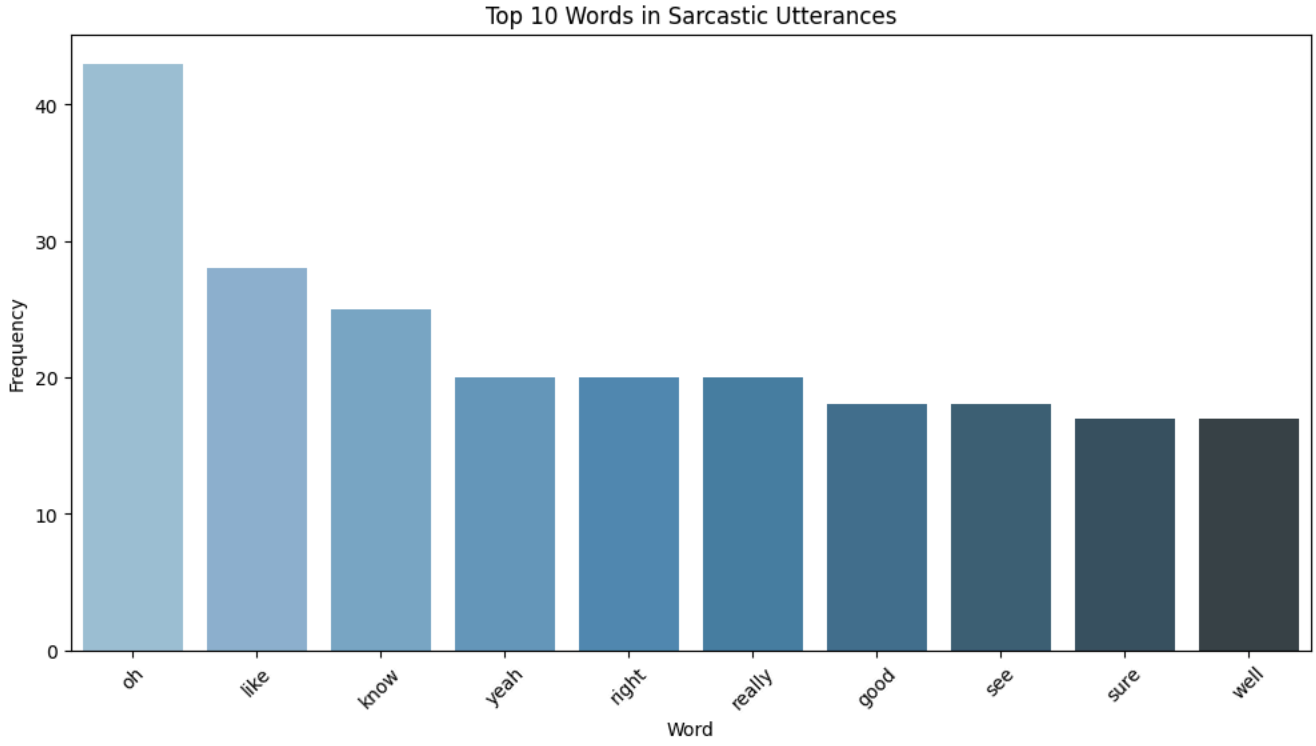

```
Word Count Stats:
          count       mean       std  min  25%   50%   75%   max
sarcasm
False     345.0  11.388406  7.863286  1.0  6.0  11.0  15.0  51.0
True      345.0  13.498551  7.896542  1.0  9.0  12.0  17.0  65.0
```

Step 5: Word and Phrase Patterns

```
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from collections import Counter
import seaborn as sns

# Tokenize and clean text
stop_words = set(stopwords.words('english'))
def clean_text(text):
    tokens = word_tokenize(str(text).lower())
    return [word for word in tokens if word.isalnum() and word not in stop_words]

# Get word frequencies
sarcastic_words = Counter()
non_sarcastic_words = Counter()
for idx, row in data.iterrows():
    words = clean_text(row['utterance'])
    if row['sarcasm']:
        sarcastic_words.update(words)
    else:
        non_sarcastic_words.update(words)

# Top 10 words
sarc_top = sarcastic_words.most_common(10)
non_sarc_top = non_sarcastic_words.most_common(10)
print("Top Sarcastic Words:", sarc_top)
print("Top Non-Sarcastic Words:", non_sarc_top)

# Plot
plt.figure(figsize=(12, 6))
sns.barplot(x=[w[0] for w in sarc_top], y=[w[1] for w in sarc_top], palette='Blues_d')
plt.title('Top 10 Words in Sarcastic Utterances')
plt.xlabel('Word')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.show()

# Sarcastic Word Cloud
sarcastic_wc = WordCloud(width=800, height=400, background_color='white', colormap='viridis', max_words=150)
sarcastic_wc.generate_from_frequencies(sarcastic_words) # Use the counter

# Non-Sarcastic Word Cloud
non_sarcastic_wc = WordCloud(width=800, height=400, background_color='white', colormap='plasma', max_words=150)
non_sarcastic_wc.generate_from_frequencies(non_sarcastic_words) # Use the counter
```

```
# --- Plot the Word Clouds ---
plt.figure(figsize=(18, 8))

# Display Sarcastic Word Cloud
plt.subplot(1, 2, 1) # 1 row, 2 columns, 1st plot
plt.imshow(sarcastic_wc, interpolation='bilinear')
plt.axis('off') # Turn off axis lines and labels
plt.title('Word Cloud for Sarcastic Utterances', fontsize=16)

# Display Non-Sarcastic Word Cloud
plt.subplot(1, 2, 2) # 1 row, 2 columns, 2nd plot
plt.imshow(non_sarcastic_wc, interpolation='bilinear')
plt.axis('off') # Turn off axis lines and labels
plt.title('Word Cloud for Non-Sarcastic Utterances', fontsize=16)

plt.tight_layout(pad=2.0) # Adjust layout to prevent overlap
plt.show()
```

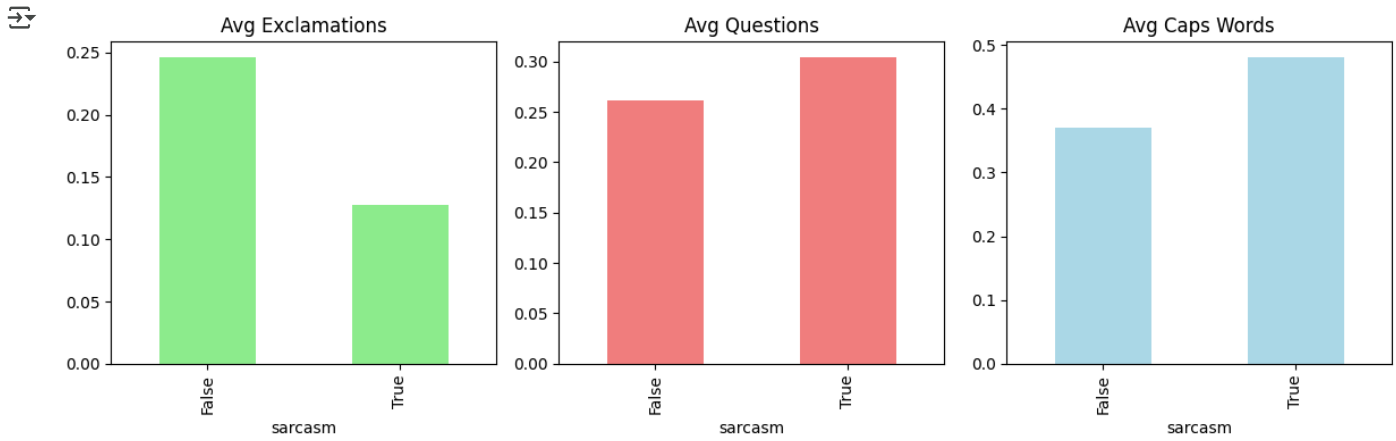
```
Top Sarcastic Words: [('oh', 43), ('like', 28), ('know', 25), ('yeah', 20), ('right', 20), ('really', 20), ('good', 18
Top Non-Sarcastic Words: [('oh', 29), ('yeah', 23), ('well', 22), ('go', 20), ('know', 19), ('like', 18), ('okay', 17)
<ipython-input-8-2f756b9ff098>:30: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hu

  sns.barplot(x=[w[0] for w in sarc_top], y=[w[1] for w in sarc_top], palette='Blues_d')
```



Step 6: Punctuation and Capitalization

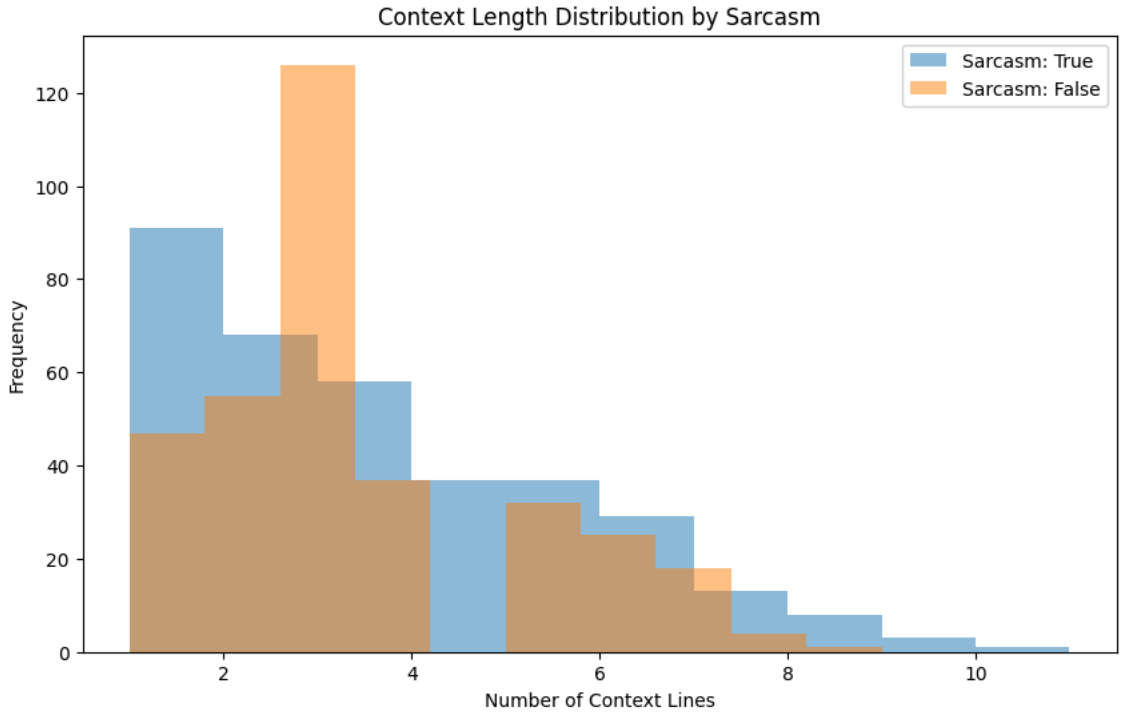
```
import re

# Count punctuation and caps
data['exclamation'] = data['utterance'].apply(lambda x: len(re.findall(r'!', str(x))))
data['question'] = data['utterance'].apply(lambda x: len(re.findall(r'\?', str(x))))
data['caps_words'] = data['utterance'].apply(lambda x: sum(1 for w in str(x).split() if w.isupper()))
```

```
# Plot
plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 1)
data.groupby('sarcasm')['exclamation'].mean().plot(kind='bar', color='lightgreen')
plt.title('Avg Exclamations')
plt.subplot(1, 3, 2)
data.groupby('sarcasm')['question'].mean().plot(kind='bar', color='lightcoral')
plt.title('Avg Questions')
plt.subplot(1, 3, 3)
data.groupby('sarcasm')['caps_words'].mean().plot(kind='bar', color='lightblue')
plt.title('Avg Caps Words')
plt.tight_layout()
plt.show()
```



Step 7: Context Analysis

```
# Context length
data['context_len'] = data['context'].apply(len)  # Number of previous lines

# Plot
plt.figure(figsize=(10, 6))
for label in [True, False]:
    subset = data[data['sarcasm'] == label]
    plt.hist(subset['context_len'], bins=10, alpha=0.5, label=f'Sarcasm: {label}')
plt.title('Context Length Distribution by Sarcasm')
plt.xlabel('Number of Context Lines')
plt.ylabel('Frequency')
plt.legend()
plt.show()

# Context word frequency (simplified)
context_sarc = Counter()
context_non_sarc = Counter()
for idx, row in data.iterrows():
    words = clean_text(' '.join(row['context']))
    if row['sarcasm']:
        context_sarc.update(words)
    else:
        context_non_sarc.update(words)
print("Top Context Words (Sarcastic):", context_sarc.most_common(10))
```

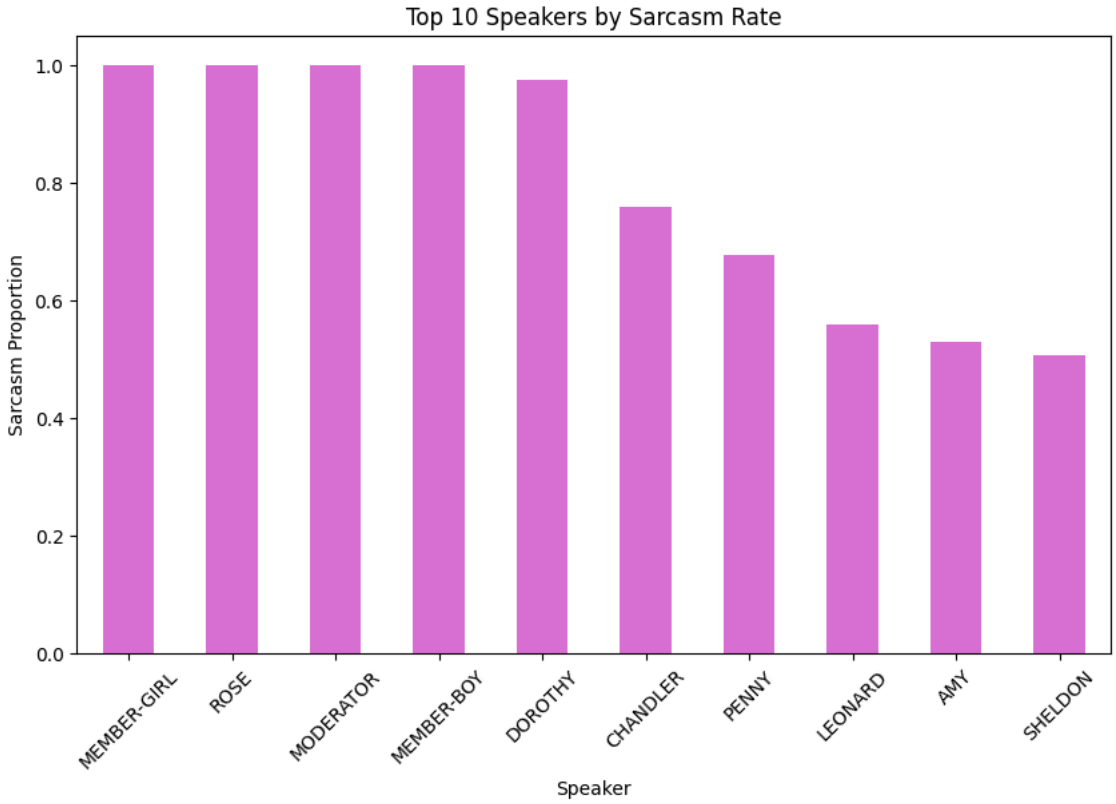



```
Top Context Words (Sarcastic): [('oh', 73), ('know', 65), ('hey', 43), ('well', 41), ('like', 40), ('would', 34), ('go',
```

Step 8: Speaker Analysis

```
# Sarcasm rate by speaker
speaker_sarcasm = data.groupby('speaker')['sarcasm'].mean().sort_values(ascending=False)
print("Sarcasm Rate by Speaker:\n", speaker_sarcasm.head(10))

# Plot top 10
plt.figure(figsize=(10, 6))
speaker_sarcasm.head(10).plot(kind='bar', color='orchid')
plt.title('Top 10 Speakers by Sarcasm Rate')
plt.xlabel('Speaker')
plt.ylabel('Sarcasm Proportion')
plt.xticks(rotation=45)
plt.show()
```

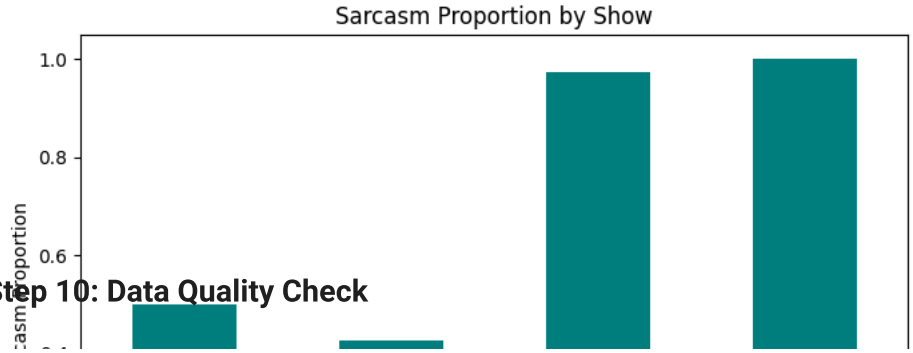
```
Sarcasm Rate by Speaker:
 speaker
MEMBER-GIRL    1.000000
ROSE           1.000000
MODERATOR      1.000000
MEMBER-BOY     1.000000
DOROTHY        0.974359
CHANDLER       0.759494
PENNY          0.676471
LEONARD        0.558824
AMY            0.529412
SHELDON        0.505618
Name: sarcasm, dtype: float64
```



Step 9: Show Analysis

```
# Sarcasm by show
show_sarcasm = data.groupby('show')['sarcasm'].mean()
print("Sarcasm Rate by Show:\n", show_sarcasm)

# Plot
plt.figure(figsize=(8, 5))
show_sarcasm.plot(kind='bar', color='teal')
plt.title('Sarcasm Proportion by Show')
plt.xlabel('Show')
plt.ylabel('Sarcasm Proportion')
plt.xticks(rotation=45)
plt.show()
```

```
Sarcasm Rate by Show:
 show
BBT              0.500000
FRIENDS          0.426966
GOLDENGIRLS      0.975000
SARCASMOHOLICS   1.000000
Name: sarcasm, dtype: float64
```



Step 10: Data Quality Check

```
# Check for missing values
print("Missing Values:\n", data.isnull().sum())
```