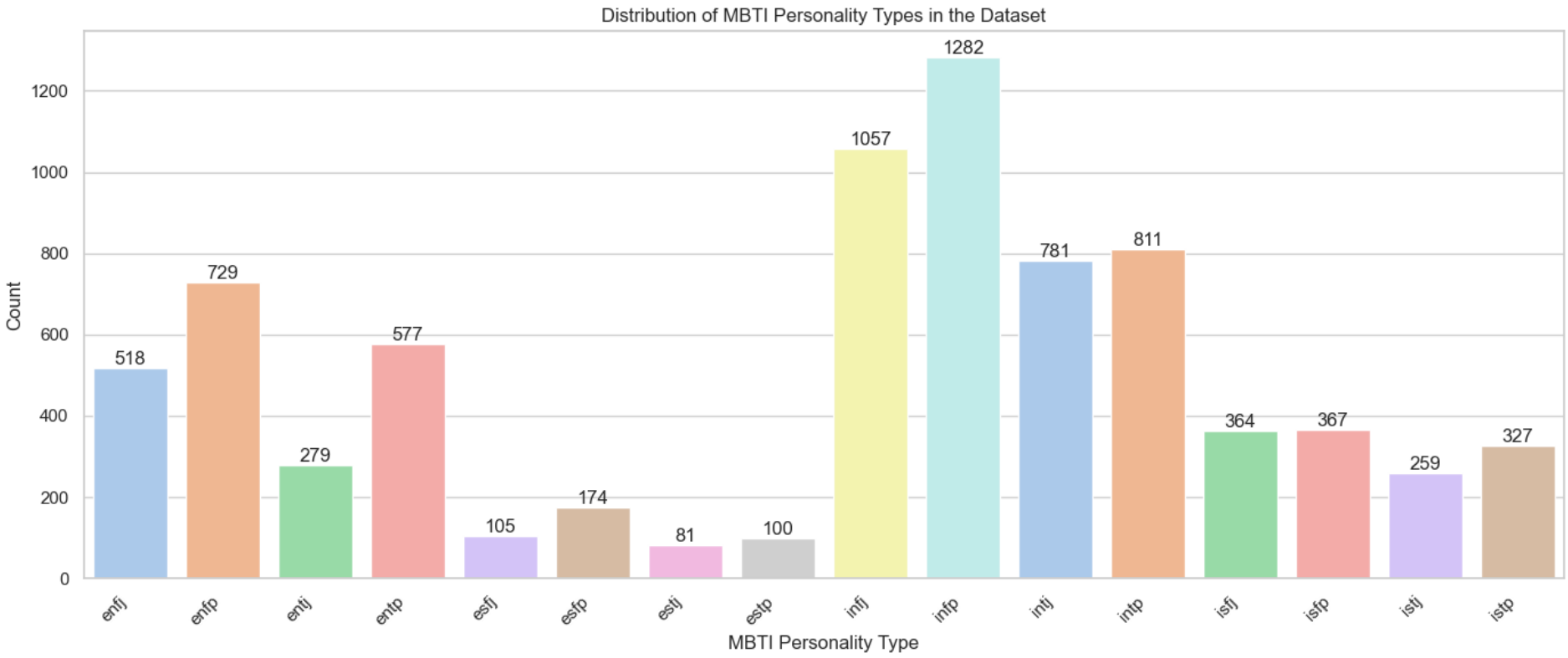
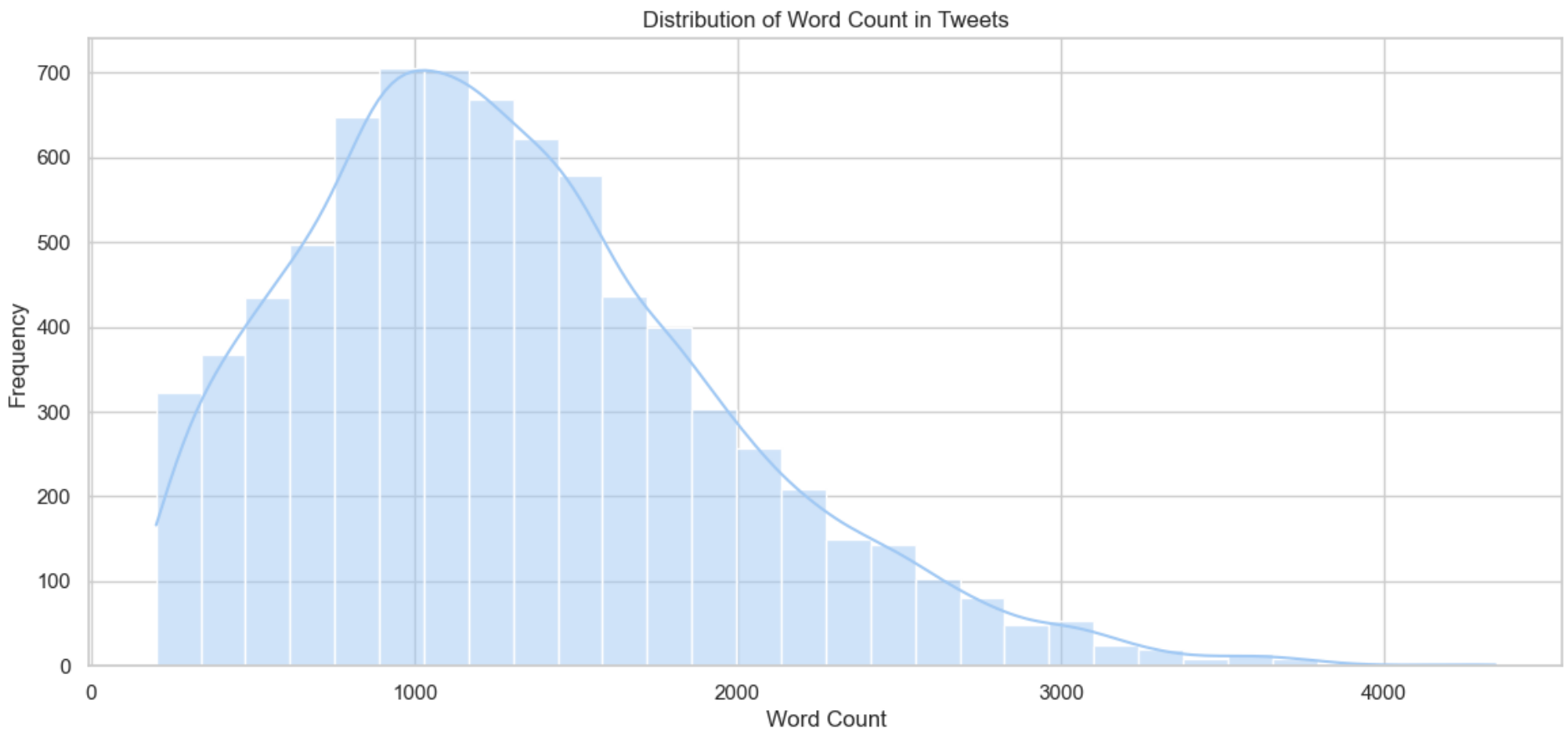


provided creates a count plot (bar chart) using the Seaborn library. The plot shows the distribution of MBTI personality types in the dataset, with the MBTI personality type on the x-axis and the count of each type on the y-axis. The bars are colored using a pastel palette.

In [ ]:
```
# Add a new column for the word count of each tweet
data['word_count'] = data['text'].apply(lambda x: len(x.split()))

plt.figure(figsize=(14, 6))
sns.histplot(data=data, x='word_count', bins=30, kde=True)
plt.title('Distribution of Word Count in Tweets')
plt.xlabel('Word Count')
plt.ylabel('Frequency')
plt.show()
```



- Add a new column named 'word_count' to the 'data' DataFrame. The new column contains the number of words in each 'text' entry (tweet). This is done using the apply() function and a lambda function that splits the text by spaces and counts the resulting words.
- Create a histogram plot using Seaborn's histplot() function. The plot shows the distribution of word counts in the tweets. The 'data' parameter is set to the 'data' DataFrame, the 'x' parameter is set to the 'word_count' column, and the number of bins is set to 30. The 'kde' parameter is set to True, which means a Kernel Density Estimate (KDE) will be plotted over the histogram.
- Set the title, x-axis label, and y-axis label of the histogram plot using plt.title(), plt.xlabel(), and plt.ylabel().

MBTI (The Myers-Briggs Type Indicators)

- More contents: https://www.linkedin.com/in/bhue/
- Download: https://www.kaggle.com/datasets/mazlumi/mbti-personality-type-twitter-dataset?select=twitter_MBTI.csv

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

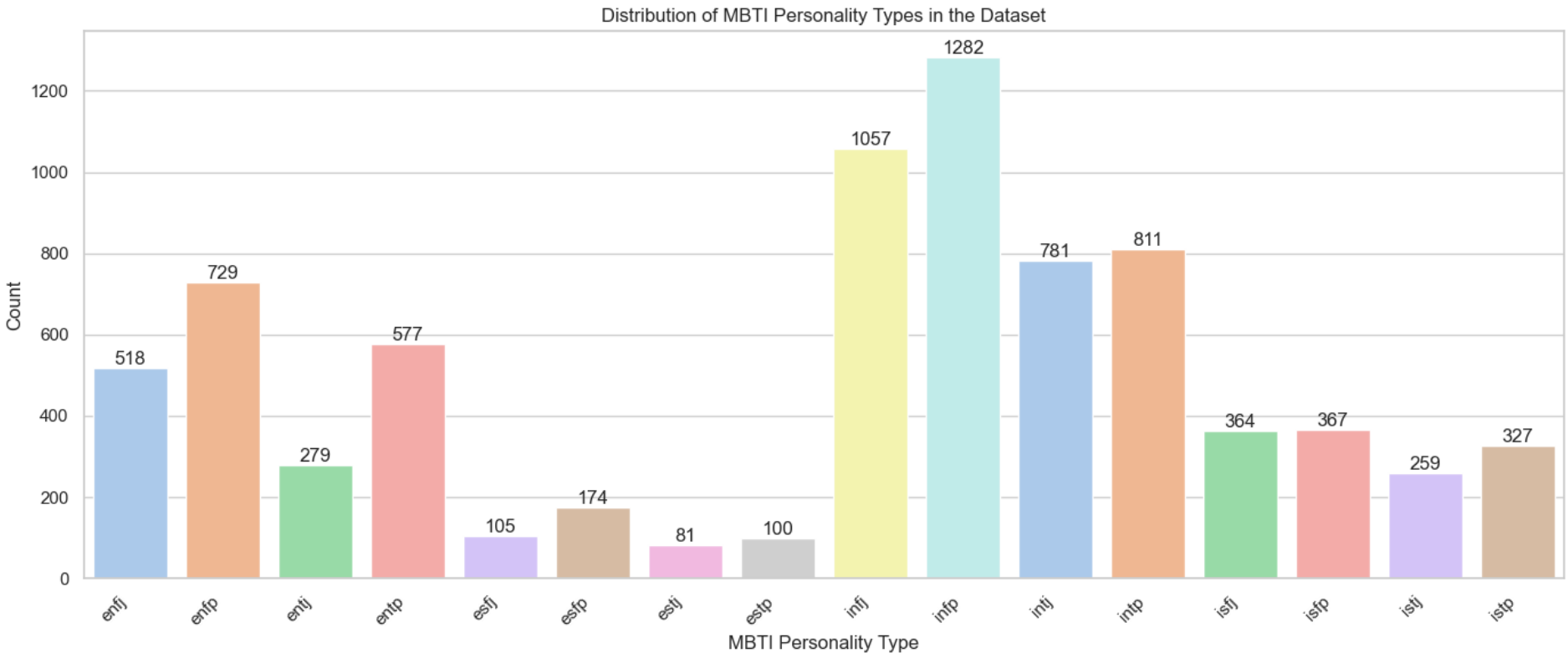
- import pandas as pd: Imports the Pandas library, which is commonly used for data manipulation and analysis. It is imported with the alias pd for convenience.
- import matplotlib.pyplot as plt: Imports the Pyplot module from the Matplotlib library, a popular plotting library in Python. The module is imported with the alias plt for convenience.
- import seaborn as sns: Imports the Seaborn library, which is a statistical data visualization library built on top of Matplotlib. It is imported with the alias sns for convenience.
- import numpy as np: Imports the NumPy library, which is used for numerical computing and working with arrays. It is imported with the alias np for convenience.
- import warnings: Imports the warnings module, which allows you to control how warnings are displayed.
- warnings.filterwarnings('ignore'): This line configures the warnings module to suppress all warnings. This can be useful to keep your output clean, but be cautious when using it, as you might miss important warnings that could indicate issues with your code or data.

In [ ]:
```
data = pd.read_csv('twitter_MBTI_with_descriptions.csv')
```

Data investigation

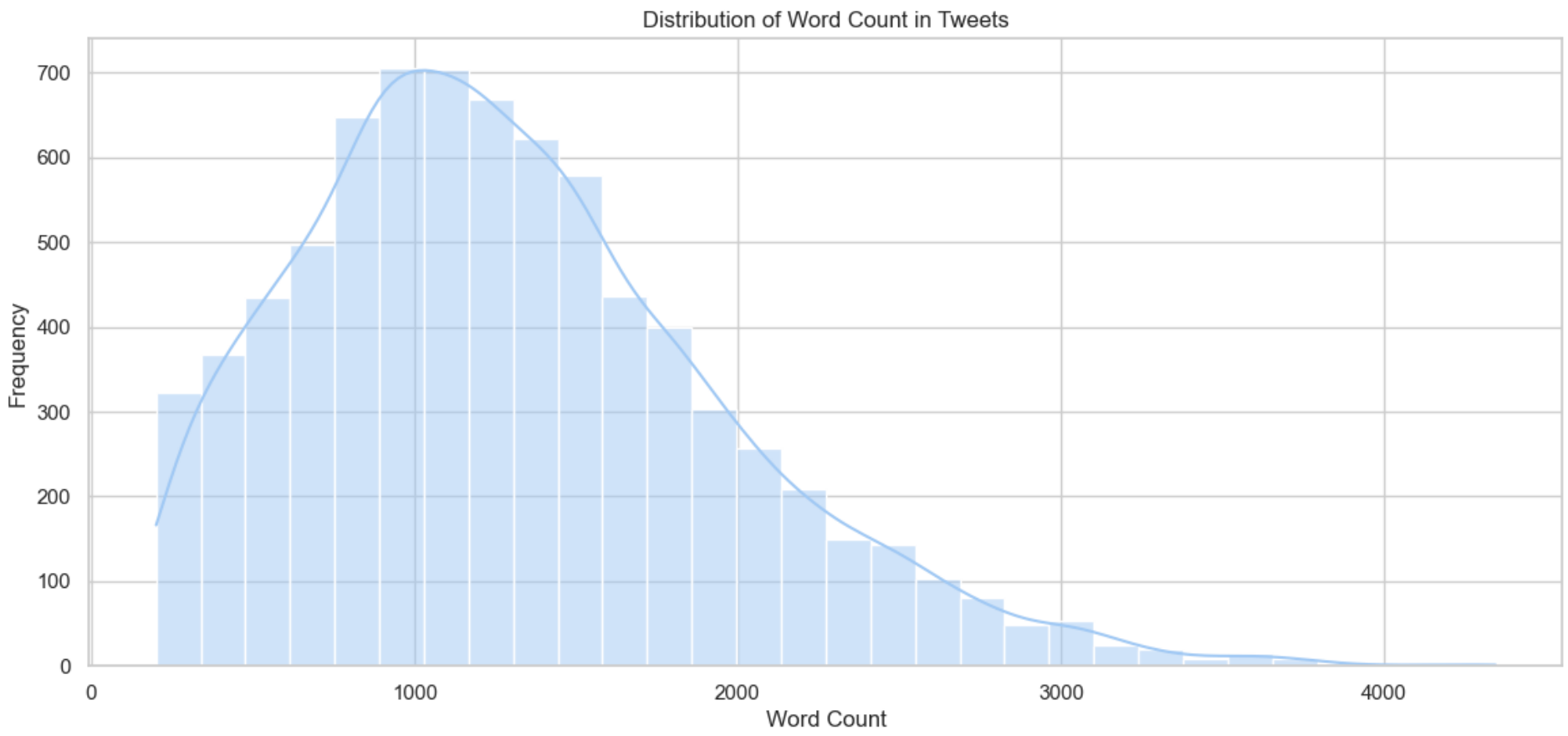
EDA

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Set the style and color palette
sns.set(style="whitegrid", palette="pastel")

# Create count plot with interactive features
plt.figure(figsize=(14, 6))
ax = sns.countplot(data=data, x='label', order=sorted(data['label'].unique()),
                   palette="pastel")
ax.set_title('Distribution of MBTI Personality Types in the Dataset')
ax.set_xlabel('MBTI Personality Type')
ax.set_ylabel('Count')
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha="right")
for p in ax.patches:
    ax.annotate(f'{p.get_height():.0f}', (p.get_x()+p.get_width()/2, p.get_height()),
                ha='center', va='bottom', fontsize=12)
plt.tight_layout()

# Show interactive plot
plt.show()
```



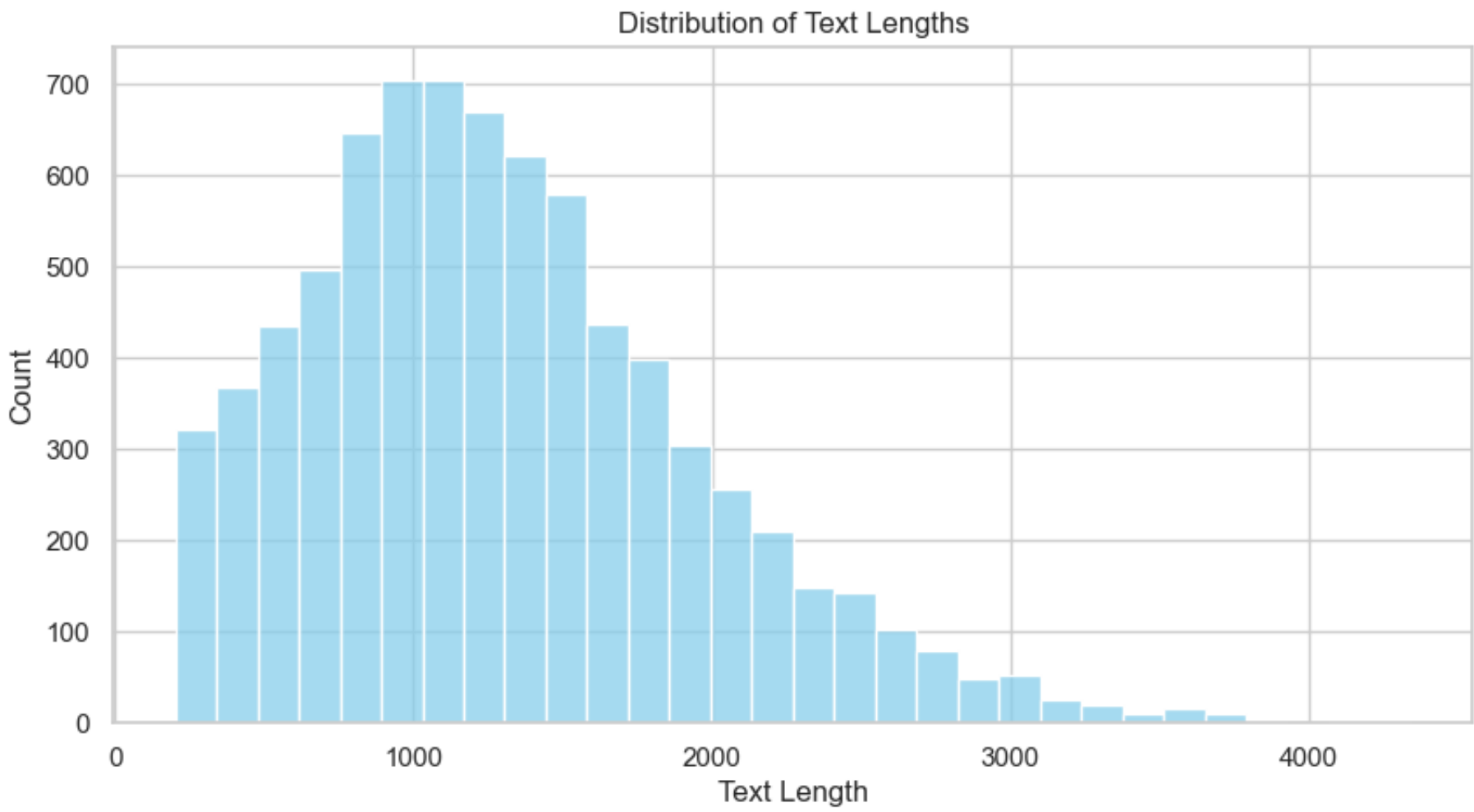
provided creates a count plot (bar chart) using the Seaborn library. The plot shows the distribution of MBTI personality types in the dataset, with the MBTI personality type on the x-axis and the count of each type on the y-axis. The bars are colored using a pastel palette.

In [ ]:
```
# Add a new column for the word count of each tweet
data['word_count'] = data['text'].apply(lambda x: len(x.split()))

plt.figure(figsize=(14, 6))
sns.histplot(data=data, x='word_count', bins=30, kde=True)
plt.title('Distribution of Word Count in Tweets')
plt.xlabel('Word Count')
plt.ylabel('Frequency')
plt.show()
```



- Add a new column named 'word_count' to the 'data' DataFrame. The new column contains the number of words in each 'text' entry (tweet). This is done using the apply() function and a lambda function that splits the text by spaces and counts the resulting words.
- Create a histogram plot using Seaborn's histplot() function. The plot shows the distribution of word counts in the tweets. The 'data' parameter is set to the 'data' DataFrame, the 'x' parameter is set to the 'word_count' column, and the number of bins is set to 30. The 'kde' parameter is set to True, which means a Kernel Density Estimate (KDE) will be plotted over the histogram.
- Set the title, x-axis label, and y-axis label of the histogram plot using plt.title(), plt.xlabel(), and plt.ylabel().

In [ ]:
```
data['text_length'] = data['text'].apply(lambda x: len(x.split()))

plt.figure(figsize=(10, 5))
sns.histplot(data=data, x='text_length', bins=30, color='skyblue')
plt.title('Distribution of Text Lengths')
plt.xlabel('Text Length')
plt.ylabel('Count')
plt.show()
```



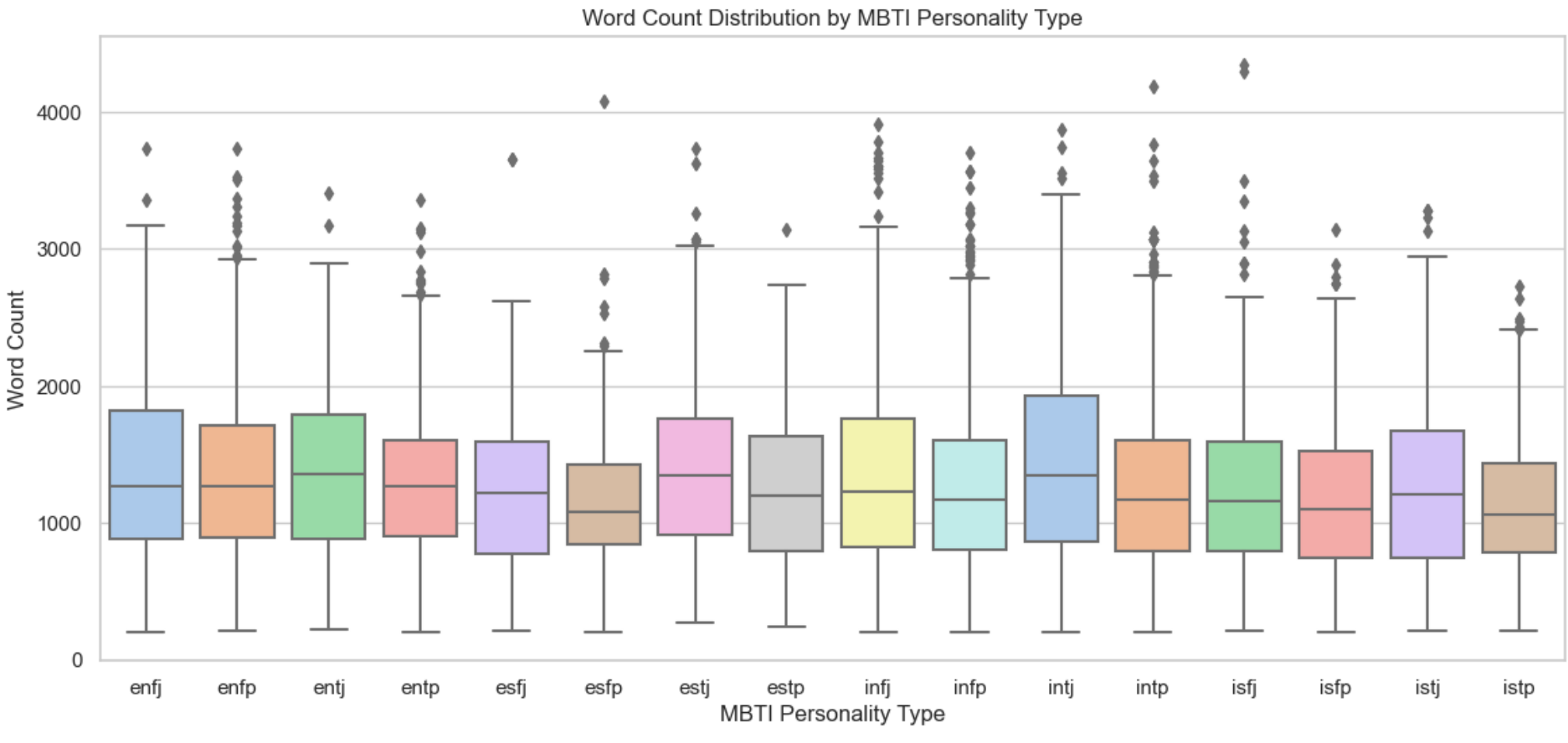
- Add a new column called 'text_length' to the 'data' DataFrame. The values in this column are calculated by counting the number of words in each 'text' column entry using the 'split()' function.
- Configure the Seaborn histogram plot using the 'data' DataFrame, 'text_length' column, and 30 bins. The color of the bars in the histogram is set to 'skyblue'.

In [ ]:
```
import pandas as pd
from IPython.display import display

# Set pandas display options to show full description
pd.set_option('display.max_colwidth', None)

# Create a new DataFrame with only the 'label' and 'Description' columns
label_description_table = data[['label', 'Description']]

# Drop duplicate rows to keep only unique label-description pairs
label_description_table = label_description_table.drop_duplicates()

# Reset the index
label_description_table.reset_index(drop=True, inplace=True)

# Display the resulting table using IPython's display function
display(label_description_table)
```

	label	Description
0	intj	The Mastermind (INTJ): Imaginative, strategic, and determined thinkers who value intellect and competence. They are independent, decisive, and focused on achieving their goals.
1	intp	The Architect (INTP): Innovative, logical, and curious thinkers who are driven by a thirst for knowledge. They are analytical, adaptable, and enjoy exploring abstract ideas and theories.
2	entj	The Commander (ENTJ): Bold, imaginative, and strong-willed leaders who excel at strategic planning and achieving their goals. They are assertive, confident, and value competence and efficiency.
3	entp	The Visionary (ENTP): Creative, enthusiastic, and resourceful individuals who enjoy generating ideas and seeking challenges. They are quick-witted, adaptable, and skilled at seeing new opportunities.
4	infj	The Counselor (INFJ): Insightful, inspiring, and empathetic people who seek meaning and connection in their relationships. They are idealistic, visionary, and have a strong sense of purpose.
5	infp	The Healer (INFP): Idealistic, creative, and nurturing individuals who seek meaning and authenticity in their relationships. They are empathetic, introspective, and value personal growth.
6	enfj	The Teacher (ENFJ): Charismatic, inspiring, and empathetic individuals who are skilled at understanding and motivating others. They are idealistic, visionary, and value personal growth.
7	enfp	The Champion (ENFP): Imaginative, warm, and future-oriented individuals who seek inspiration and connection. They are enthusiastic, optimistic, and enjoy exploring new ideas and possibilities.
8	istj	The Inspector (ISTJ): Practical, fact-minded individuals who are responsible and organized. They value traditions and have a strong work ethic, focusing on details and efficiency.
9	isfj	The Protector (ISFJ): Warm, caring individuals who are committed to others and prioritize harmony. They are observant, empathetic, and loyal, providing practical support to those in need.
10	estj	The Supervisor (ESTJ): Organized, decisive, and practical individuals who excel at managing resources and people. They are logical, assertive, and value structure and efficiency.
11	esfj	The Provider (ESFJ): Warm, loyal, and cooperative individuals who prioritize harmony and social connections. They are empathetic, organized, and excel at providing support to others.
12	istp	The Craftsman (ISTP): Bold, practical, and action-oriented individuals who excel at hands-on tasks. They are adaptable, analytical, and prefer to solve problems in the moment.
13	isfp	The Composer (ISFP): Gentle, artistic, and sensitive individuals who value harmony and personal expression. They are curious, empathetic, and enjoy exploring their surroundings.
14	estp	The Dynamo (ESTP): Energetic, adventurous, and adaptable individuals who thrive in dynamic environments. They are practical, observant, and excel at solving problems in the moment.
15	esfp	The Performer (ESFP): Spontaneous, enthusiastic, and social individuals who enjoy living in the moment. They are outgoing, expressive, and excel at connecting with others.

- Imports the necessary libraries: pandas and IPython.display.
- Sets the pandas display option to show the full description without truncation.
- Creates a new DataFrame called label_description_table by selecting only the 'label' and 'Description' columns from the original dataset (data).
- Drops duplicate rows in the label_description_table DataFrame, keeping only unique label-description pairs.
- Resets the index of the label_description_table DataFrame to start from 0, and drops the old index.
- Displays the resulting label_description_table DataFrame using IPython's display function, which provides a nicely formatted output in Jupyter Notebook or other similar environments.

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Define color palette
color_palette = sns.color_palette("pastel")

# Create box plot with pastel colors and interactivity
plt.figure(figsize=(14, 6))
ax = sns.boxplot(data=data, x='label', y='word_count',
                 order=sorted(data['label'].unique()),
                 palette=color_palette)
ax.set_title('Word Count Distribution by MBTI Personality Type')
ax.set_xlabel('MBTI Personality Type')
ax.set_ylabel('Word Count')
for patch in ax.artists:
    r, g, b, a = patch.get_facecolor()
    patch.set_facecolor((r, g, b, .7))

# Show interactive plot
plt.show()
```

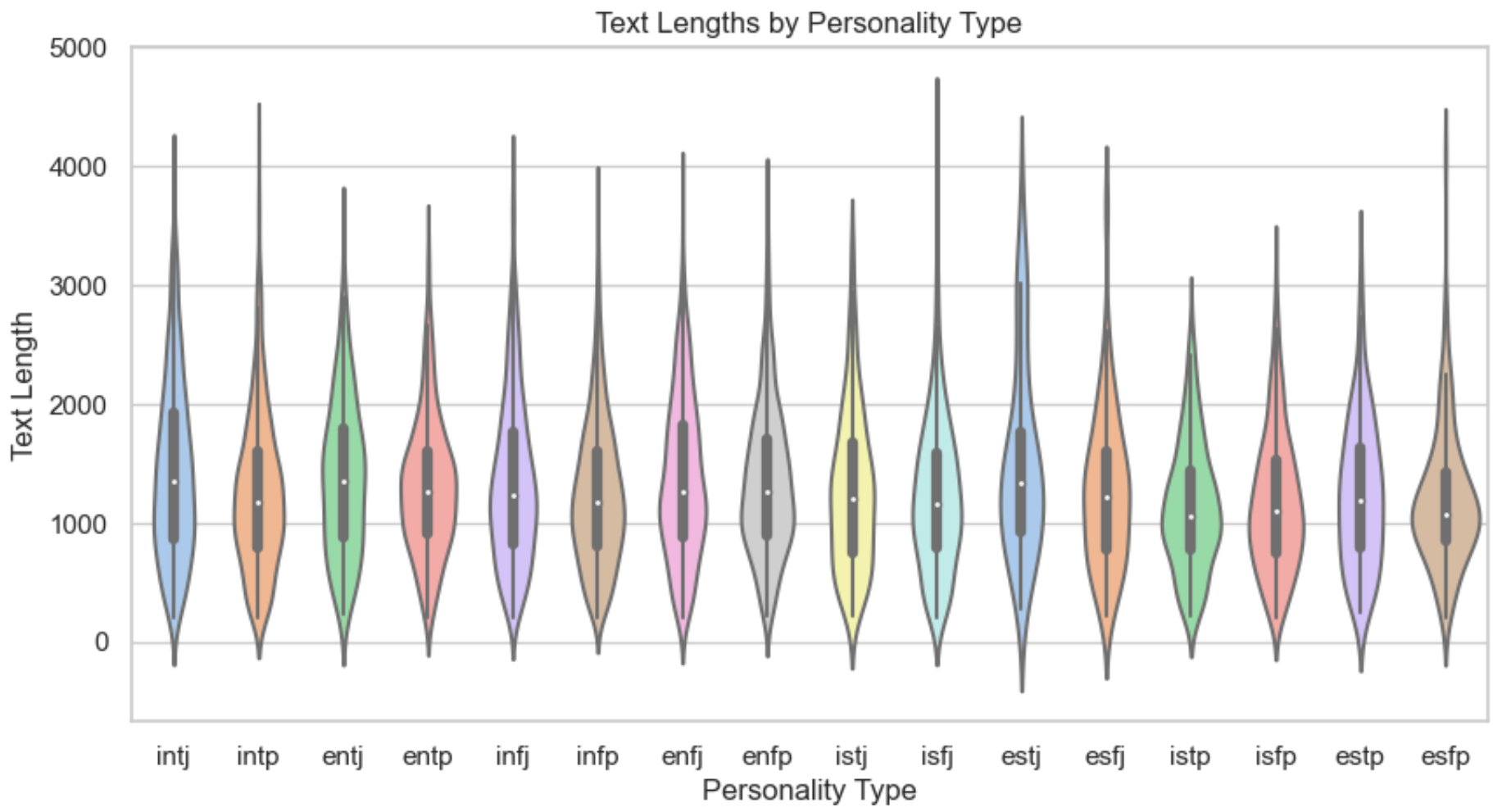


- Define a color palette using seaborn's "pastel" colors.
- Create a new figure with a specified size (14 inches wide and 6 inches tall).
- Create a box plot using seaborn's boxplot function. It takes the following parameters:

data: The dataset to use. x: The column representing the x-axis (in this case, the MBTI personality type labels). y: The column representing the y-axis (in this case, the word count). order: The order in which the MBTI personality types should be displayed on the x-axis (sorted alphabetically). palette: The color palette to use for the plot (the pastel color palette defined earlier).

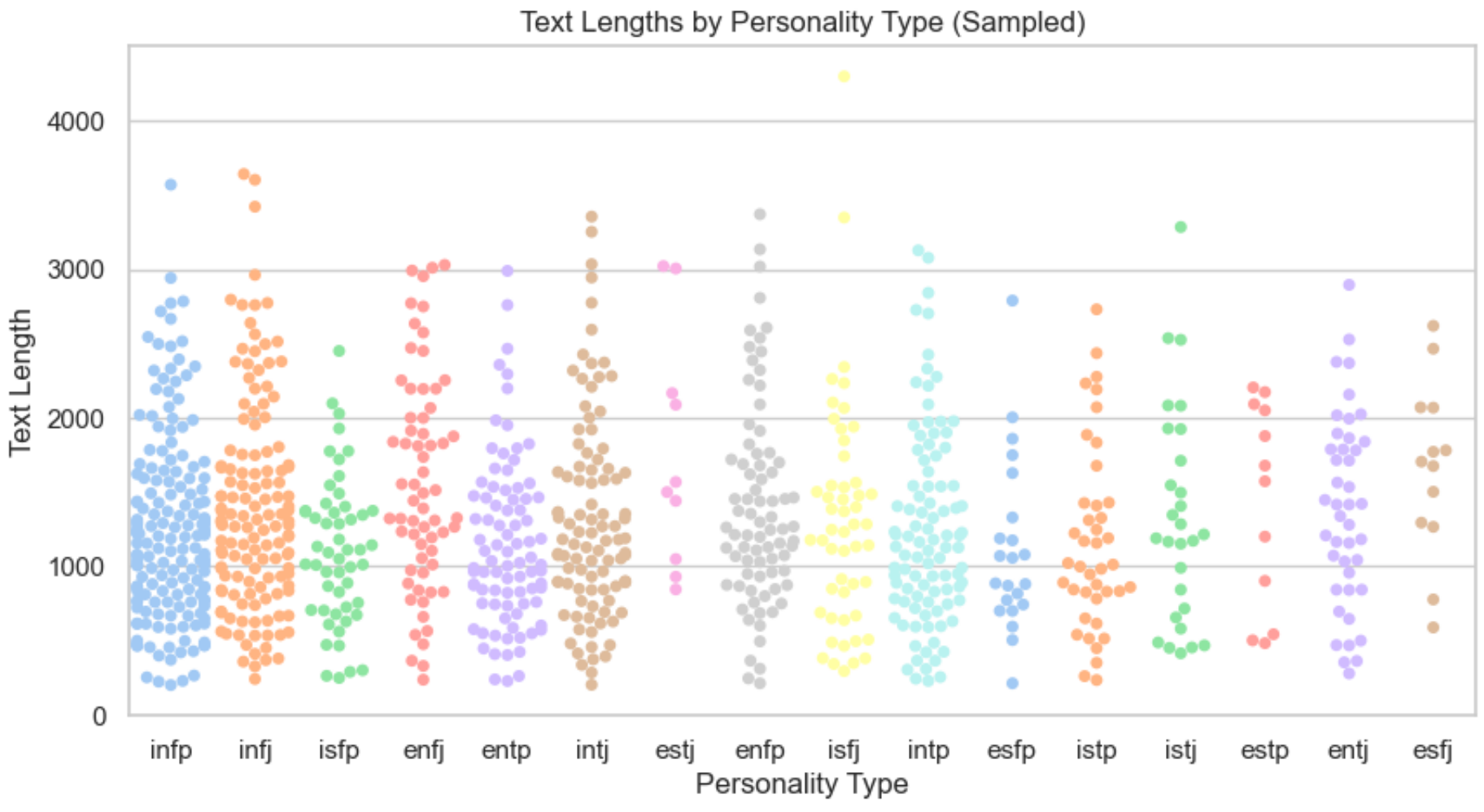
- Adjust the transparency of the box plot's colors (set alpha to 0.7).

In [ ]:
```
plt.figure(figsize=(10, 5))
sns.violinplot(x='label', y='text_length', data=data, palette='pastel')
plt.title('Text Lengths by Personality Type')
plt.xlabel('Personality Type')
plt.ylabel('Text Length')
plt.show()
```



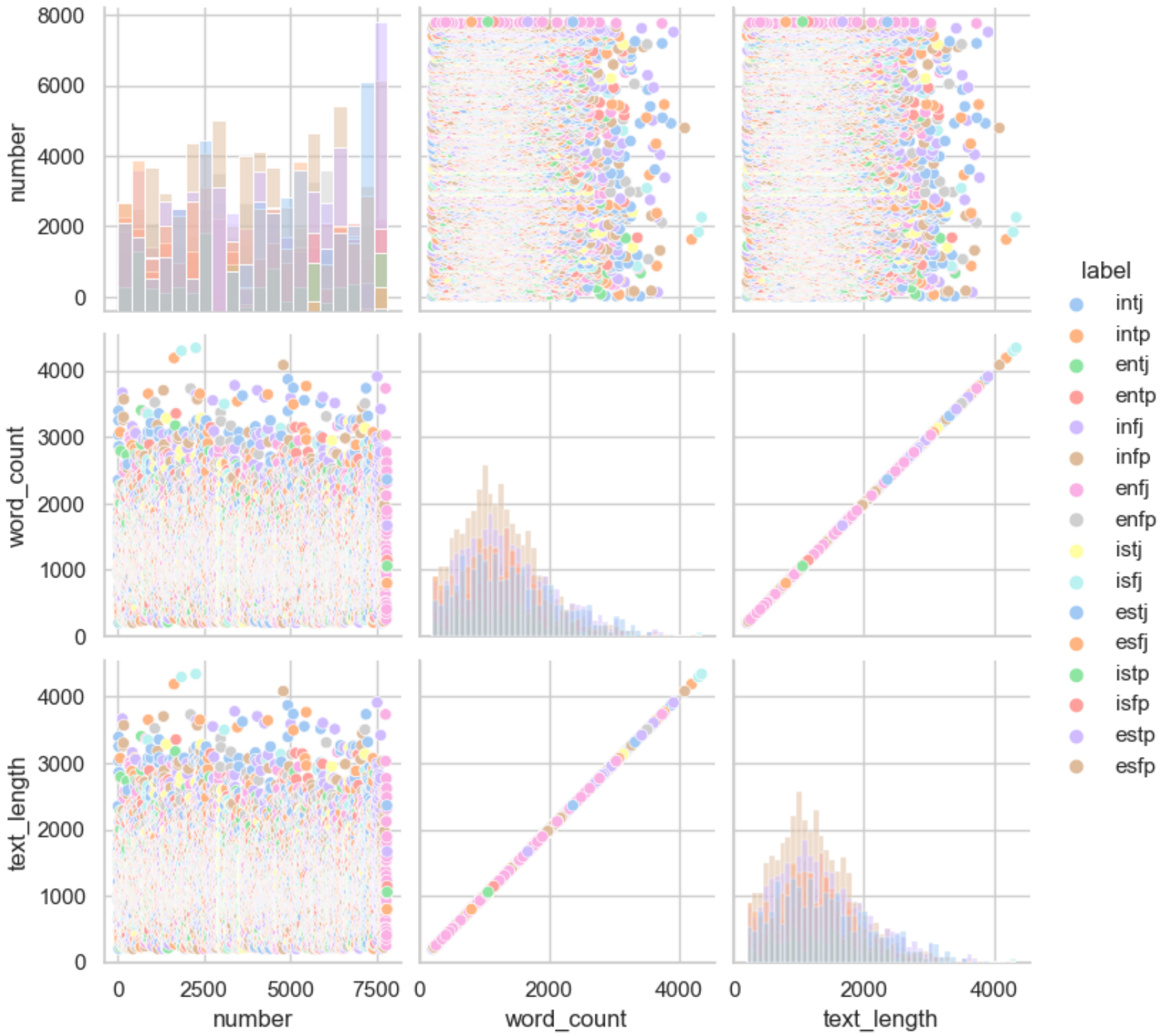
- plt.figure(figsize=(10, 5)): This line creates a new figure with the specified dimensions (10 inches wide and 5 inches tall).
- sns.violinplot(x='label', y='text_length', data=data, palette='pastel'): This line creates a violin plot using Seaborn. The x-axis represents the personality types (label), the y-axis represents the text lengths, and the data comes from the DataFrame named "data". The colors of the violins are defined by the 'pastel' palette.
- plt.title('Text Lengths by Personality Type'): This line adds a title to the plot: "Text Lengths by Personality Type".
- plt.xlabel('Personality Type'): This line labels the x-axis as "Personality Type".

In [ ]:
```
sample_data = data.sample(n=1000, random_state=42)

plt.figure(figsize=(10, 5))
sns.swarmplot(x='label', y='text_length', data=sample_data, palette='pastel')
plt.title('Text Lengths by Personality Type (Sampled)')
plt.xlabel('Personality Type')
plt.ylabel('Text Length')
plt.show()
```

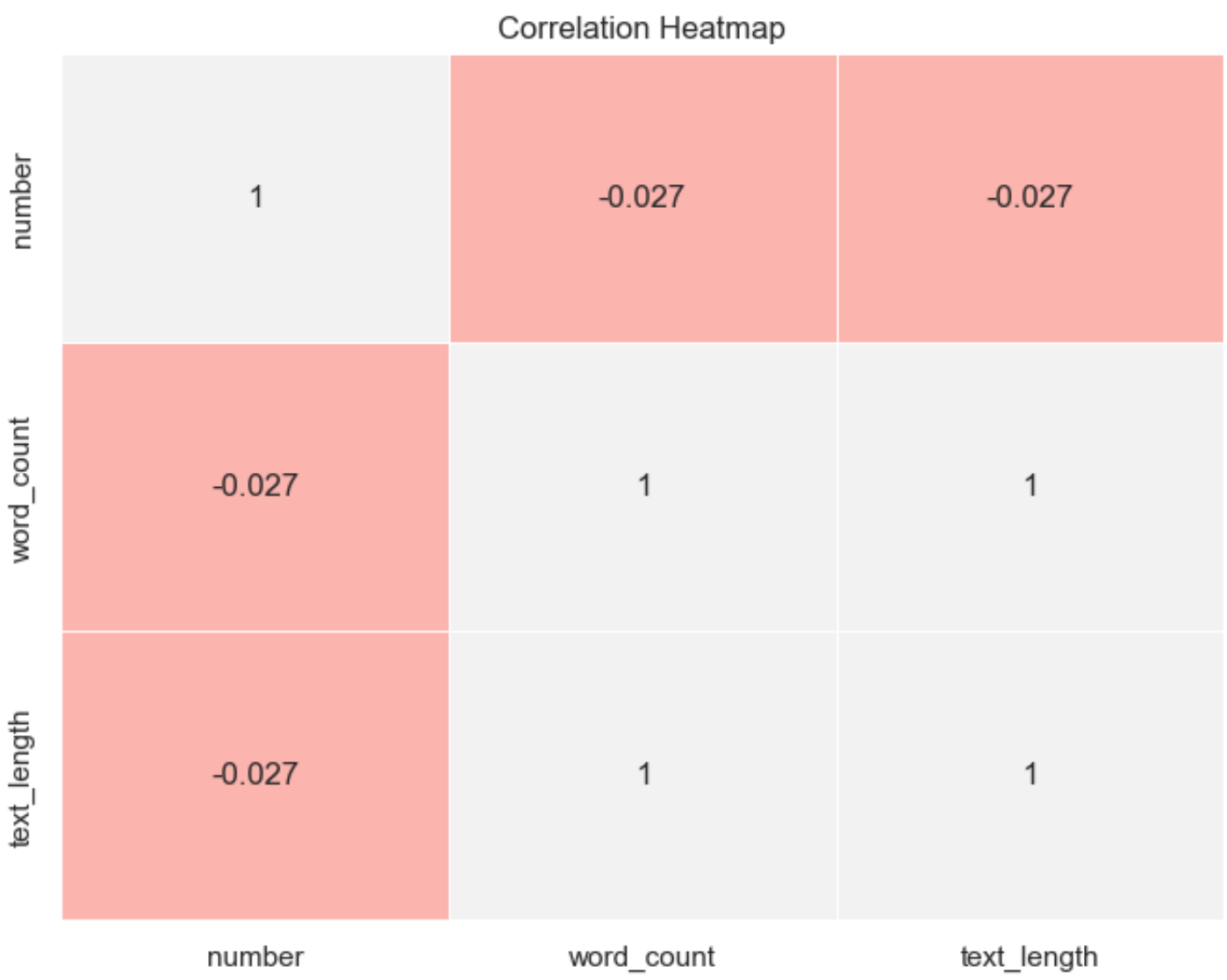


- plt.figure(figsize=(10, 5)): This line creates a new figure with the specified dimensions (10 inches wide and 5 inches tall).
- sns.violinplot(x='label', y='text_length', data=data, palette='pastel'): This line creates a violin plot using Seaborn. The x-axis represents the personality types (label), the y-axis represents the text lengths, and the data comes from the DataFrame named "data". The colors of the violins are defined by the 'pastel' palette.
- plt.title('Text Lengths by Personality Type'): This line adds a title to the plot: "Text Lengths by Personality Type".
- plt.xlabel('Personality Type'): This line labels the x-axis as "Personality Type".

In [ ]:
```
sns.pairplot(data, hue='label', palette='pastel', diag_kind='hist')
plt.show()
```

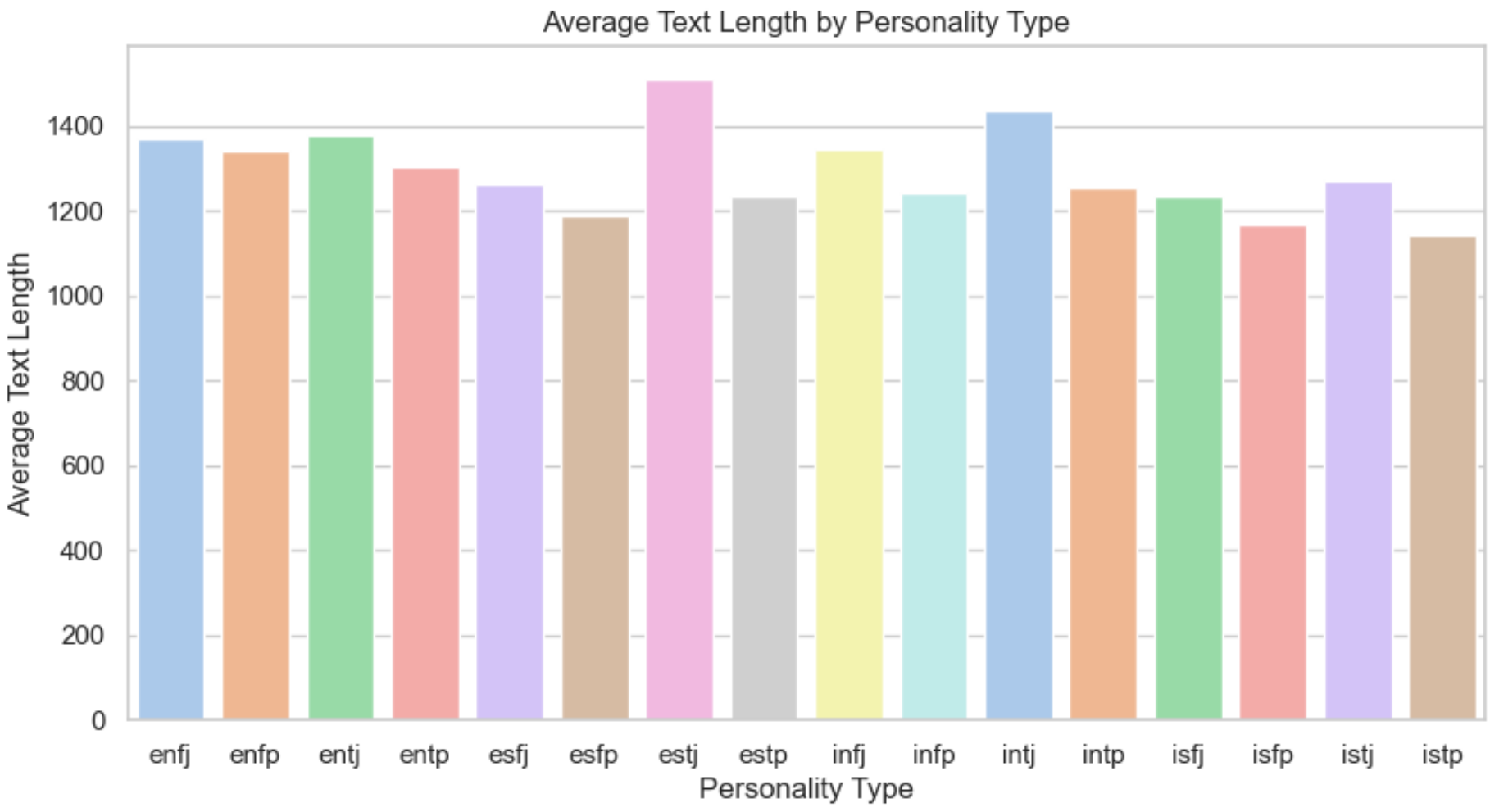


In [ ]:
```
corr = data.corr()

plt.figure(figsize=(8, 6))
sns.heatmap(corr, annot=True, cmap='Pastel1', linewidths=.5, cbar=False)
plt.title('Correlation Heatmap')
plt.show()
```

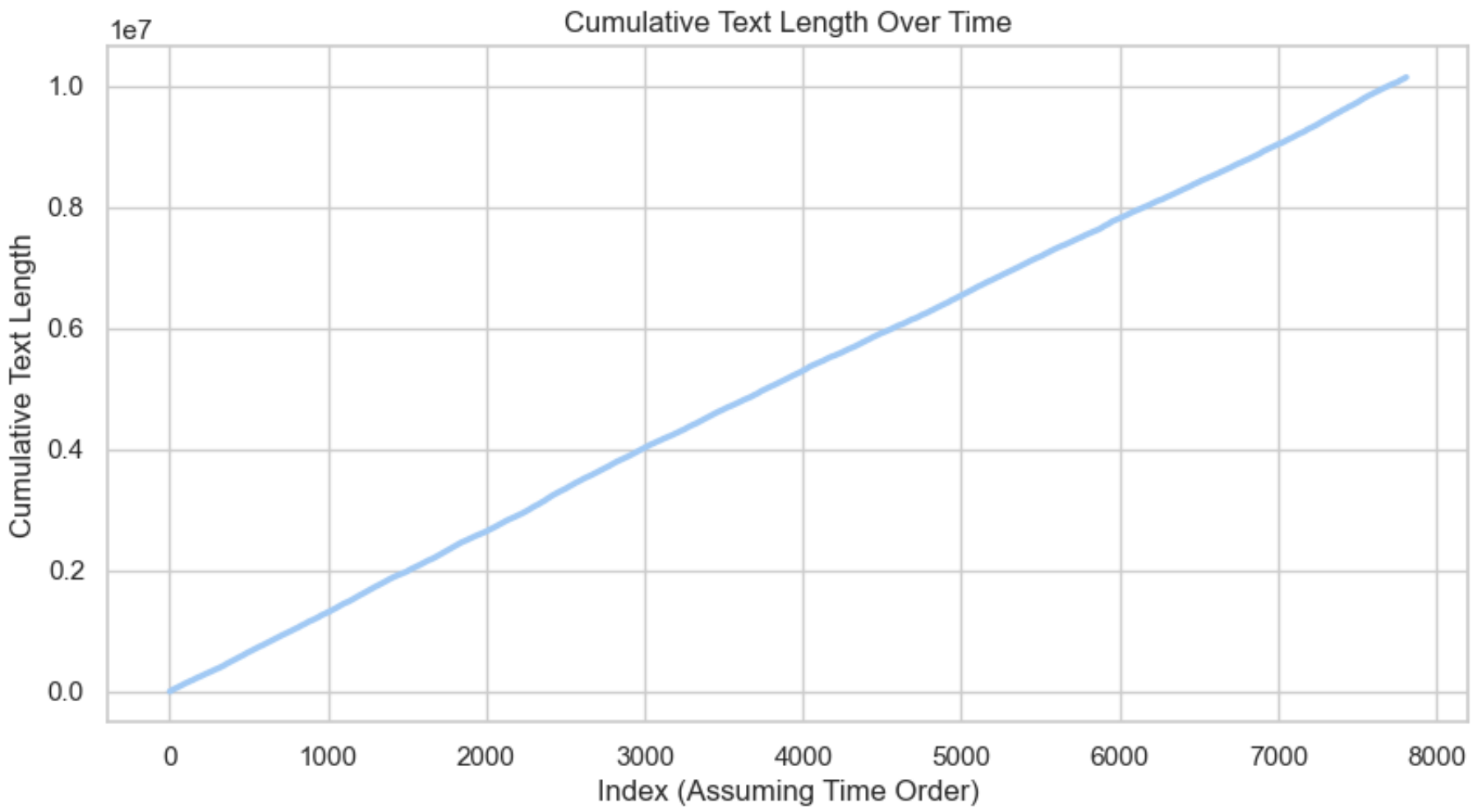


- corr = data.corr(): Compute pairwise correlation of numeric columns in the DataFrame, excluding NA/null values. It returns a correlation matrix, which is a DataFrame.
- plt.figure(figsize=(8, 6)): Create a new figure with the specified width (8) and height (6) in inches.
- sns.heatmap(corr, annot=True, cmap='Pastel1', linewidths=.5, cbar=False): Create a heatmap using the Seaborn library, passing in the correlation matrix, enabling annotations to show the correlation values, setting the color map to 'Pastel1', specifying the line widths between cells as 0.5, and disabling the color bar.

In [ ]:
```
avg_text_length = data.groupby('label')['text_length'].mean().reset_index()

plt.figure(figsize=(10, 5))
sns.barplot(x='label', y='text_length', data=avg_text_length, palette='pastel')
plt.title('Average Text Length by Personality Type')
plt.xlabel('Personality Type')
plt.ylabel('Average Text Length')
plt.show()
```



In [ ]:
```
data['cumulative_text_length'] = data['text_length'].cumsum()

plt.figure(figsize=(10, 5))
sns.lineplot(x=data.index, y='cumulative_text_length', data=data, palette='pastel', linewidth=2.5)
plt.title('Cumulative Text Length Over Time')
plt.xlabel('Index (Assuming Time Order)')
plt.ylabel('Cumulative Text Length')
plt.show()
```



- data['cumulative_text_length'] = data['text_length'].cumsum(): This line calculates the cumulative sum of the 'text_length' column and stores it in a new column called 'cumulative_text_length'.
- plt.figure(figsize=(10, 5)): This line creates a new empty figure with a specified width of 10 inches and a height of 5 inches.
- sns.lineplot(x=data.index, y='cumulative_text_length', data=data, palette='pastel', linewidth=2.5): This line creates a Seaborn line plot using the index of the DataFrame as the x-axis, 'cumulative_text_length' as the y-axis, and uses the 'pastel' color palette with a linewidth of 2.5.

In [ ]:
```
import pandas as pd
import plotly.express as px

# Define color palette
color_palette = px.colors.qualitative.Pastel

# Get value counts for each personality type
label_counts = data["label"].value_counts()

# Create pie chart with pastel colors
fig = px.pie(names=label_counts.index, values=label_counts.values,
             title="Distribution of MBTI Personality Types in the Dataset",
             color_discrete_sequence=color_palette)

# Update chart labels and font
fig.update_traces(textposition="inside", textinfo="percent+label",
                  marker=dict(line=dict(color="#000000", width=1)))
fig.update_layout(title_font=dict(size=24, color="darkblue"),
                  legend=dict(title="MBTI Personality Type",
                              font=dict(size=12, color="black")),
                  font=dict(size=12, color="black"))

# Show interactive plot
fig.show()
```

- Get value counts for each personality type: The code calculates the frequency of each personality type (label) in the dataset.
- Create pie chart with pastel colors: The code creates a pie chart using the plotly.express.pie function, using the personality types (labels) and their respective frequencies.
- Update chart labels and font: The code customizes the appearance of the pie chart by setting the text position, text information, marker lines, title font, legend, and font size and color.

In [ ]:
```
import plotly.express as px

# Define color palette
color_palette = px.colors.qualitative.Pastel

# Calculate word count for each tweet
data['word_count'] = data['text'].apply(lambda x: len(x.split()))

# Create violin plot with pastel colors
fig = px.violin(data, y="word_count", box=True, points="all",
                title="Distribution of Word Count in Tweets",
                color_discrete_sequence=color_palette)
fig.update_yaxes(title="Word Count")
fig.update_xaxes(title="Frequency")

# Show interactive plot
fig.show()
```

- Import Plotly Express as px.
- Define a color palette using Plotly's built-in pastel colors.
- Calculate the word count for each tweet and store it in a new column called 'word_count'.
- Create a violin plot using Plotly Express' px.violin() function. The plot is configured with a box plot inside and displays all data points. The pastel color palette is used for the plot.
- Set the y-axis title to "Word Count" and the x-axis title to "Frequency".
- Display the interactive plot using the fig.show() function.

In [ ]:
```
import plotly.express as px

# Define color palette
color_palette = px.colors.qualitative.Pastel

# Create box plot with color-coded labels
fig = px.box(data, x="label", y="word_count", color="label",
             title="Word Count Distribution by MBTI Personality Type",
             category_orders={"label": sorted(data["label"].unique())},
             color_discrete_sequence=color_palette)

# Update axis labels and font
fig.update_xaxes(title="MBTI Personality Type", showgrid=False,
                 tickfont=dict(size=12, color="black"))
fig.update_yaxes(title="Word Count", showgrid=False,
                 tickfont=dict(size=12, color="black"))

# Update title font and color
fig.update_layout(title_font=dict(size=24, color="darkblue"))

# Show plot
fig.show()
```

In [ ]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Split the DataFrame into train and test sets
train_data, test_data = train_test_split(data, test_size=0.1, random_state=42)

# Encode the labels using LabelEncoder
le = LabelEncoder()
le.fit(data['label'].unique())
train_data['label'] = le.transform(train_data['label'])
test_data['label'] = le.transform(test_data['label'])

# Initialize TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words='english', max_features=1000)

# Fit and transform the text data
X_train = vectorizer.fit_transform(train_data['text'])
X_test = vectorizer.transform(test_data['text'])

# Get the feature names from the vectorizer
feature_names = vectorizer.get_feature_names_out()

# Create a DataFrame from the feature names and their corresponding TF-IDF scores
tfidf_scores = pd.DataFrame(X_train.toarray(), columns=feature_names)

# Group the data by label and calculate the mean of each feature for each label
mean_tfidf_by_label = tfidf_scores.groupby(train_data['label']).mean()

# Display the top 10 features for each label
for idx, row in mean_tfidf_by_label.iterrows():
    label_idx = int(idx)
    label = le.inverse_transform([label_idx])[0]
    print(f"Top 10 features for label {label}:")
    print(row.nlargest(10))
    print("\n")
```

```
Top 10 features for label enfj:
https     0.544064
like      0.088445
just      0.073376
im        0.062178
love      0.054354
don       0.045868
good      0.044032
people    0.038759
thank     0.037399
think     0.036237
Name: 0.0, dtype: float64


Top 10 features for label enfp:
https    0.544385
like     0.087725
im       0.075049
just     0.074734
love     0.055559
good     0.045889
don      0.043349
thank    0.040182
lt       0.039768
know     0.035386
Name: 1.0, dtype: float64


Top 10 features for label entj:
https     0.541774
like      0.085349
just      0.075224
im        0.059616
don       0.046998
love      0.045708
good      0.043065
know      0.040048
people    0.037159
thank     0.036534
Name: 2.0, dtype: float64
```

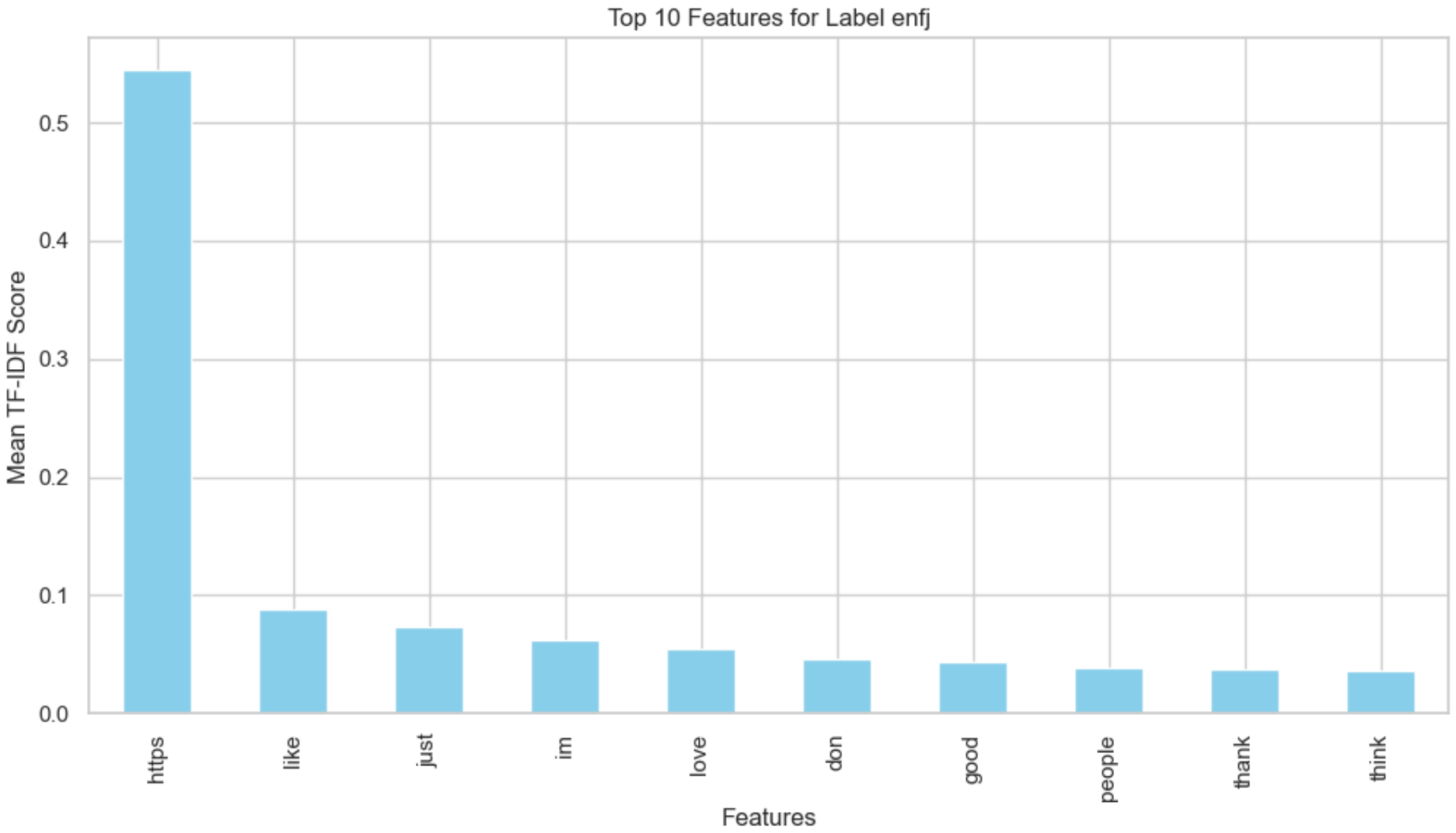
```
Top 10 features for label entp:
https     0.542707
like      0.081749
just      0.073859
im        0.061867
love      0.054395
good      0.046557
don       0.043300
thank     0.041849
lt        0.040374
people    0.035310
Name: 3.0, dtype: float64


Top 10 features for label esfj:
https     0.561008
like      0.078436
just      0.070634
im        0.059814
good      0.048755
love      0.046679
thank     0.046433
don       0.042690
people    0.040582
lt        0.039046
Name: 4.0, dtype: float64


Top 10 features for label esfp:
https    0.531055
like     0.084362
just     0.075558
im       0.073661
love     0.053065
thank    0.048551
good     0.040424
don      0.039235
bc       0.039114
know     0.037974
Name: 5.0, dtype: float64


Top 10 features for label estj:
https     0.571518
like      0.082115
just      0.073212
im        0.066115
love      0.053623
don       0.050215
good      0.047132
ur        0.042847
people    0.040426
oh        0.039707
Name: 6.0, dtype: float64


Top 10 features for label estp:
https    0.552264
like     0.084491
just     0.074036
im       0.071888
love     0.054965
don      0.048087
good     0.040940
think    0.040723
know     0.037247
thank    0.037196
Name: 7.0, dtype: float64


Top 10 features for label infj:
https     0.560947
like      0.087569
just      0.077065
im        0.073290
love      0.053375
good      0.044557
don       0.040778
```

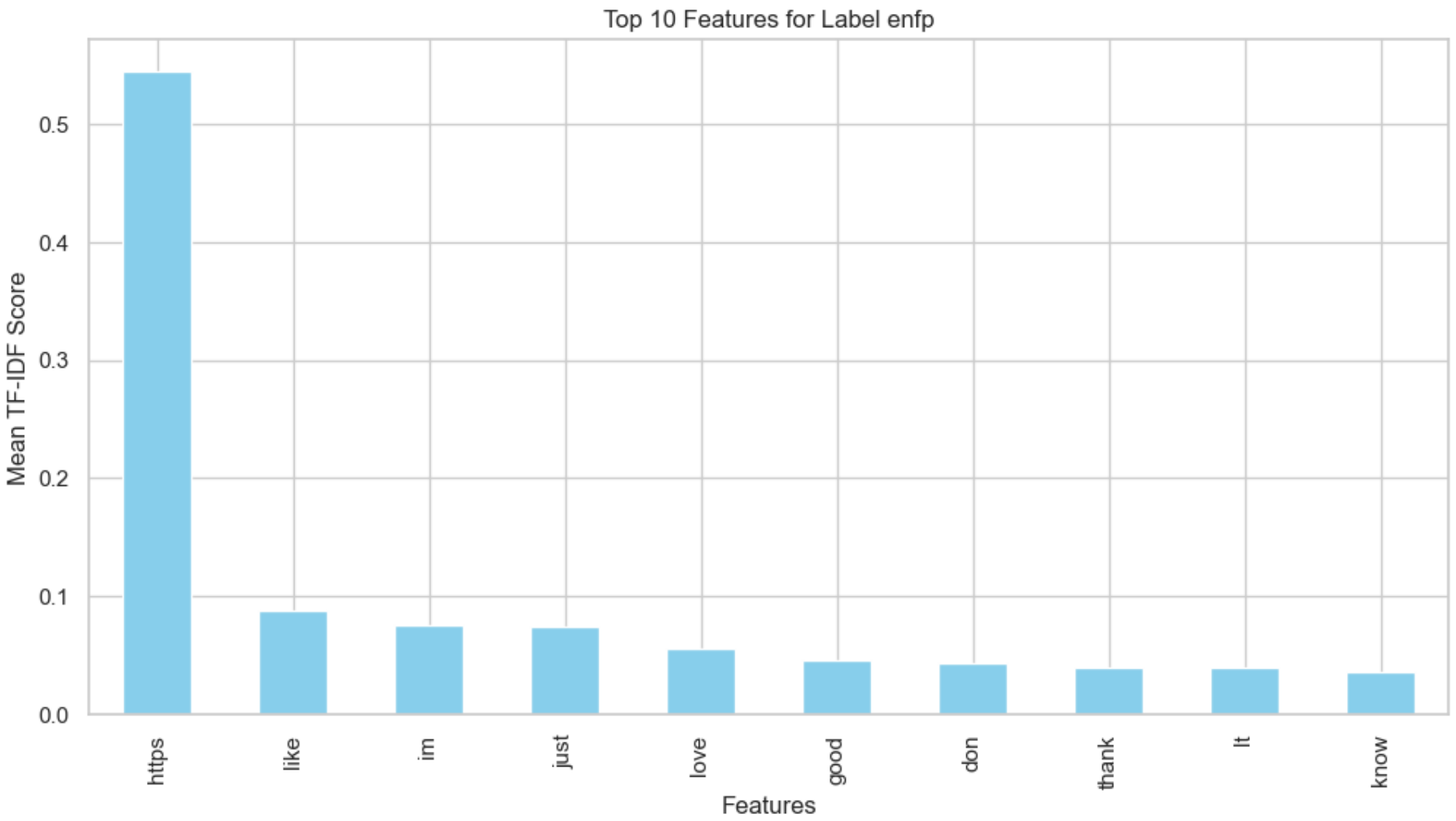
```
thank     0.036204
know      0.035712
people    0.035671
Name: 8.0, dtype: float64


Top 10 features for label infp:
https     0.560719
like      0.085920
just      0.073152
im        0.062472
love      0.052656
don       0.044018
thank     0.041546
good      0.040254
people    0.037387
know      0.036744
Name: 9.0, dtype: float64


Top 10 features for label intj:
https     0.550307
like      0.084191
just      0.074420
im        0.060392
love      0.054960
don       0.045576
good      0.041979
thank     0.040367
know      0.037432
people    0.036651
Name: 10.0, dtype: float64


Top 10 features for label intp:
https    0.557129
like     0.089548
just     0.072059
im       0.067288
love     0.055833
don      0.046009
good     0.045614
omg      0.038897
lt       0.037420
know     0.036872
Name: 11.0, dtype: float64


Top 10 features for label isfj:
https     0.548550
like      0.090753
just      0.077889
im        0.060503
love      0.056123
don       0.046623
good      0.043154
thank     0.042043
people    0.040799
know      0.038099
Name: 12.0, dtype: float64


Top 10 features for label isfp:
https     0.548239
like      0.087374
just      0.073125
im        0.067553
love      0.058253
don       0.050132
good      0.042300
thank     0.041189
people    0.038470
know      0.037891
Name: 13.0, dtype: float64


Top 10 features for label istj:
https    0.535890
like     0.090069
```

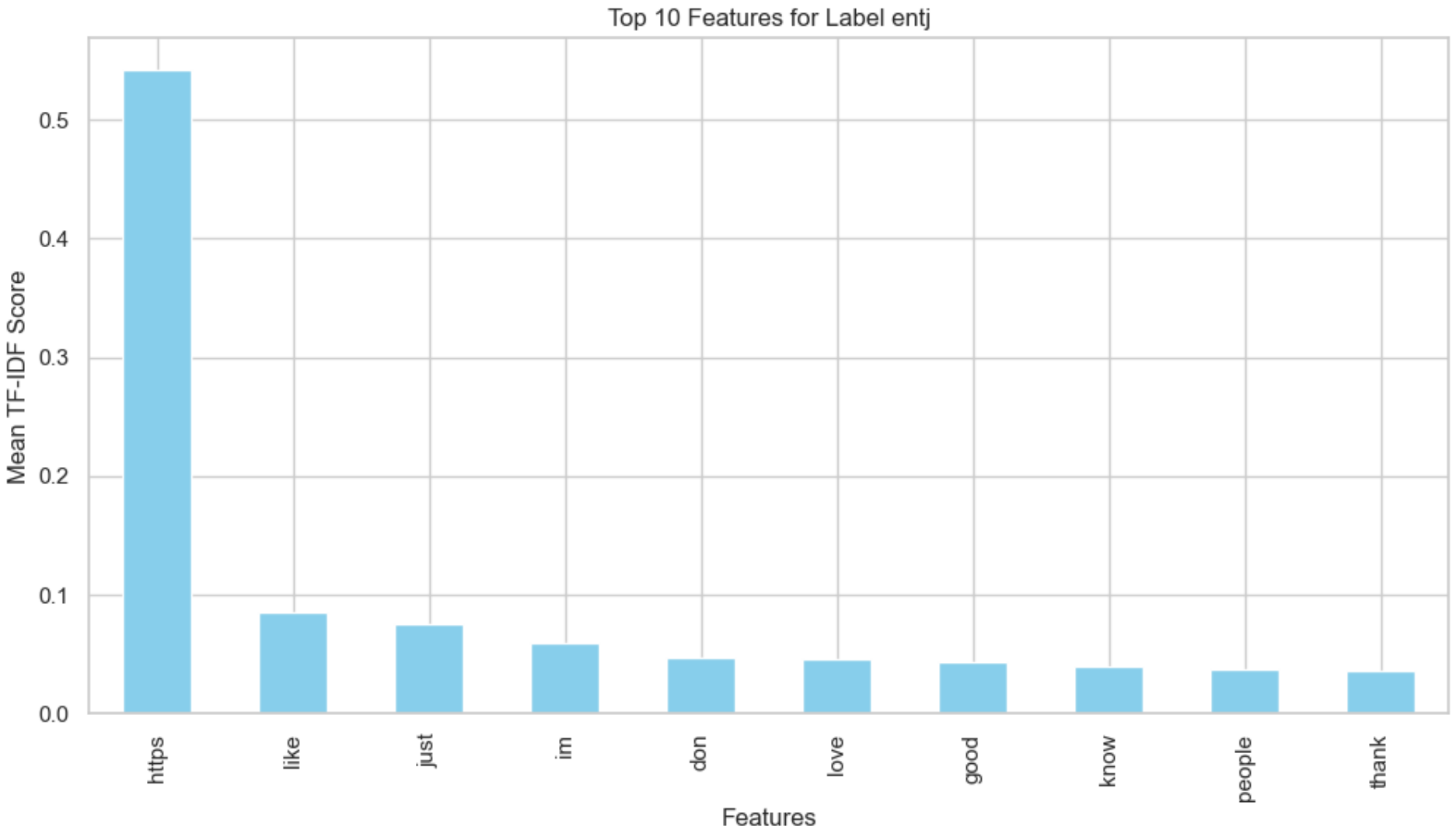
```
just     0.078515
im       0.065596
love     0.052505
don      0.047940
good     0.043867
know     0.040670
amp      0.039617
thank    0.038146
Name: 14.0, dtype: float64


Top 10 features for label istp:
https    0.554382
like     0.084266
just     0.070412
im       0.065598
love     0.051578
don      0.039678
good     0.038423
know     0.036918
omg      0.035138
thank    0.033308
Name: 15.0, dtype: float64
```

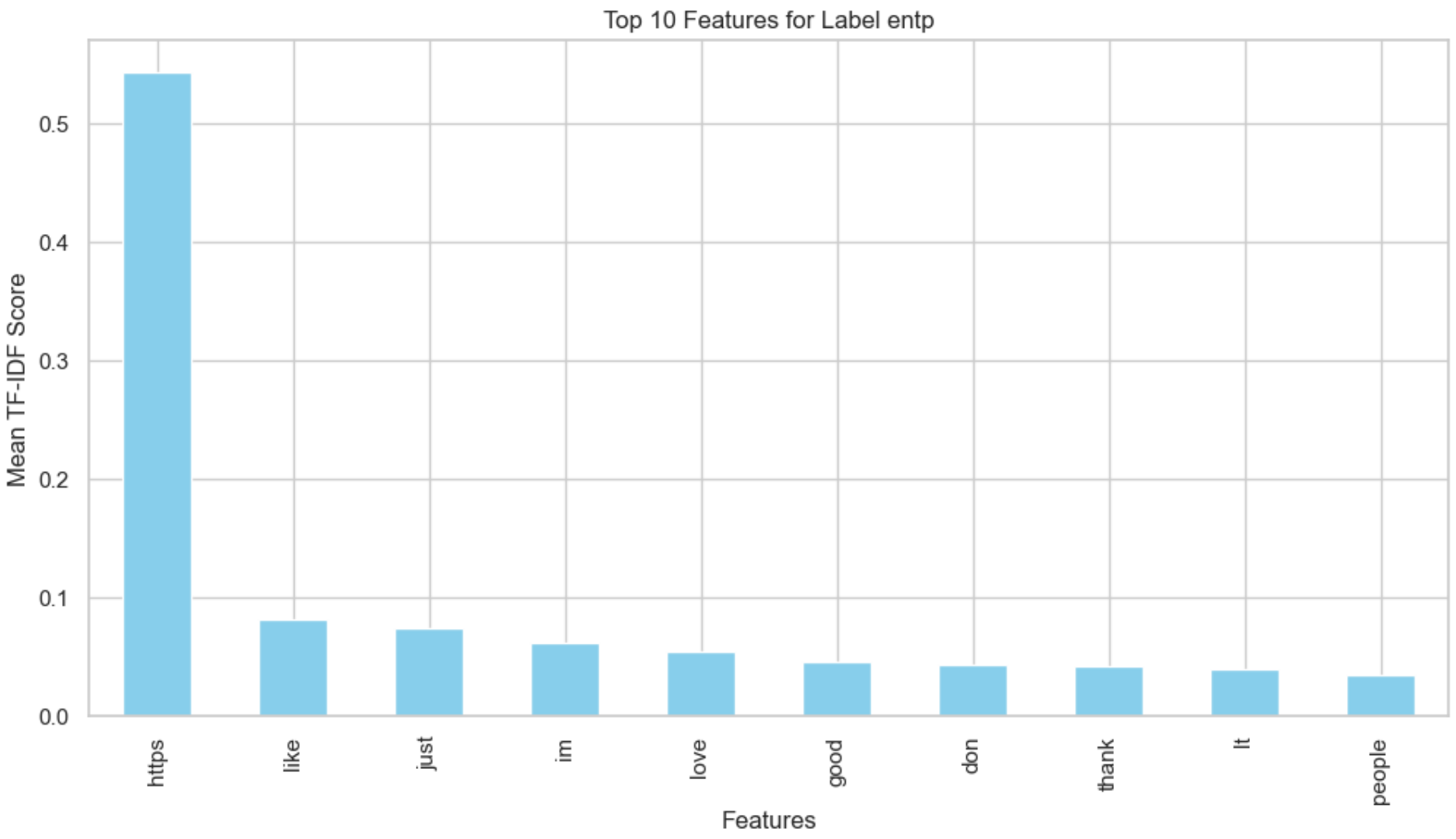
- Split the DataFrame into train and test sets
- Encode the labels using LabelEncoder
- Initialize TfidfVectorizer
- Fit and transform the text data
- Get the feature names from the vectorizer
- Create a DataFrame from the feature names and their corresponding TF-IDF scores
- Group the data by label and calculate the mean of each feature for each label
- Display the top 10 features for each label

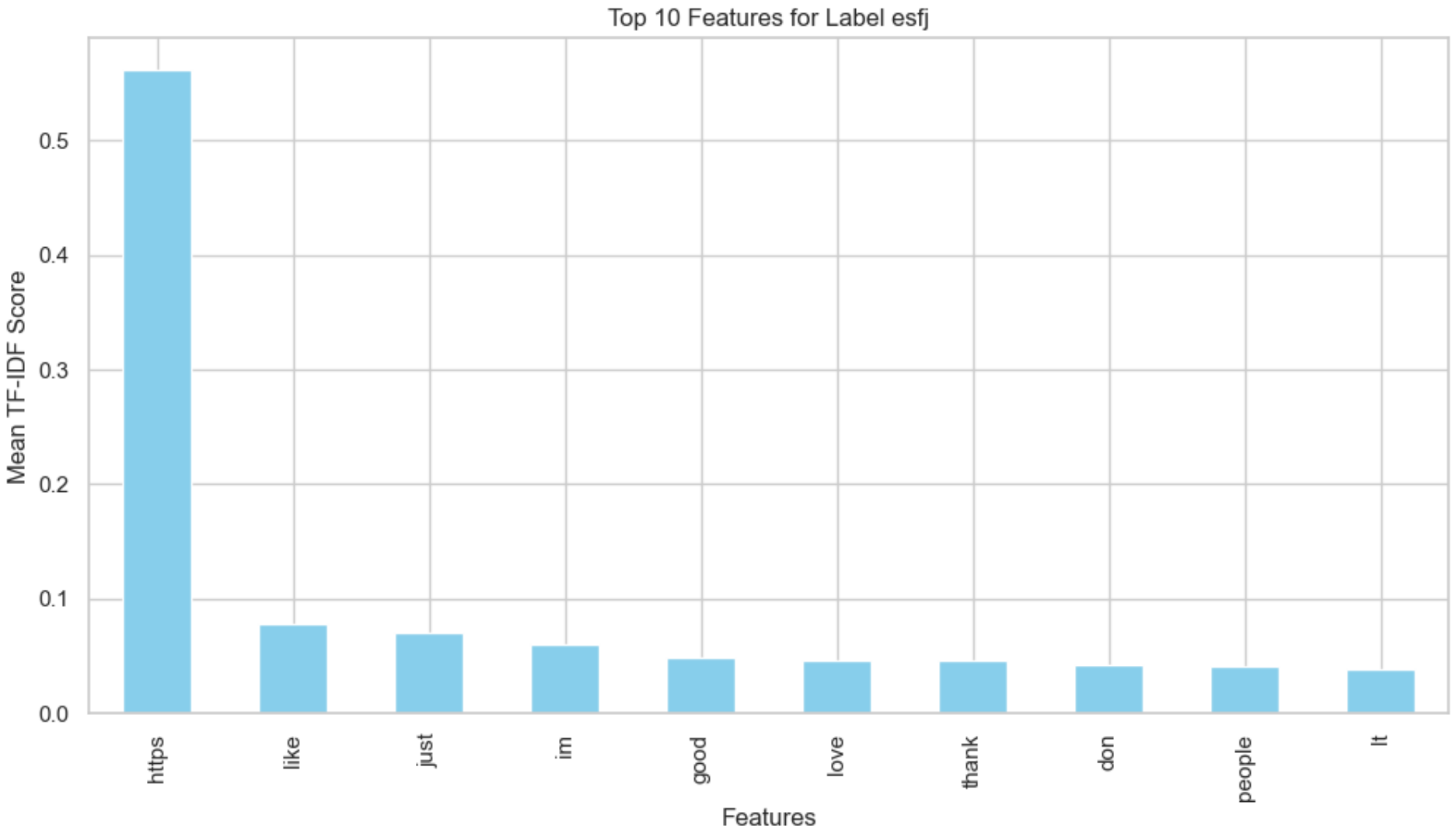
In [ ]:
```python
import matplotlib.pyplot as plt

def plot_top_features(row, label):
    plt.figure(figsize=(12, 6))
    row.nlargest(10).plot(kind='bar', color='skyblue')
    plt.title(f"Top 10 Features for Label {label}")
    plt.xlabel("Features")
    plt.ylabel("Mean TF-IDF Score")
    plt.show()

# Display the top 10 features for each label and plot them
for idx, row in mean_tfidf_by_label.iterrows():
    label_idx = int(idx)
    label = le.inverse_transform([label_idx])[0]
    plot_top_features(row, label)
```

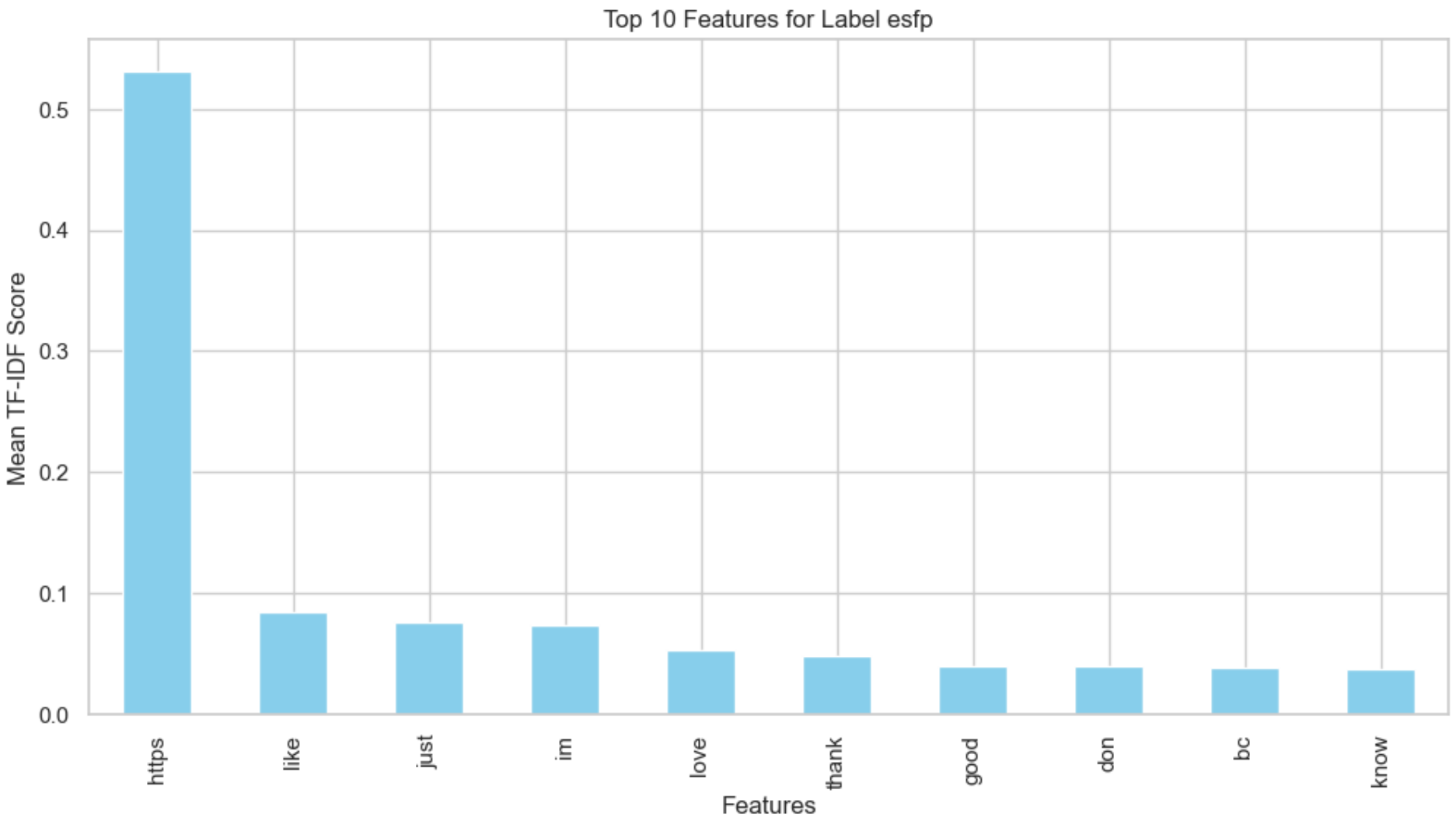

Top 10 Features for Label enfj
Mean TF-IDF Score
0.0
0.1
0.2
0.3
0.4
0.5
https
like
just
im
love
don
good
people
thank
think
Features

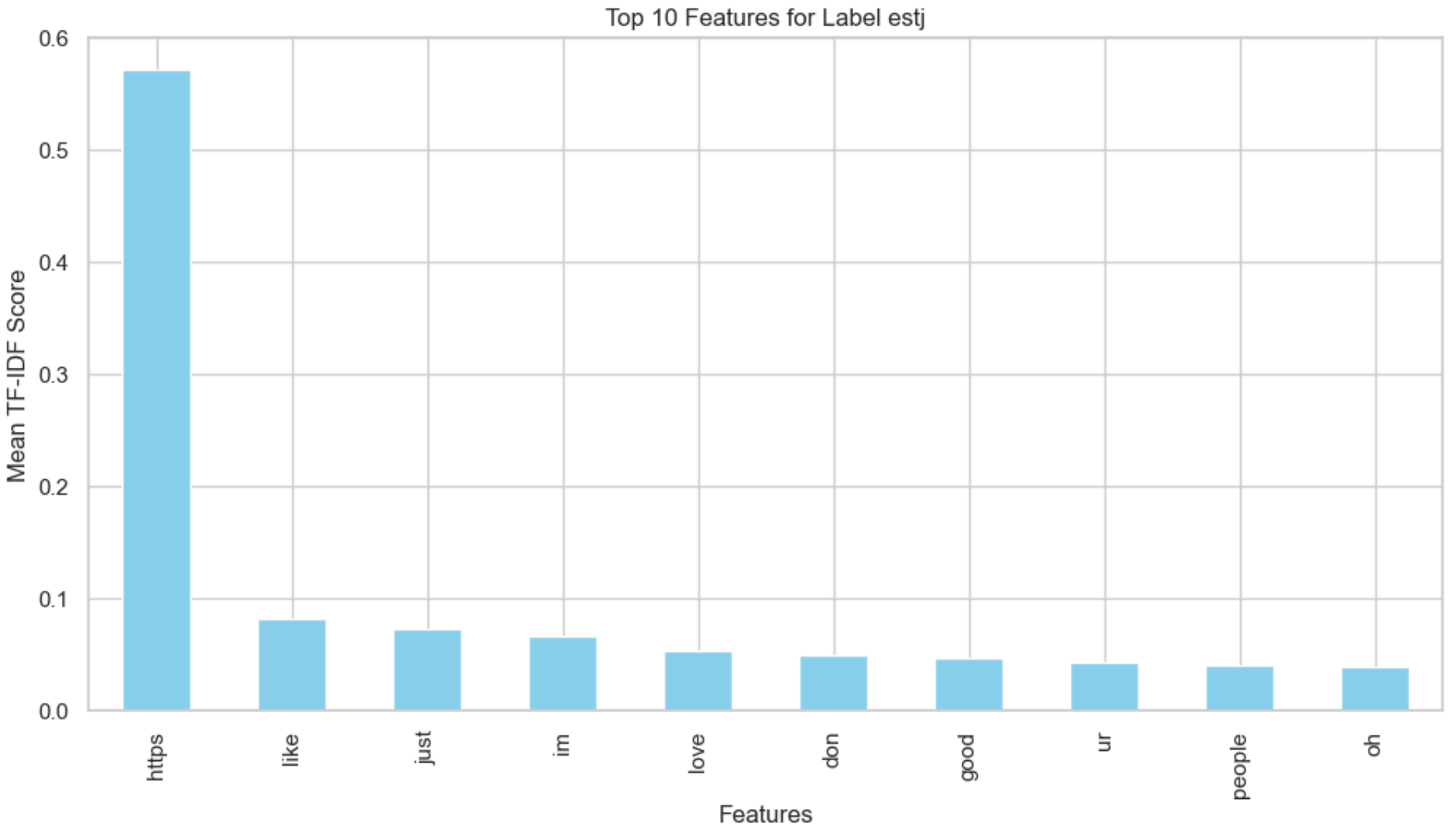

Top 10 Features for Label enfp
Mean TF-IDF Score
0.0
0.1
0.2
0.3
0.4
0.5
https
like
im
just
love
good
don
thank
It
know
Features

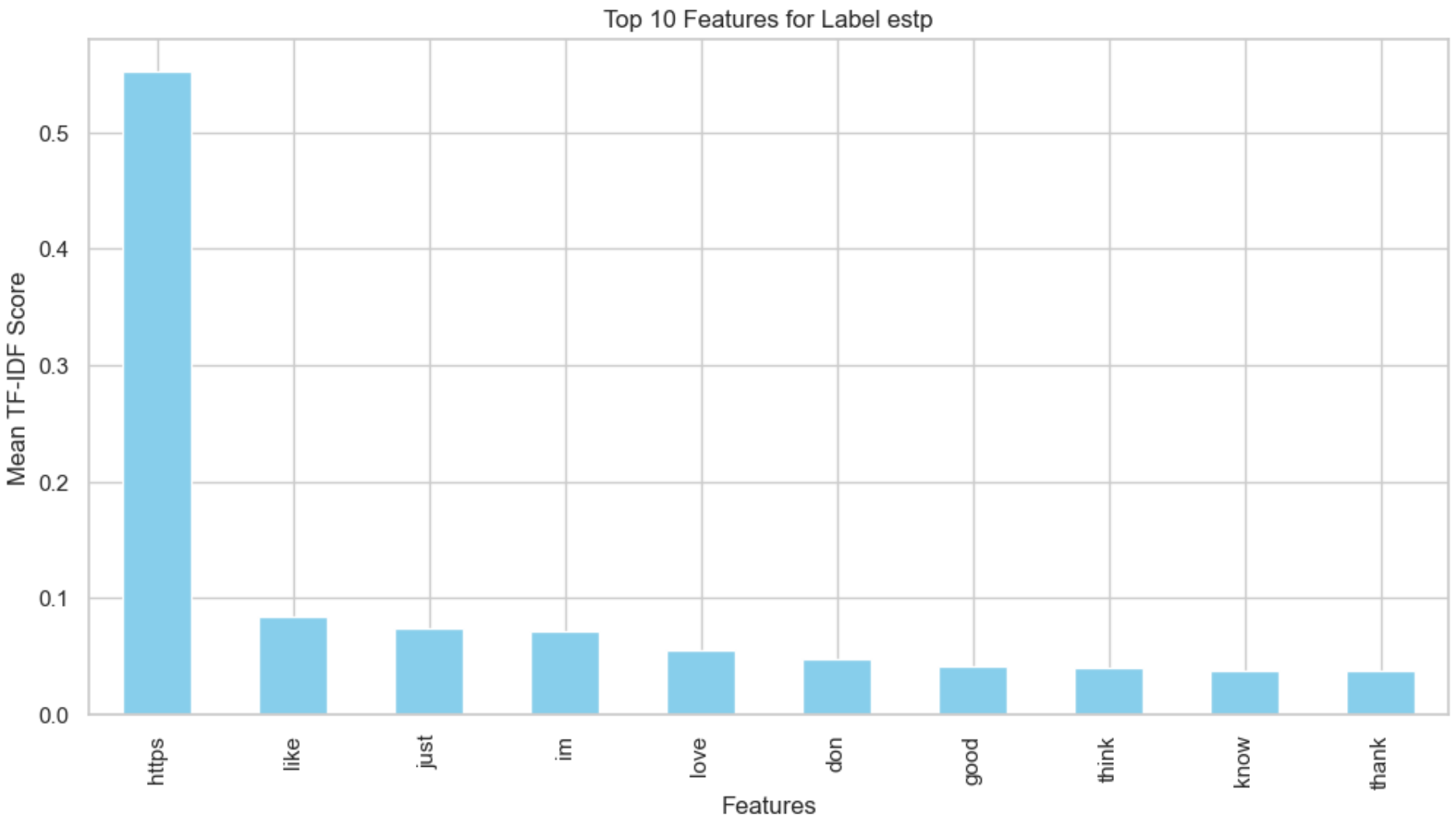

Top 10 Features for Label entj
Mean TF-IDF Score
0.0
0.1
0.2
0.3
0.4
0.5
https
like
just
im
don
love
good
know
people
thank
Features

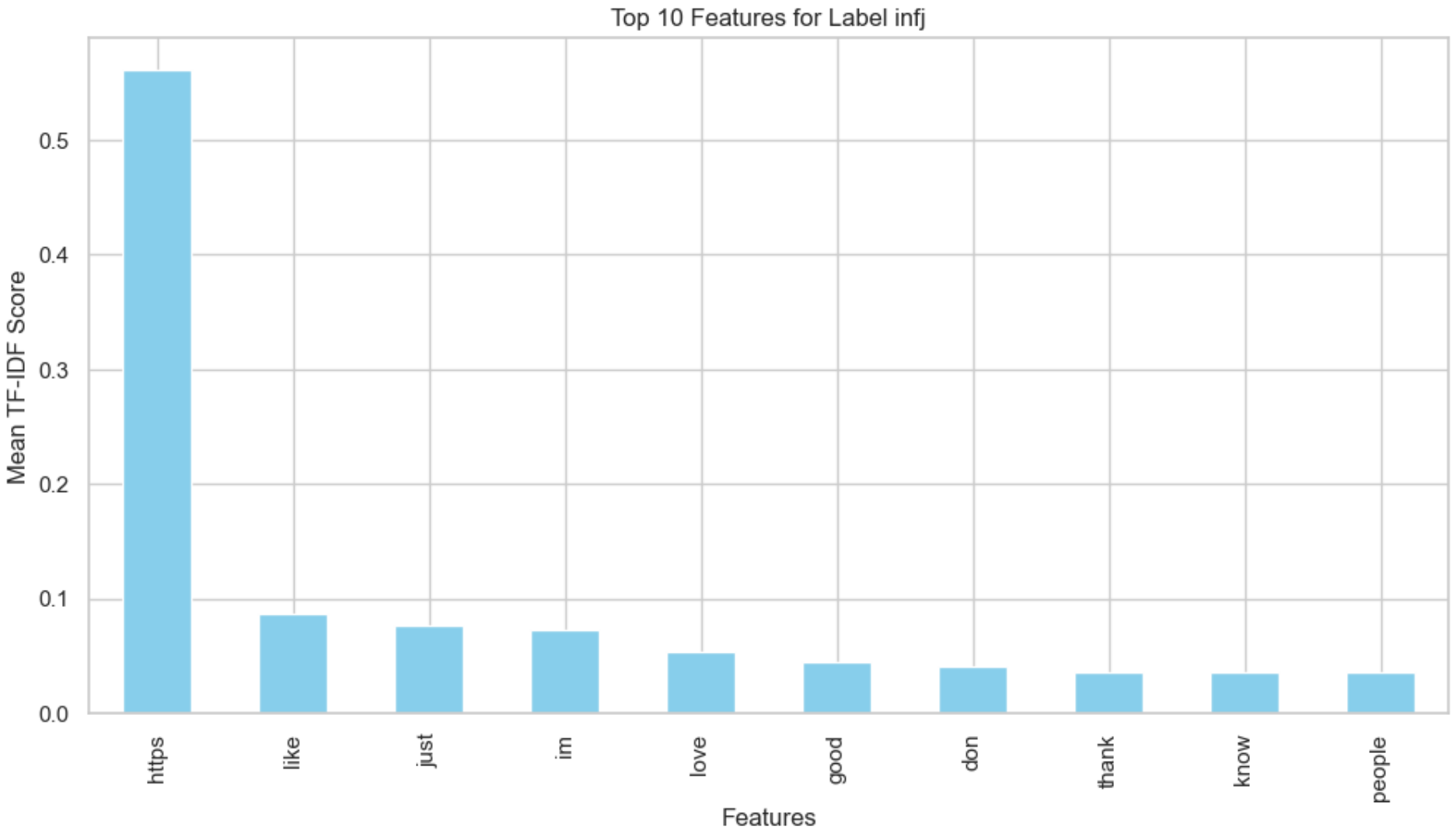

Top 10 Features for Label entp
Mean TF-IDF Score
0.0
0.1
0.2
0.3
0.4
0.5
https
like
just
im
love
good
don
thank
It
people
Features

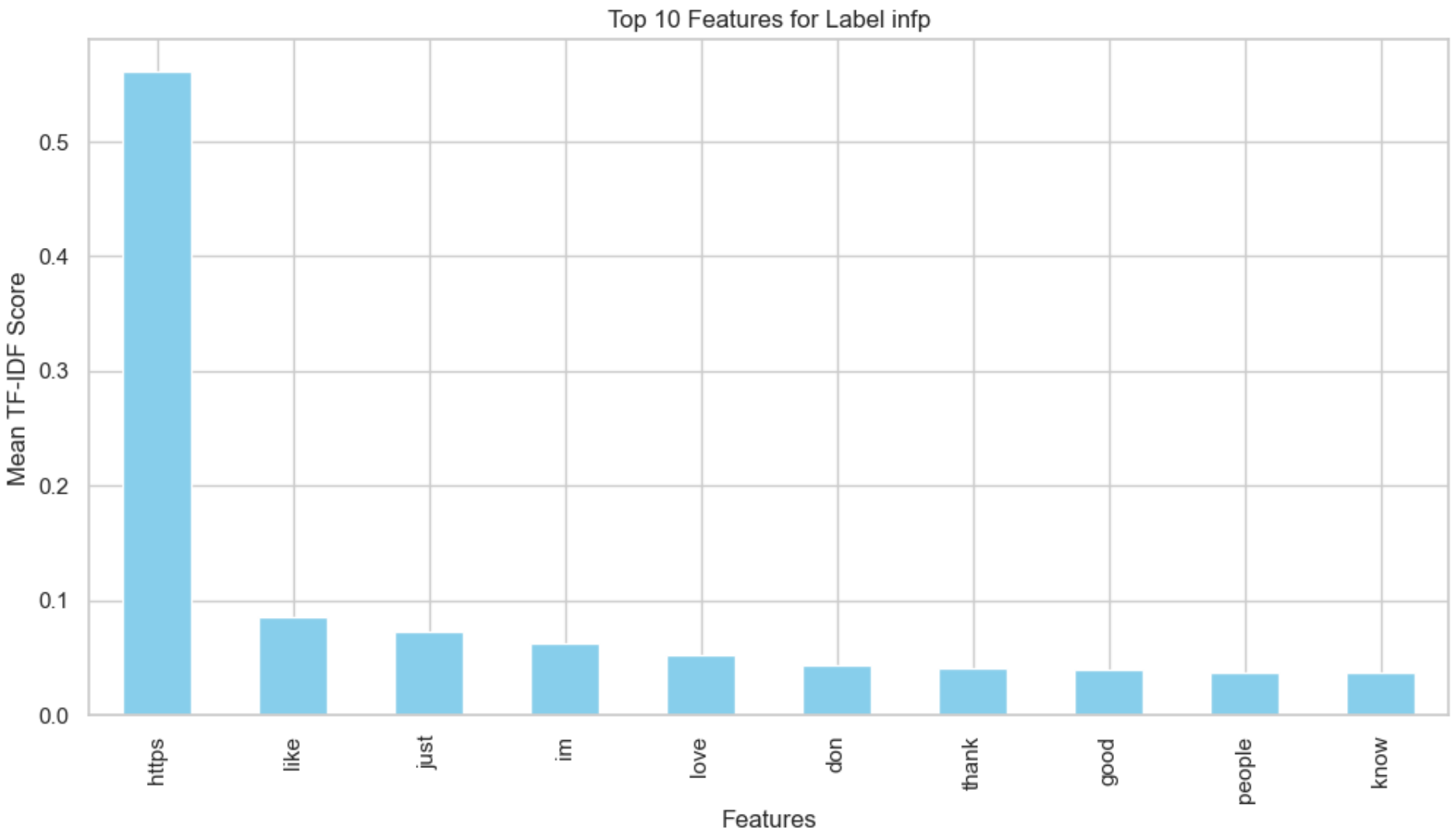

Top 10 Features for Label esfj
Mean TF-IDF Score
0.0
0.1
0.2
0.3
0.4
0.5
https
like
just
im
good
love
thank
don
people
It
Features

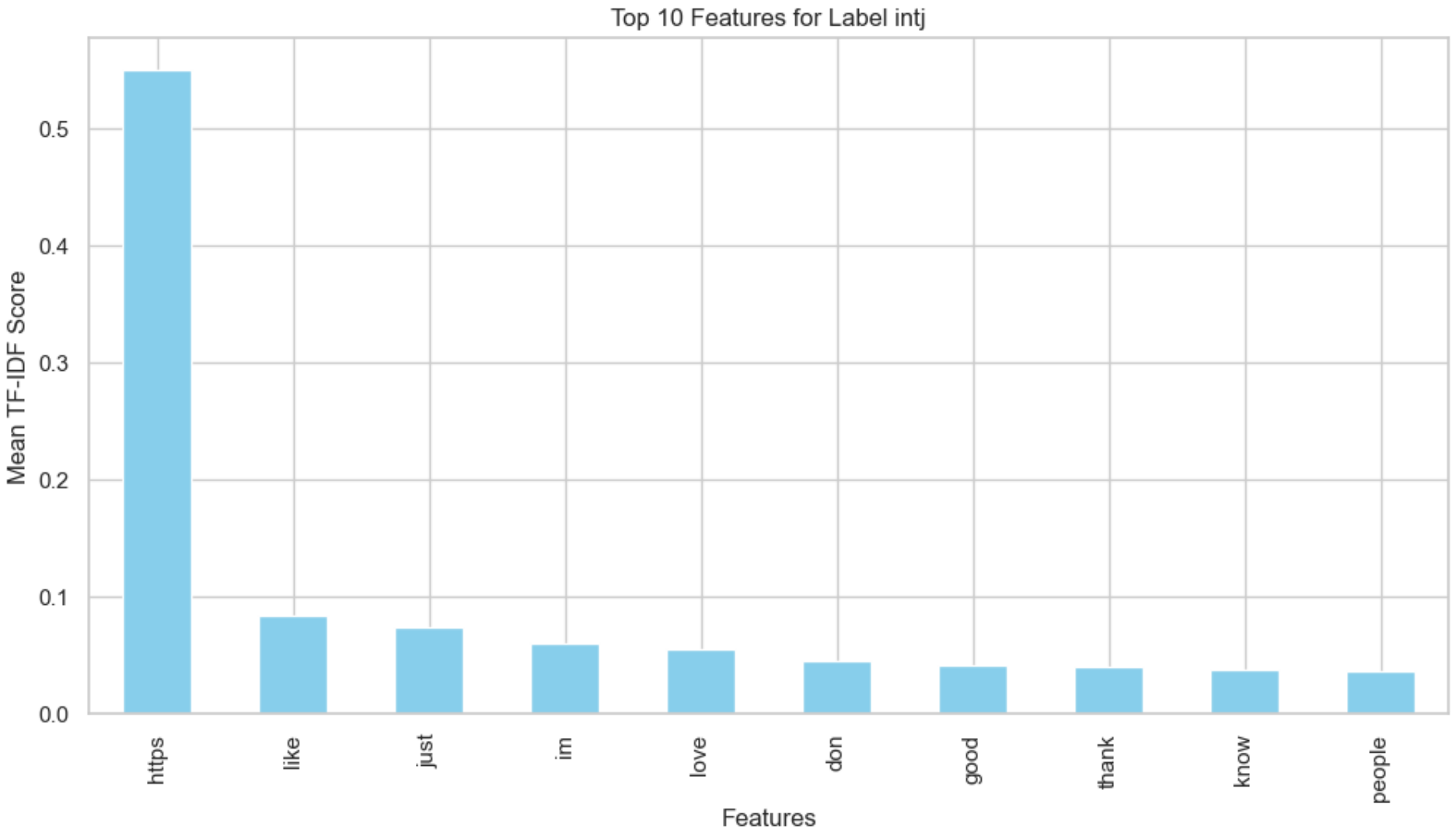

Top 10 Features for Label esfp
Mean TF-IDF Score
0.0
0.1
0.2
0.3
0.4
0.5
https
like
just
im
love
thank
good
don
bc
know
Features

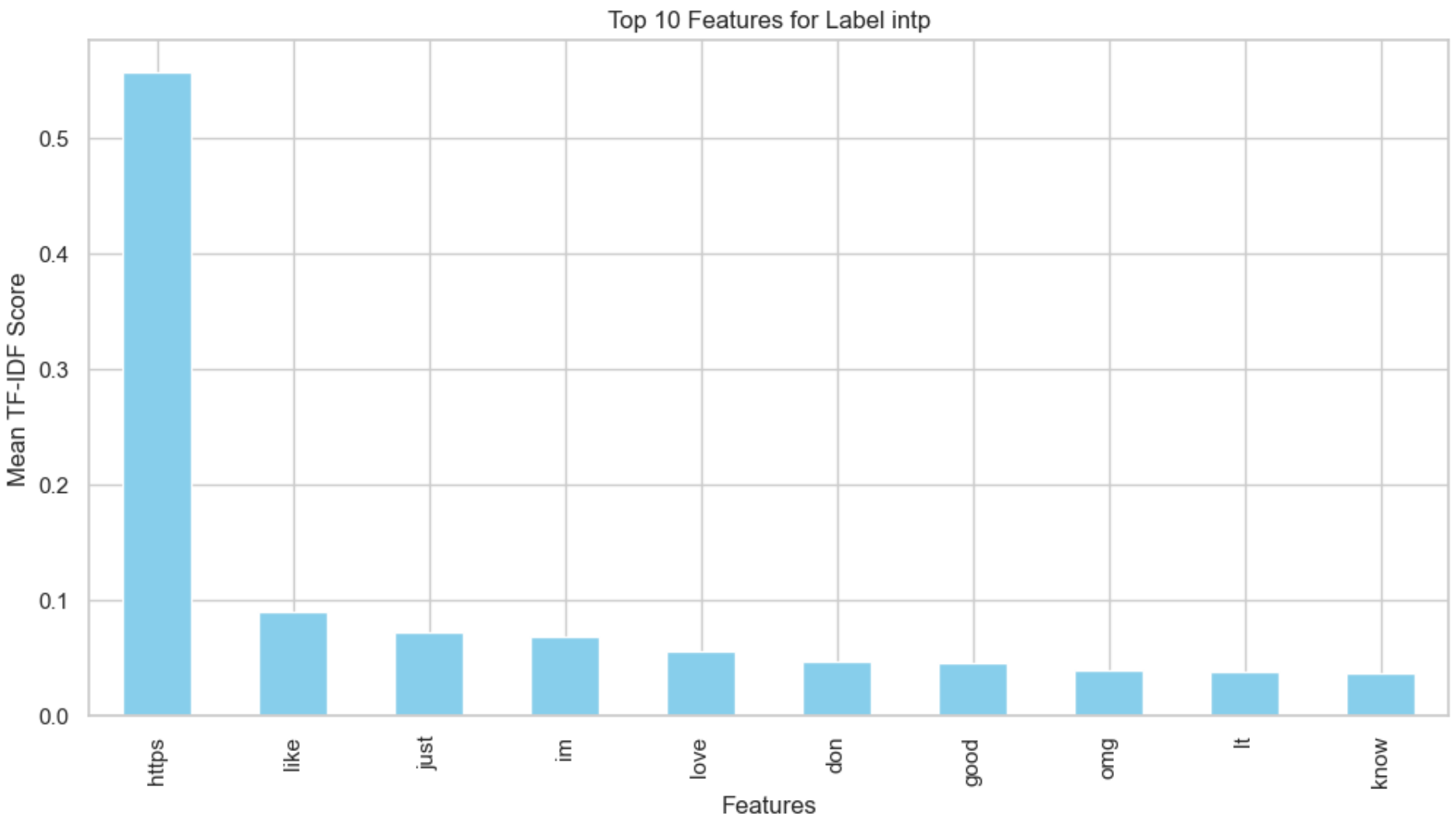

Top 10 Features for Label estj
Mean TF-IDF Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
https
like
just
im
love
don
good
ur
people
oh
Features

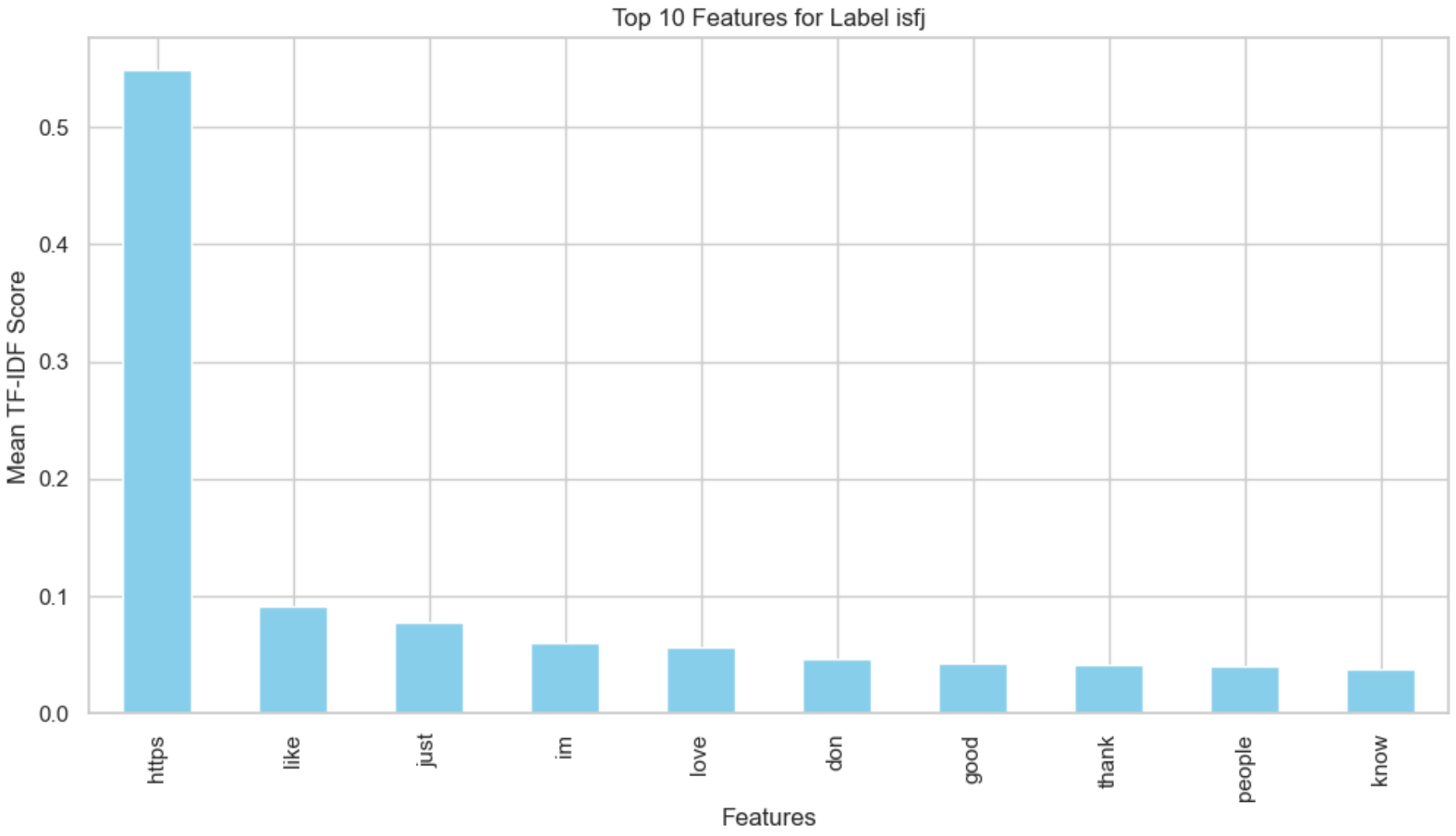

Top 10 Features for Label estp
Mean TF-IDF Score
0.0
0.1
0.2
0.3
0.4
0.5
https
like
just
im
love
don
good
think
know
thank
Features

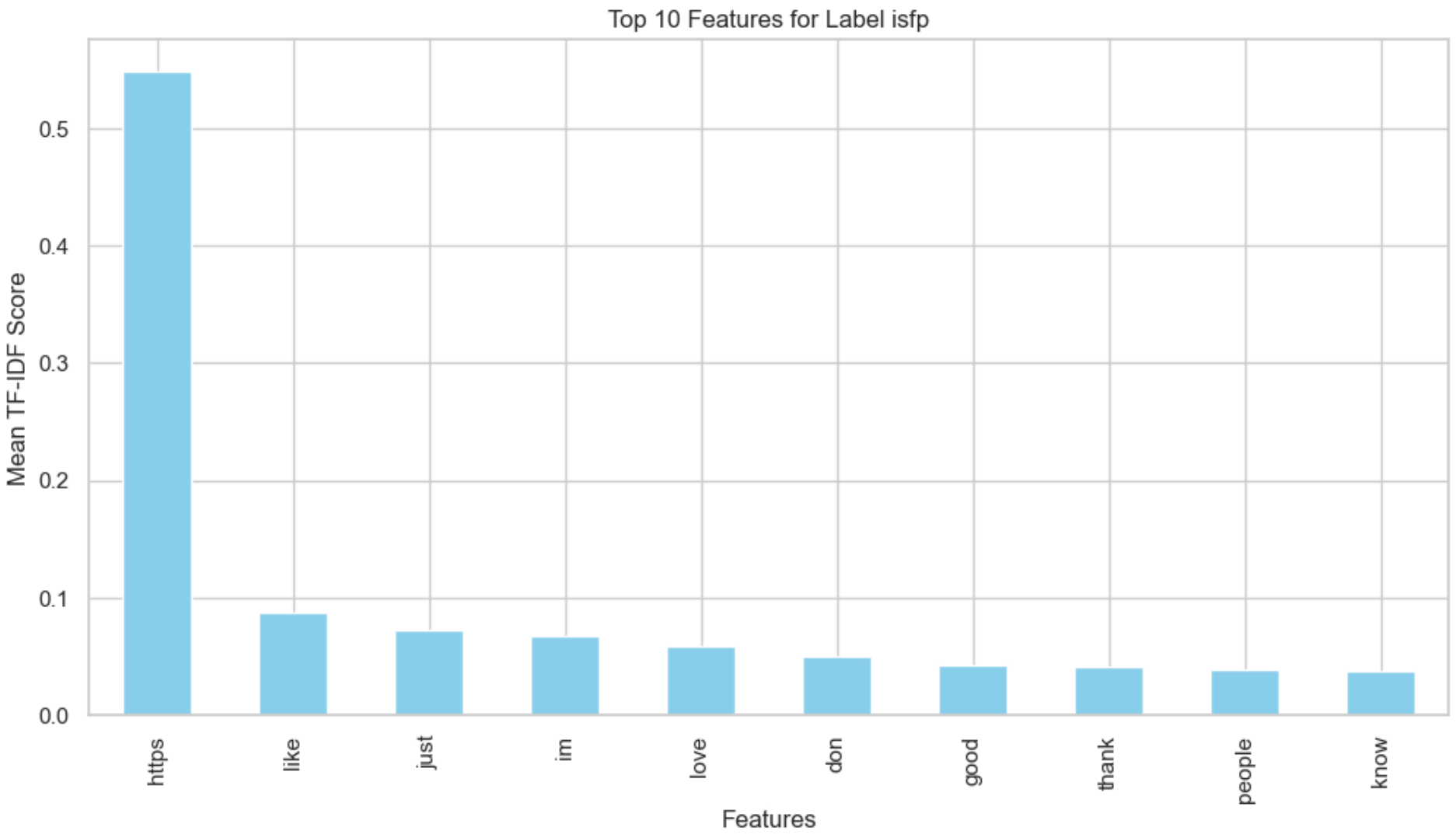

Top 10 Features for Label infj
Mean TF-IDF Score
0.0
0.1
0.2
0.3
0.4
0.5
https
like
just
im
love
good
don
thank
know
people
Features

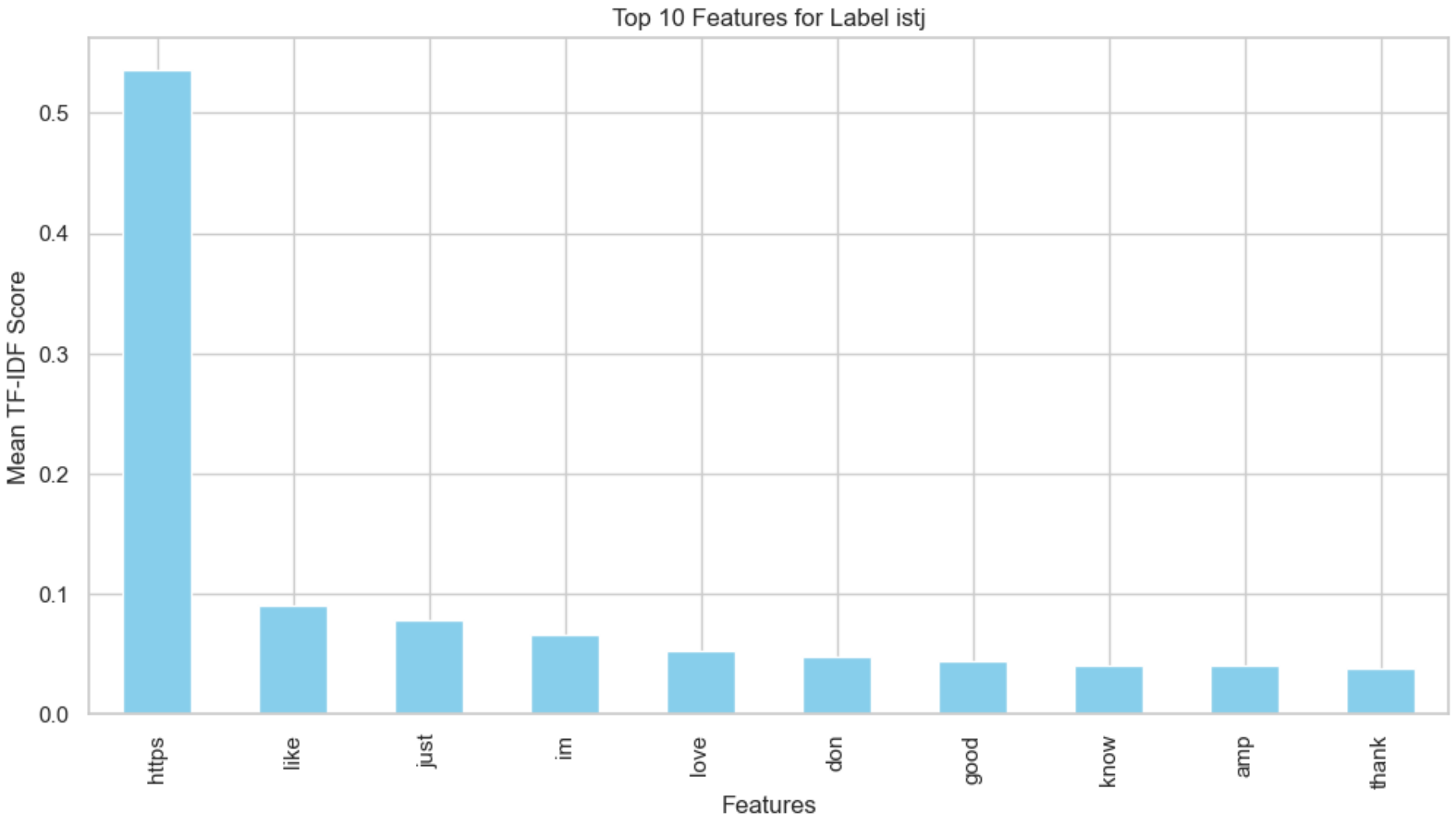

Top 10 Features for Label infp
Mean TF-IDF Score
0.0
0.1
0.2
0.3
0.4
0.5
https
like
just
im
love
don
thank
good
people
know
Features

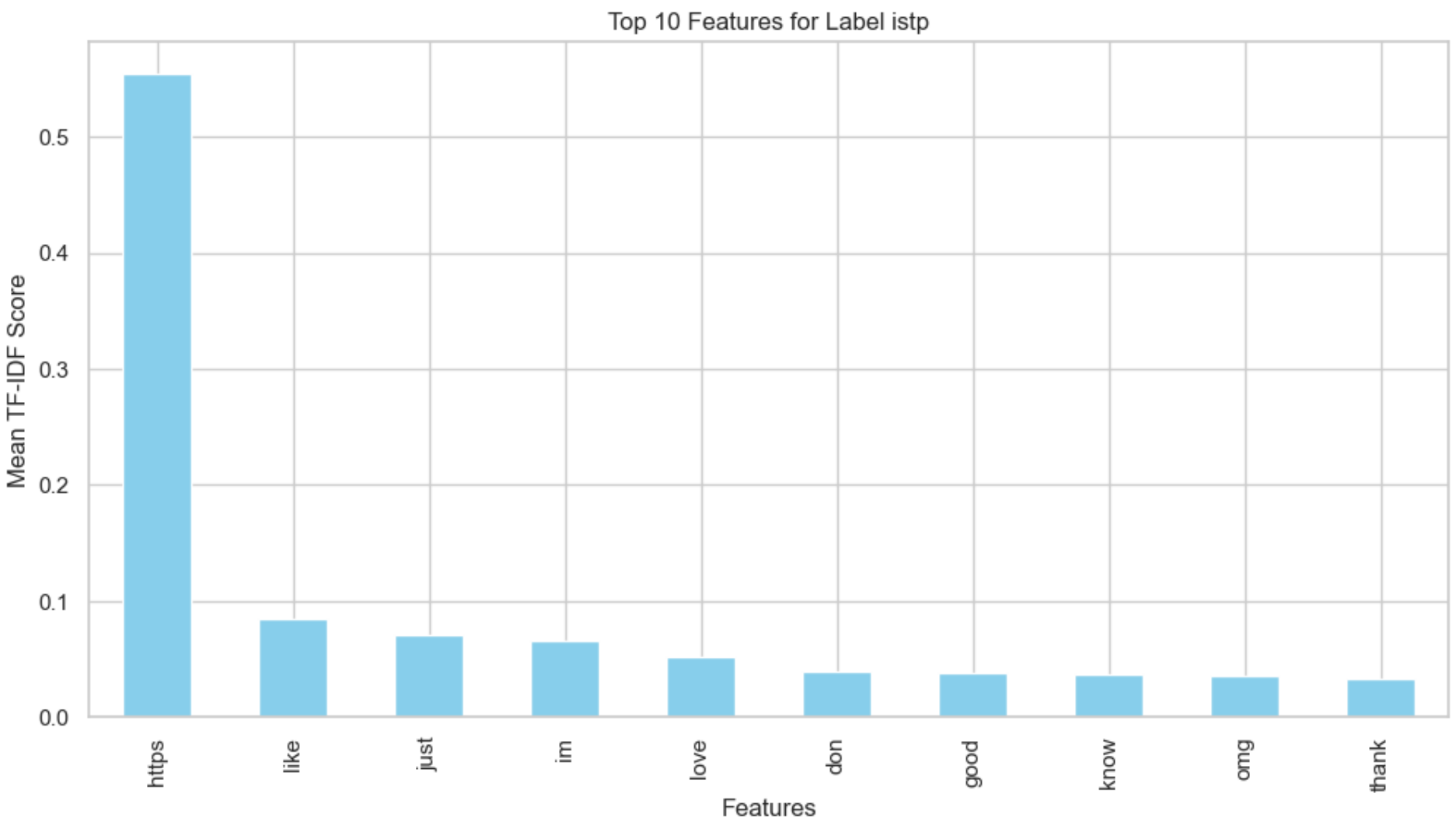

Top 10 Features for Label intj
Mean TF-IDF Score
0.0
0.1
0.2
0.3
0.4
0.5
https
like
just
im
love
don
good
thank
know
people
Features


Top 10 Features for Label intp
Mean TF-IDF Score
0.0
0.1
0.2
0.3
0.4
0.5
https
like
just
im
love
don
good
omg
lt
know
Features


Top 10 Features for Label isfj
Mean TF-IDF Score
0.0
0.1
0.2
0.3
0.4
0.5
https
like
just
im
love
don
good
thank
people
know
Features


Top 10 Features for Label isfp
Mean TF-IDF Score
0.0
0.1
0.2
0.3
0.4
0.5
https
like
just
im
love
don
good
thank
people
know
Features


Top 10 Features for Label istj
Mean TF-IDF Score
0.0
0.1
0.2
0.3
0.4
0.5
https
like
just
im
love
don
good
know
amp
thank
Features


Top 10 Features for Label istp
Mean TF-IDF Score
0.0
0.1
0.2
0.3
0.4
0.5
https
like
just
im
love
don
good
know
omg
thank
Features

- import the matplotlib.pyplot module and give it an alias, plt.
- Define a function called plot_top_features that takes two arguments, row and label. This function will create a bar chart of the top 10 features for a given label.
- Inside the function, create a new figure with a width of 12 inches and a height of 6 inches.
- Call the nlargest() method on the row object to get the 10 largest values. Plot these values as a bar chart with a sky blue color.
- Set the title of the chart to "Top 10 Features for Label {label}".
- Set the x-axis label to "Features" and the y-axis label to "Mean TF-IDF Score".
- Display the chart by calling the plt.show() function.
- Outside the function, iterate through each row of the mean_tfidf_by_label DataFrame using a for loop. The loop variables are idx and row.
- Within the loop, convert the index idx to an integer and store it in the variable label_idx. Use the label encoder le to transform the integer index back to its original label form, and store the result in the variable label.
- Call the plot_top_features function and pass the row and label variables as arguments. This will create a bar chart of the top 10 features for each label in the dataset.

In [ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer, ENGLISH_STOP_WORDS
from sklearn.preprocessing import LabelEncoder

# Add common words to the stop_words list
custom_stop_words = ['https', 'like', 'just', 'im', 'love', 'don', 'good', 'thank', 'people', 'know', 'think']

# Combine the default 'english' stop words and the custom stop words
stop_words = list(ENGLISH_STOP_WORDS) + custom_stop_words

# Create the TfidfVectorizer with the updated stop_words list
vectorizer = TfidfVectorizer(stop_words=stop_words)
le = LabelEncoder()

# Fit and transform the data
X_train = vectorizer.fit_transform(train_data['text'])
y_train = le.fit_transform(train_data['label'])

# Transform the test data
X_test = vectorizer.transform(test_data['text'])

# Calculate the mean TF-IDF score for each label
mean_tfidf_by_label = pd.DataFrame(X_train.toarray(), columns=vectorizer.get_feature_names_out()).groupby(y_trai

# Display the top 10 features for each label
for idx, row in mean_tfidf_by_label.iterrows():
    label = le.inverse_transform([idx])[0]
    print(f"Top 10 features for label {label}:")
    print(row.nlargest(10))
    print("\n")
```

```
Top 10 features for label 0:
ve        0.013251
really    0.013129
lt        0.012899
day       0.012547
amp       0.012473
omg       0.012216
time      0.012174
today     0.011517
got       0.010816
yes       0.010775
Name: 0, dtype: float64


Top 10 features for label 1:
time      0.013075
lt        0.012994
omg       0.012984
ve        0.011553
oh        0.011525
really    0.011395
ur        0.011345
want      0.011191
got       0.011111
day       0.010848
Name: 1, dtype: float64


Top 10 features for label 2:
```

```
gt        0.014365
time      0.012998
day       0.012349
want      0.011662
ve        0.011180
amp       0.011010
need      0.010926
really    0.010796
got       0.010755
yes       0.010737
Name: 2, dtype: float64


Top 10 features for label 3:
ur         0.012735
lt         0.012421
fuck       0.011824
dont       0.011579
fucking    0.011410
want       0.011403
omg        0.011265
time       0.011160
need       0.011133
entp       0.011077
Name: 3, dtype: float64


Top 10 features for label 4:
aye          0.020371
gonna        0.014227
omg          0.013121
oh           0.013086
literally    0.012519
want         0.012086
time         0.012014
day          0.011923
amp          0.011921
really       0.011861
Name: 4, dtype: float64


Top 10 features for label 5:
omg          0.017702
ur           0.017131
lt           0.015264
bc           0.013823
oh           0.013409
happy        0.011778
lol          0.011706
literally    0.011640
want         0.011588
day          0.011269
Name: 5, dtype: float64


Top 10 features for label 6:
amp              0.018520
epistolarys      0.017798
maimai04753      0.017270
god              0.016935
time             0.014610
artistwissues    0.013501
loveisland       0.013087
na               0.012770
gt               0.012704
day              0.011986
Name: 6, dtype: float64


Top 10 features for label 7:
fiirtykook       0.017963
tohrhu           0.015386
estp             0.015251
gt               0.014708
pipawoof         0.014119
ur               0.013964
jiyong           0.013496
pepperharrows    0.012168
fucking          0.012127
```

```
catboyspurr      0.011793
Name: 7, dtype: float64


Top 10 features for label 8:
amp       0.014795
time      0.013676
really    0.013362
ve        0.012446
day       0.012426
na        0.011809
omg       0.011649
lol       0.011429
happy     0.011181
oh        0.010923
Name: 8, dtype: float64


Top 10 features for label 9:
lt        0.013995
oh        0.013338
time      0.012641
really    0.012564
omg       0.012272
want      0.011851
amp       0.010994
need      0.010768
gonna     0.010693
day       0.010540
Name: 9, dtype: float64


Top 10 features for label 10:
time      0.013169
ve        0.012966
amp       0.012269
na        0.012168
really    0.011849
want      0.011028
lol       0.011010
day       0.010995
yes       0.010962
need      0.010224
Name: 10, dtype: float64


Top 10 features for label 11:
oh         0.011992
really     0.011666
omg        0.011507
time       0.011418
gonna      0.011094
want       0.011063
gt         0.010921
fucking    0.010620
intp       0.010517
need       0.010419
Name: 11, dtype: float64


Top 10 features for label 12:
lt        0.018761
na        0.015458
oh        0.013597
omg       0.013482
happy     0.012154
day       0.011536
time      0.011422
really    0.011328
sa        0.011163
isfj      0.010949
Name: 12, dtype: float64


Top 10 features for label 13:
lt        0.015585
omg       0.015411
oh        0.013156
gonna     0.012679
```

```
really    0.012193
ur        0.011909
day       0.011440
want      0.011233
time      0.011172
need      0.010883
Name: 13, dtype: float64


Top 10 features for label 14:
enhypen            0.014485
enhypen_members    0.014132
time               0.012593
oh                 0.012579
omg                0.012503
bts_twt            0.012294
day                0.012114
really             0.011579
gonna              0.011174
need               0.010930
Name: 14, dtype: float64


Top 10 features for label 15:
lt      0.014560
istp    0.013288
na      0.013286
omg     0.012752
got     0.012157
want    0.011704
gt      0.011672
ur      0.011167
oh      0.011006
time    0.010908
Name: 15, dtype: float64
```

- Import the necessary libraries:

nltk: Natural Language Toolkit, a library for working with human language data (text). pandas: A library for data manipulation and analysis, especially for working with data in tables.

- Download the required resources for nltk:

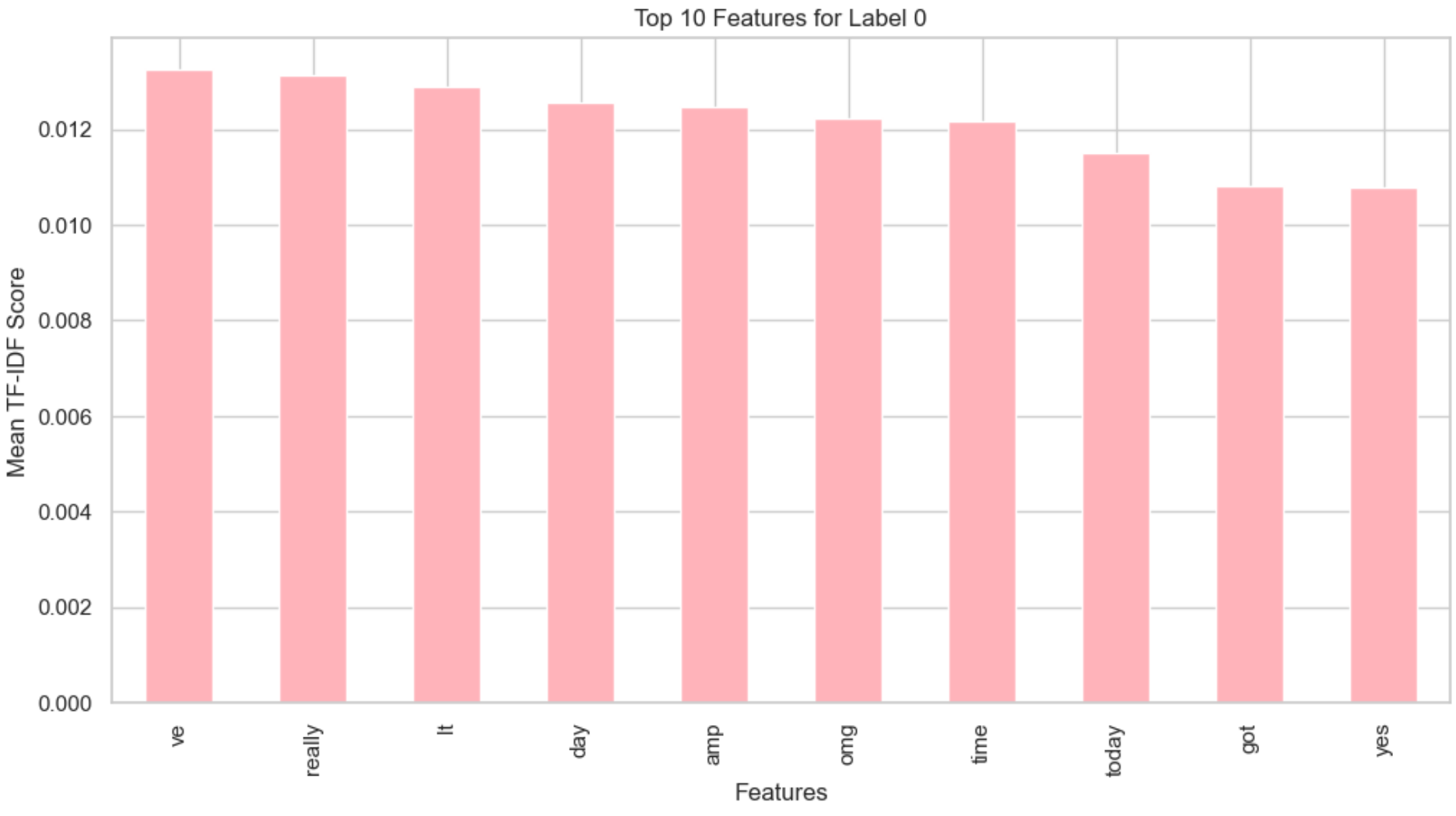
punkt: A tokenizer that divides a text into a list of sentences, by using an unsupervised algorithm to build a model for abbreviation words, collocations, and words that start sentences. vader_lexicon: A lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media.

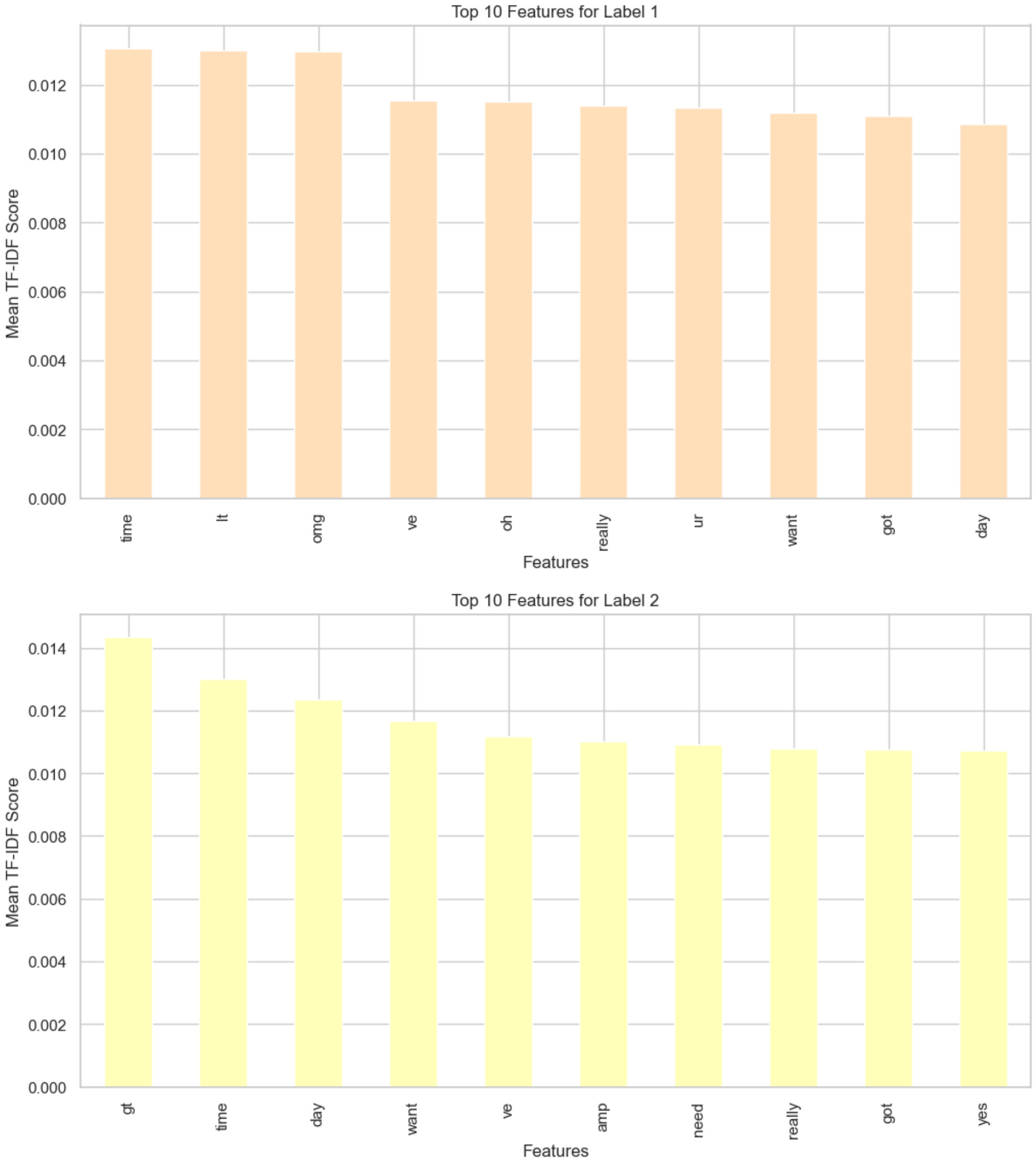
- Import SentimentIntensityAnalyzer from nltk.sentiment. This is a sentiment analysis tool that uses the VADER (Valence Aware Dictionary and sEntiment Reasoner) sentiment lexicon.
- Define the count_negative_words function that takes a text and an optional threshold value as input (default value is -0.2):

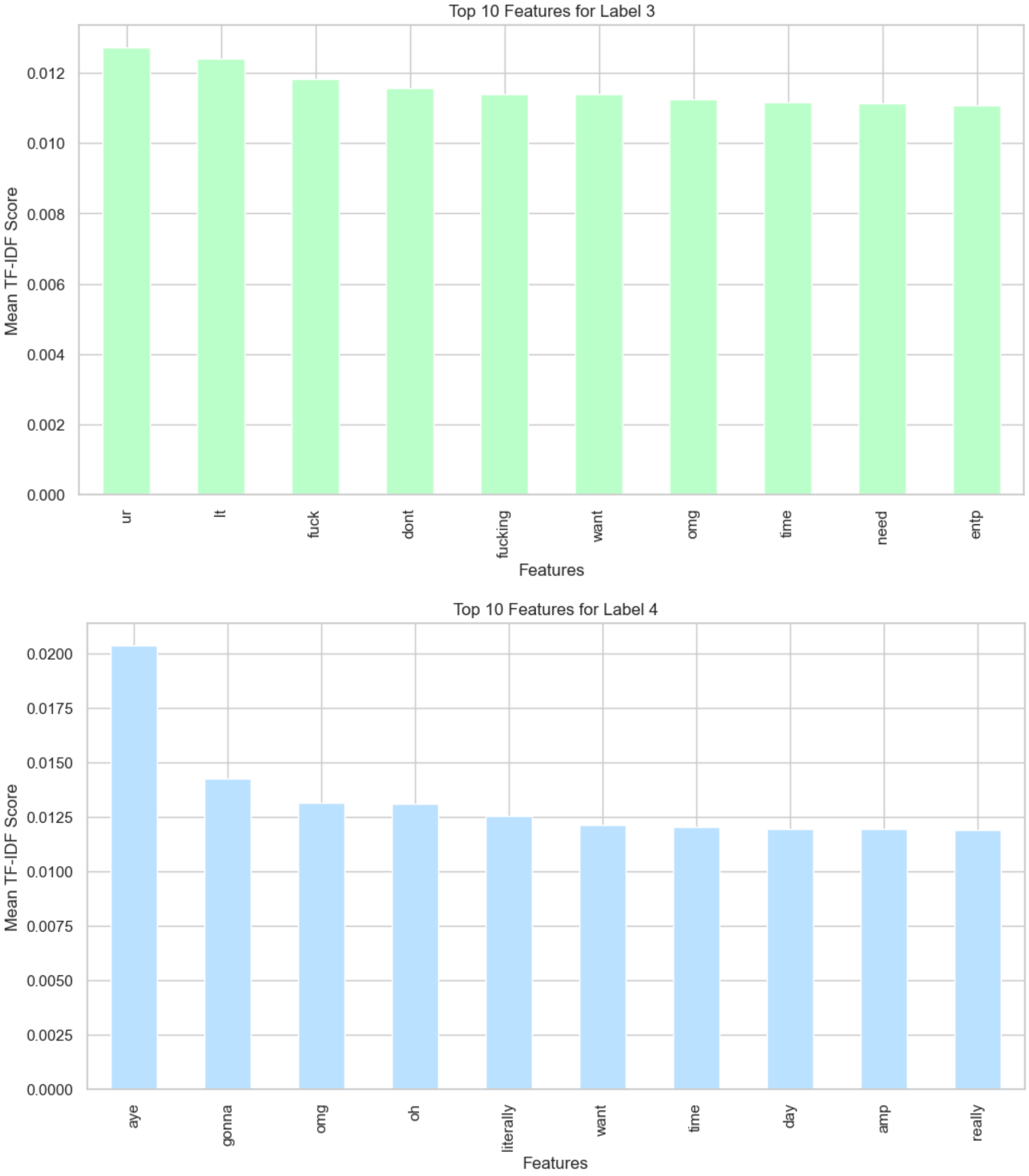
Create an instance of SentimentIntensityAnalyzer. Tokenize the input text into a list of words using nltk.word_tokenize(). Initialize an empty list to store negative words. Iterate through each token (word) in the list of tokens. Calculate the sentiment score of the token using the SentimentIntensityAnalyzer's polarity_scores method. If the sentiment score is less than or equal to the threshold value (i.e., the word is considered negative), append the token to the negative_words list. Return the length of the negative_words list and the list itself.

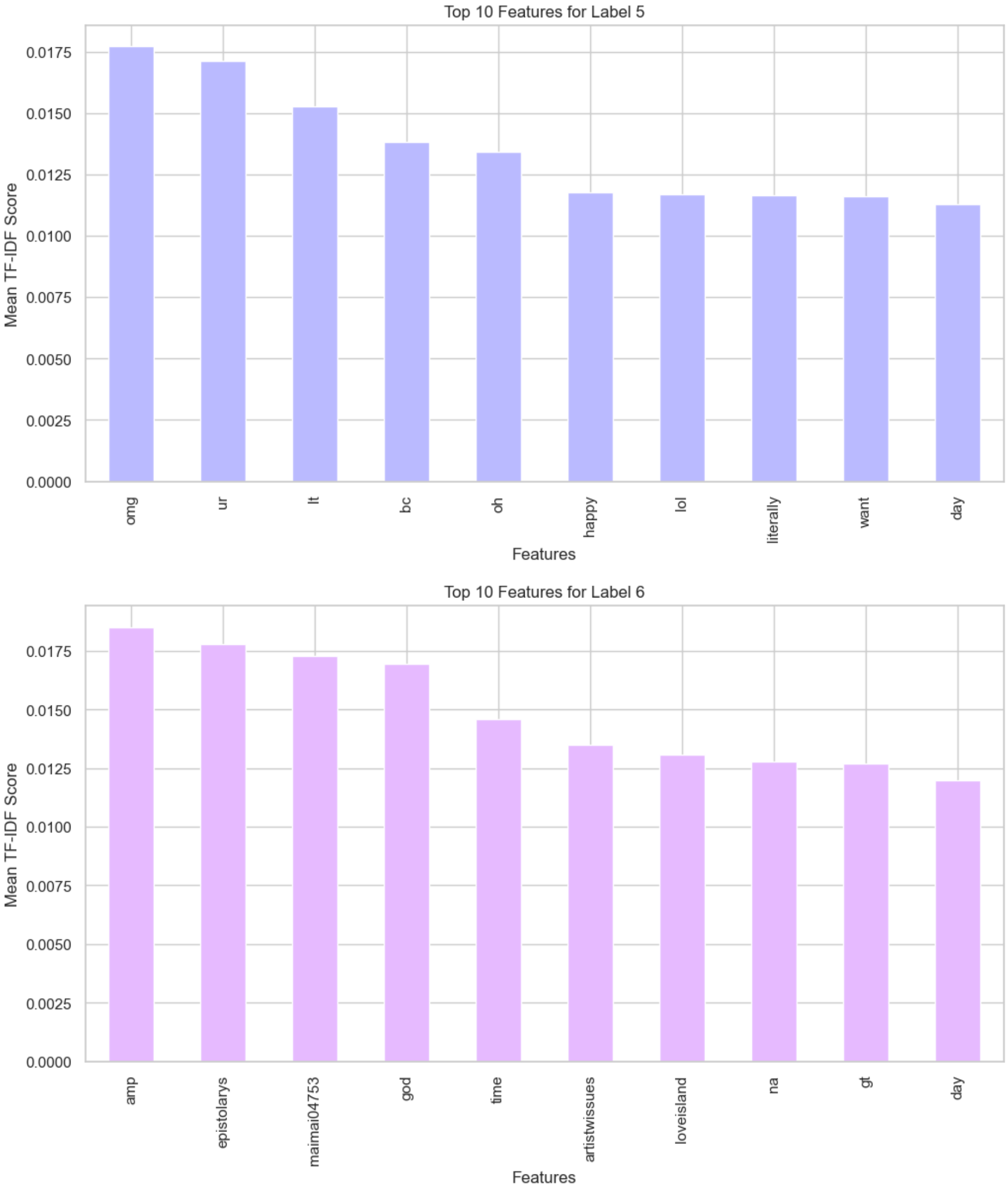
- Apply the count_negative_words function to the "text" column of the dataset, and store the results in two new columns, "negative_word_count" and "negative_words".
- Preview the dataset with the new columns by printing the first few rows using the head() method.

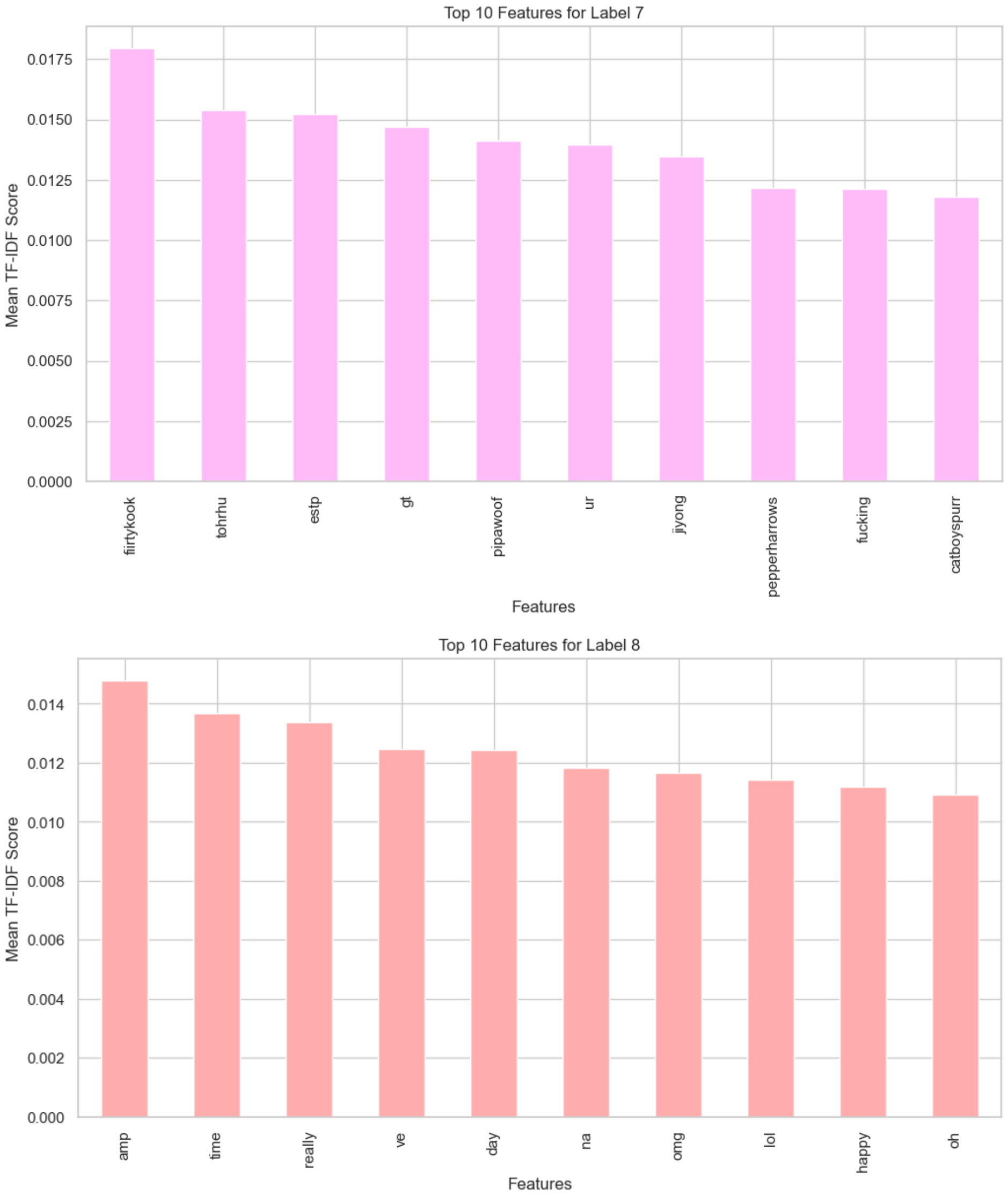
In [ ]:
```
import matplotlib.pyplot as plt

# Define a list of pastel colors
pastel_colors = [
    '#FFB3BA', '#FFDFBA', '#FFFFBA', '#BAFFC9', '#BAE1FF', '#BABAFF',
    '#E6BAFF', '#FFBAF8', '#FFACAC', '#FFDAC1', '#E3FFC1', '#C1FFE3',
    '#C1DFFF', '#D5C1FF', '#FFC1E8', '#FFC1C1'
]

def plot_top_features(row, label, color):
    plt.figure(figsize=(12, 6))
    row.nlargest(10).plot(kind='bar', color=color)
    plt.title(f"Top 10 Features for Label {label}")
    plt.xlabel("Features")
    plt.ylabel("Mean TF-IDF Score")
    plt.show()

# Display the top 10 features for each label and plot them
for idx, (row, color) in enumerate(zip(mean_tfidf_by_label.iterrows(), pastel_colors)):
    label_idx = int(row[0])
    label = le.inverse_transform([label_idx])[0]
    plot_top_features(row[1], label, color)
```

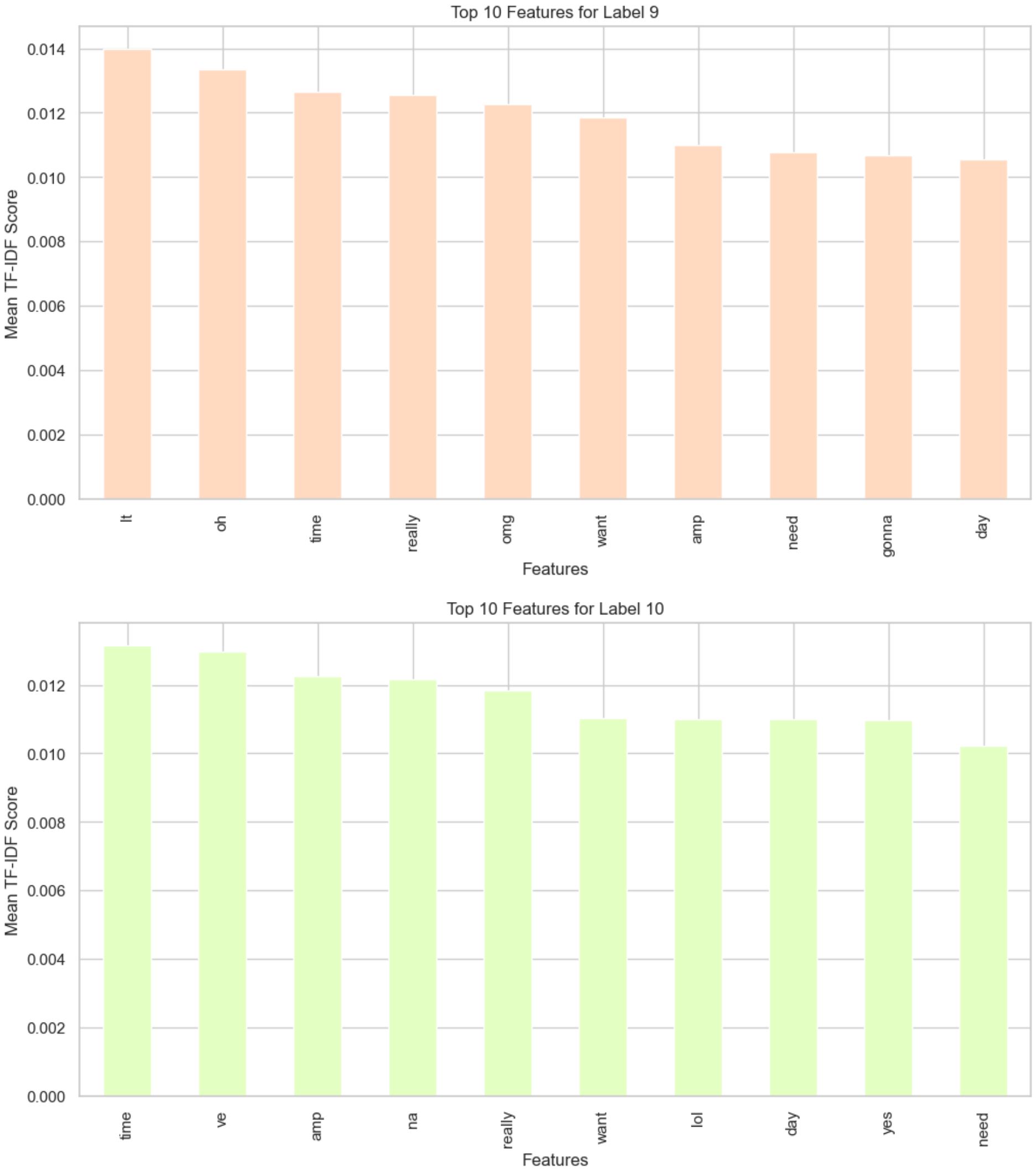


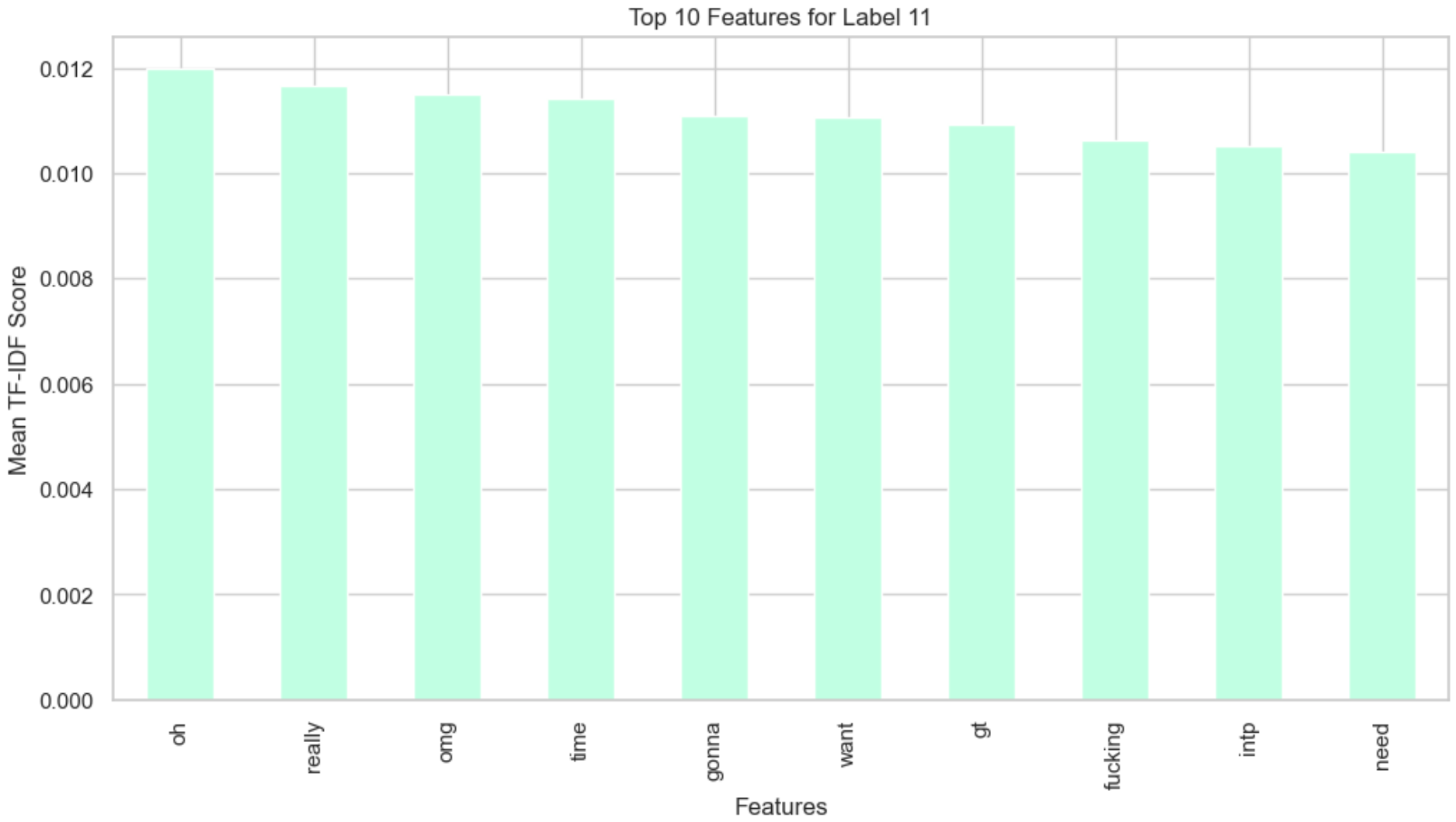

Top 10 Features for Label 1
Mean TF-IDF Score
0.000
0.002
0.004
0.006
0.008
0.010
0.012
time
lt
omg
ve
oh
really
ur
want
got
day
Features
Top 10 Features for Label 2
Mean TF-IDF Score
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
gt
time
day
want
ve
amp
need
really
got
yes
Features

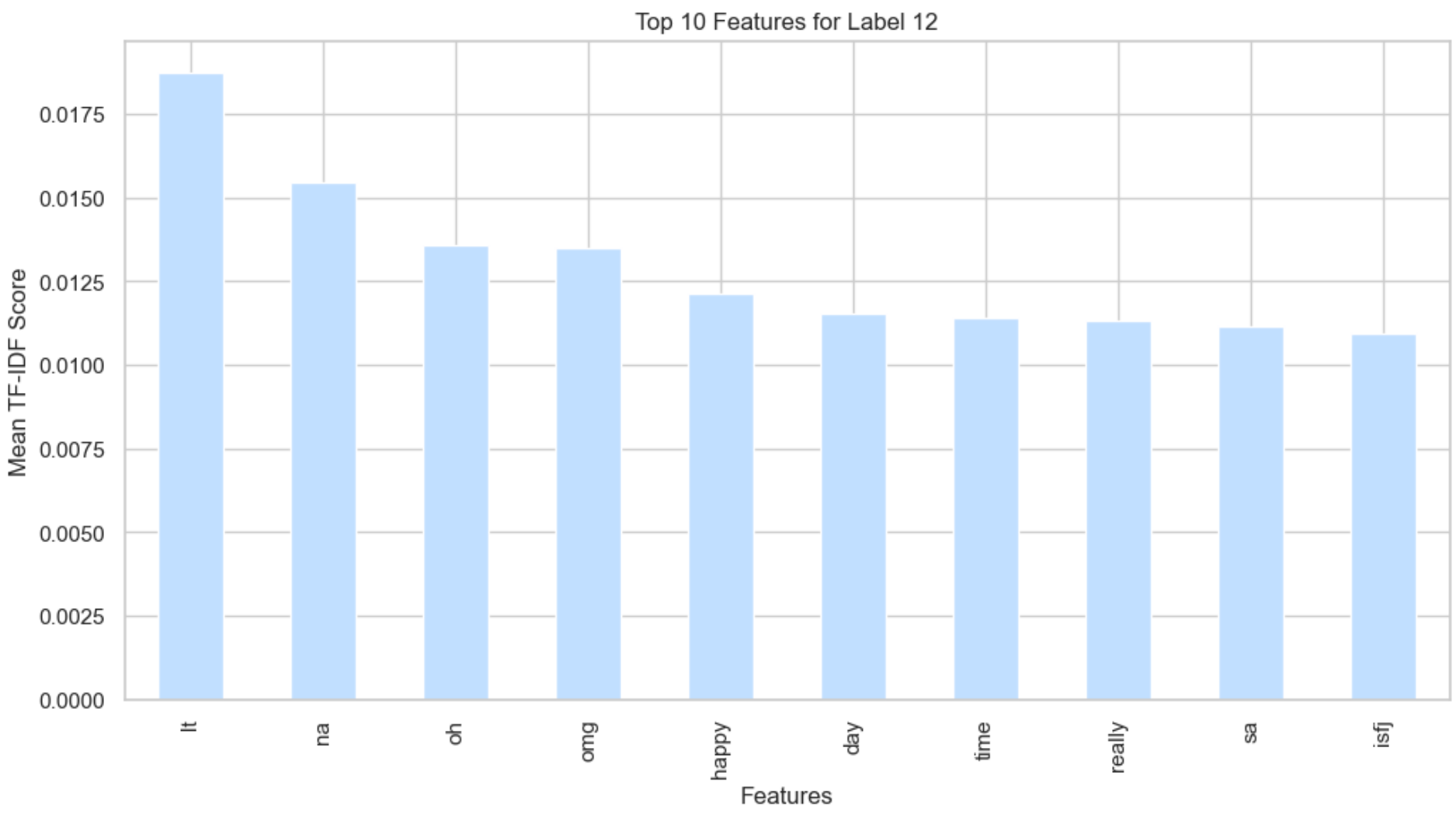

Top 10 Features for Label 3
Mean TF-IDF Score
0.000
0.002
0.004
0.006
0.008
0.010
0.012
ur
lt
fuck
dont
fucking
want
omg
time
need
entp
Features
Top 10 Features for Label 4
Mean TF-IDF Score
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
0.0200
aye
gonna
omg
oh
literally
want
time
day
amp
really
Features

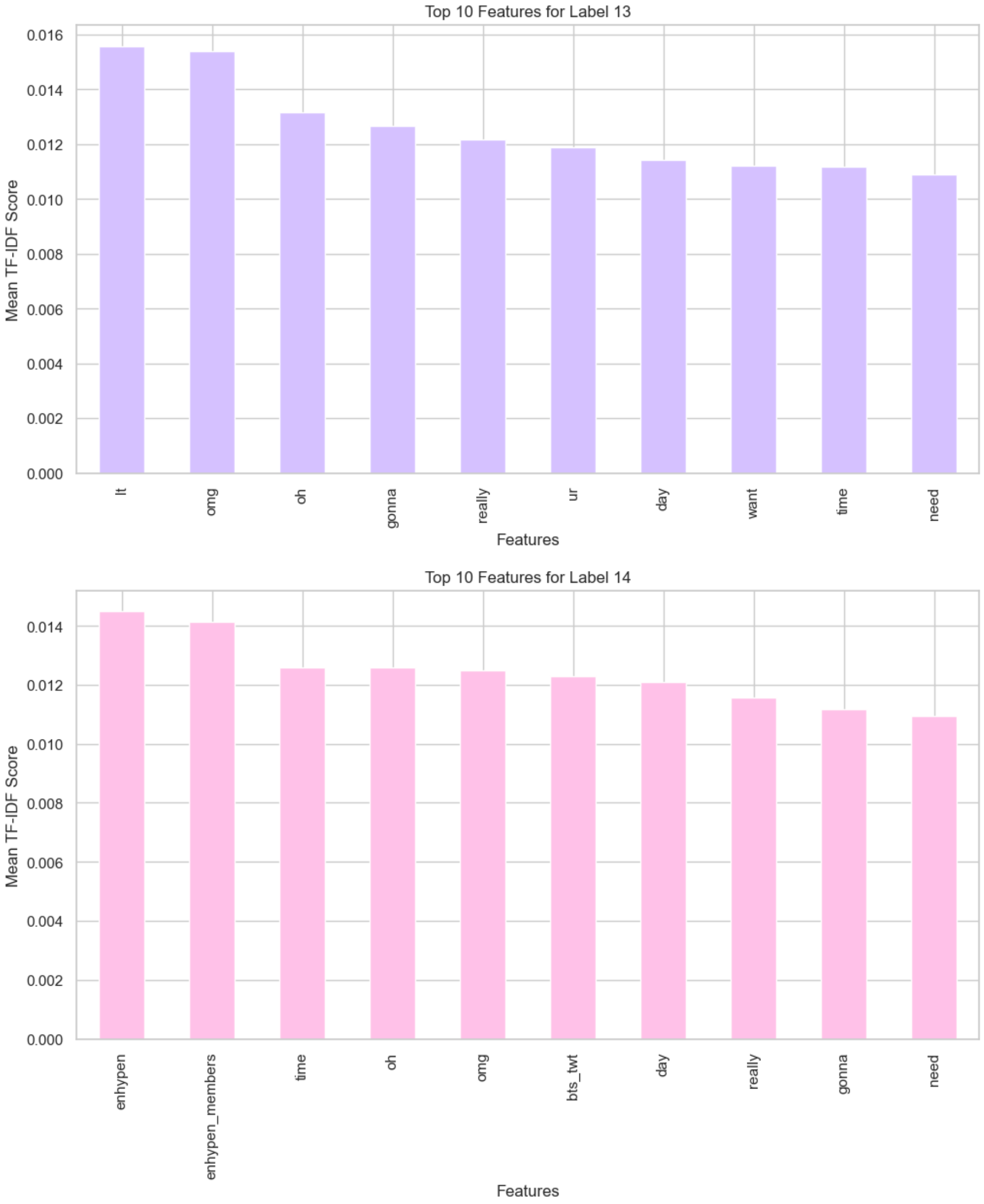

Top 10 Features for Label 5
Mean TF-IDF Score
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
omg
ur
lt
bc
oh
happy
lol
literally
want
day
Features
Top 10 Features for Label 6
Mean TF-IDF Score
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
amp
epistolarys
maimai04753
god
time
artistwissues
loveisland
na
gt
day
Features

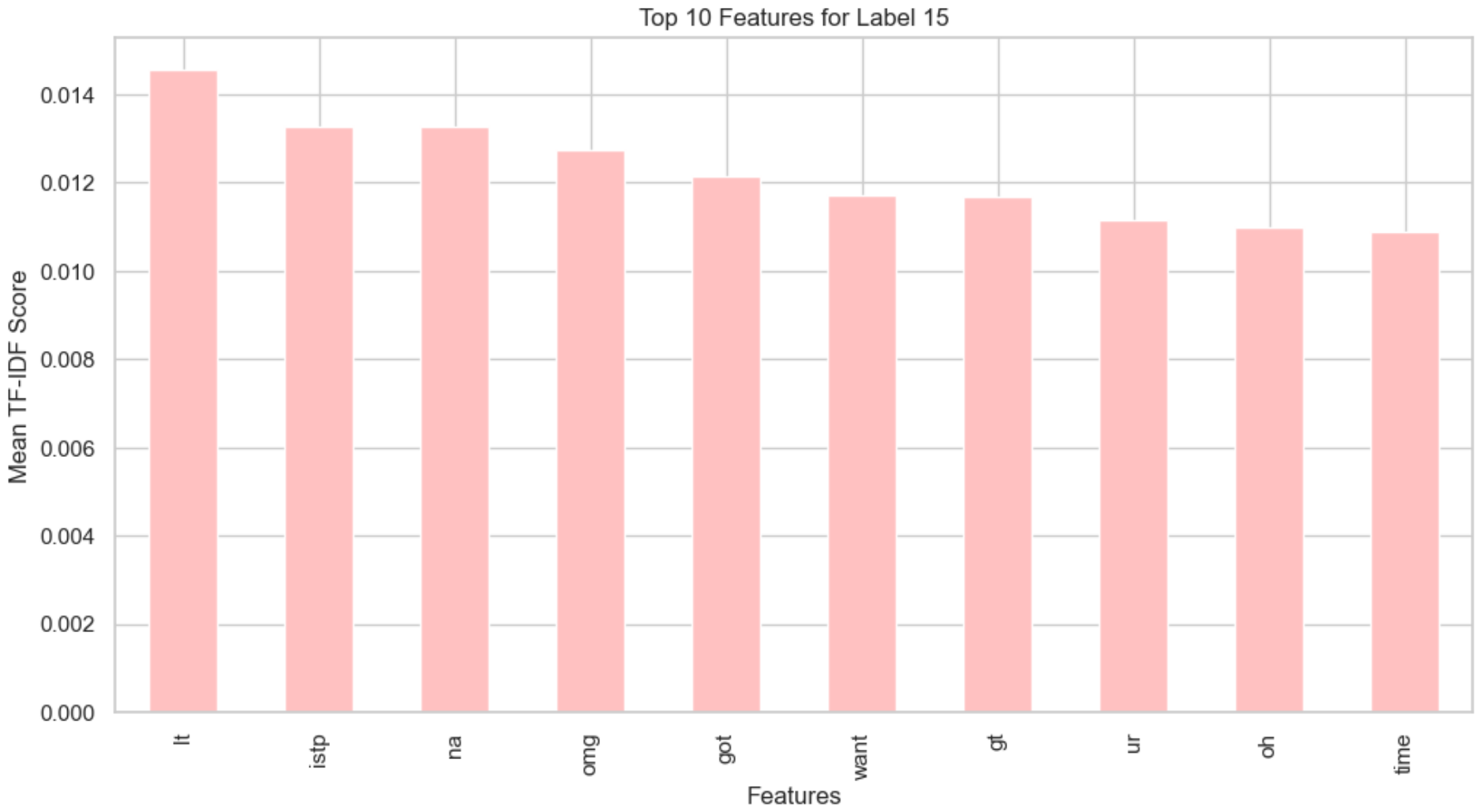

Top 10 Features for Label 7
Mean TF-IDF Score
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
fiirtykook
tohrhu
estp
gt
pipawoof
ur
jiyong
pepperharrows
fucking
catboyspurr
Features
Top 10 Features for Label 8
Mean TF-IDF Score
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
amp
time
really
ve
day
na
omg
lol
happy
oh
Features


Top 10 Features for Label 9
Mean TF-IDF Score
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
Features
Top 10 Features for Label 10
Mean TF-IDF Score
0.012
0.010
0.008
0.006
0.004
0.002
0.000
Features


Top 10 Features for Label 11
Mean TF-IDF Score
0.012
0.010
0.008
0.006
0.004
0.002
0.000
oh
really
omg
time
gonna
want
gt
fucking
intp
need
Features


Top 10 Features for Label 12
Mean TF-IDF Score
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
lt
na
oh
omg
happy
day
time
really
sa
isfj
Features


Top 10 Features for Label 13
Mean TF-IDF Score
0.016
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
Features
Top 10 Features for Label 14
Mean TF-IDF Score
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
Features



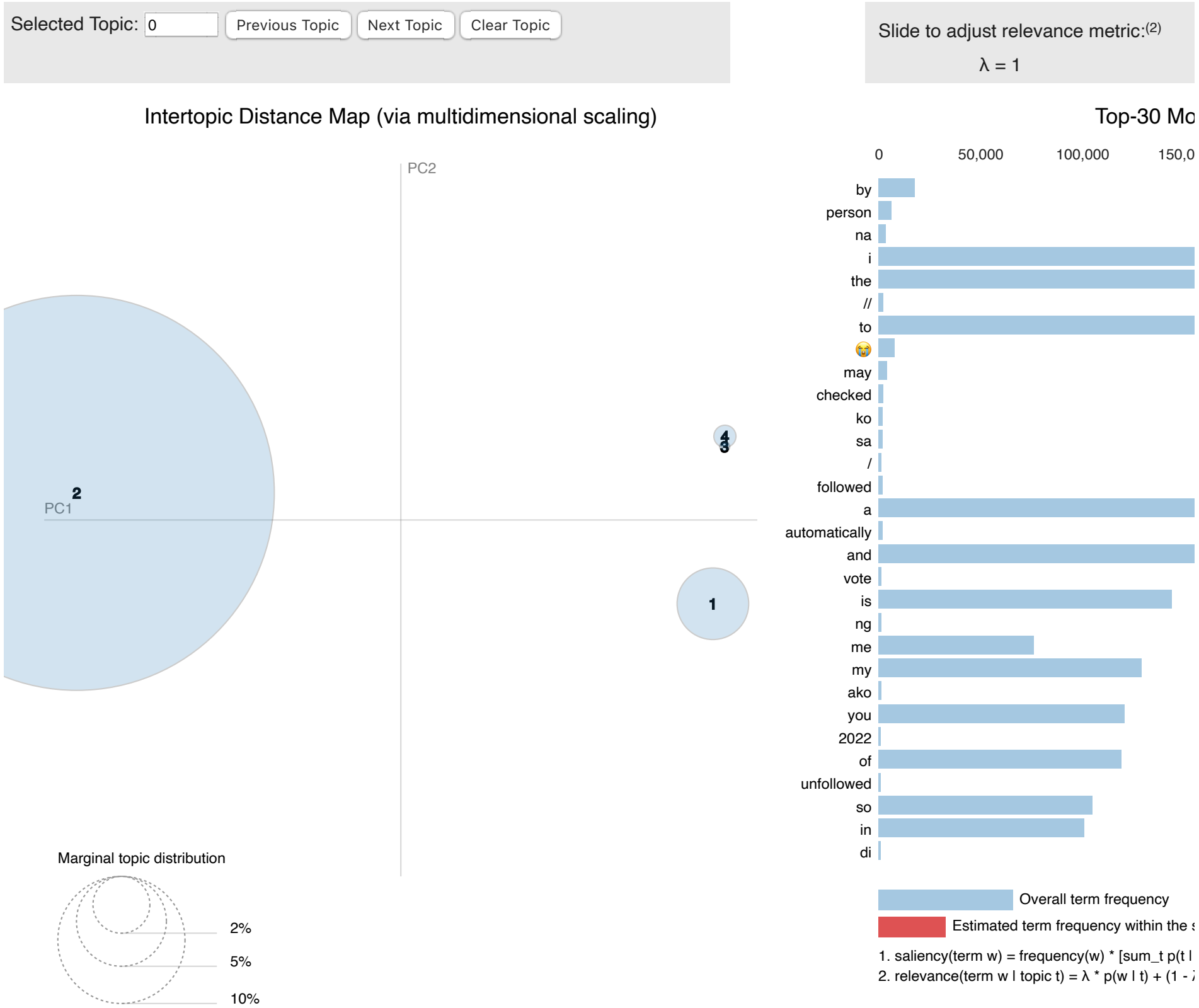
- Define a list of pastel colors in their hexadecimal format. These colors will be used for the bar charts.
- Define a function plot_top_features that takes three arguments: a row of data, the label (MBTI type), and a color. This function creates a bar chart of the top 10 features for a given label using the specified color.
- iterate through the mean TF-IDF scores by label (MBTI type) and the pastel colors. For each iteration, call the plot_top_features function to create a bar chart with the top 10 features for the current label and the corresponding color.

python

Latent Dirichlet Allocation (LDA) topic modeling to help visualize the words and their importance within the topics.

In [ ]:
```python
import gensim
from gensim import corpora
import pyLDAvis.gensim_models as gensimvis
import pyLDAvis

# Tokenize the text and remove stopwords
texts = [[word for word in document.lower().split() if word not in custom_stop_words] for document in data['text

# Create a dictionary and a corpus from the tokenized texts
dictionary = corpora.Dictionary(texts)
corpus = [dictionary.doc2bow(text) for text in texts]

# Train the LDA model
num_topics = 4
lda_model = gensim.models.ldamodel.LdaModel(corpus, num_topics=num_topics, id2word=dictionary, passes=15)

# Visualize the topics using pyLDAvis
lda_display = gensimvis.prepare(lda_model, corpus, dictionary, sort_topics=False)
pyLDAvis.display(lda_display)
```

Out[ ]:


Selected Topic: 0
Previous Topic
Next Topic
Clear Topic
Slide to adjust relevance metric:(2)
λ = 1
Intertopic Distance Map (via multidimensional scaling)
PC2
PC1
2
4
3
1
Marginal topic distribution
2%
5%
10%
Top-30 Mo
0
50,000
100,000
150,0
by
person
na
i
the
//
to
😭
may
checked
ko
sa
/
followed
a
automatically
and
vote
is
ng
me
my
ako
you
2022
of
unfollowed
so
in
di
Overall term frequency
Estimated term frequency within the
1. saliency(term w) = frequency(w) * [sum_t p(t
2. relevance(term w | topic t) = λ * p(w | t) + (1 -

- Import necessary libraries:

gensim: A library used for unsupervised topic modeling and natural language processing. corpora: A sub-module of Gensim for working with text corpora. pyLDAvis.gensim_models: A module for interactive topic model visualization. Compatible with Gensim models. pyLDAvis: The main library for interactive topic model visualization.

- Tokenize the text and remove stopwords:

texts: A list comprehension is used to tokenize each document in the 'text' column of the dataset and remove stopwords. It creates a nested list, where each inner list contains the words of a document.

- Create a dictionary and a corpus from the tokenized texts:

dictionary: A Gensim dictionary is created from the tokenized texts. This dictionary maps the words in the dataset to unique integer IDs. corpus: The tokenized texts are converted into a bag-of-words (BoW) format, where each document is represented as a list of tuples containing the integer ID of the word and its frequency in the document.

- Train the LDA model:

num_topics: The number of topics to be generated by the LDA model is set to 4. lda_model: The LDA model is trained using the BoW corpus, the number of topics, and the Gensim dictionary. The passes parameter determines the number of times the LDA algorithm iterates over the entire corpus.

- Visualize the topics using pyLDAvis:

lda_display: The trained LDA model, the BoW corpus, and the Gensim dictionary are used to prepare the data for visualization with pyLDAvis. pyLDAvis.display(lda_display): This line of code displays the interactive LDA topic visualization. Each circle represents a topic, and the size of the circle corresponds to the prevalence of the topic in the dataset. The distance between circles represents the similarity between topics. Words that are important for each topic are displayed on the right side of the visualization.

VADER (Valence Aware Dictionary and sEntiment Reasoner)

In [ ]:
```
import nltk
nltk.download('punkt')
nltk.download('vader_lexicon')
import pandas as pd
from nltk.sentiment import SentimentIntensityAnalyzer

# Define the count_negative_words function (as previously shown)
def count_negative_words(text, threshold=-0.2):
    sia = SentimentIntensityAnalyzer()
    tokens = nltk.word_tokenize(text)
    negative_words = []

    for token in tokens:
        sentiment_score = sia.polarity_scores(token)["compound"]
        if sentiment_score <= threshold:
            negative_words.append(token)

    return len(negative_words), negative_words

# Apply the count_negative_words function to the text column
data["negative_word_count"], data["negative_words"] = zip(*data["text"].apply(count_negative_words))

# Preview the dataset with the new columns
print(data.head())
```

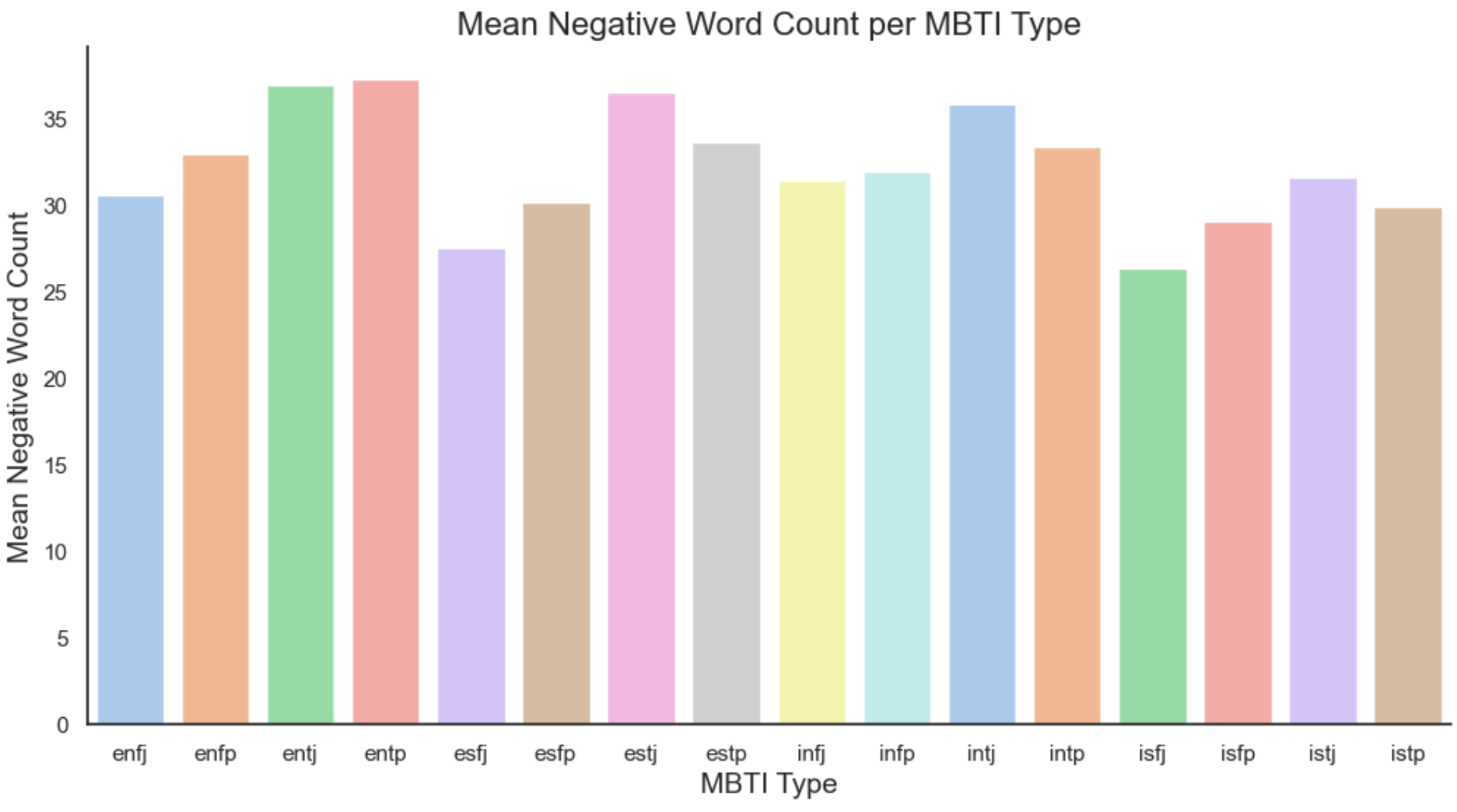
- Import the necessary libraries: nltk for natural language processing and pandas for data manipulation.
- Download the required NLTK resources, punkt and vader_lexicon. punkt is a tokenizer model and vader_lexicon is a lexicon for sentiment analysis.
- Import the SentimentIntensityAnalyzer class from the nltk.sentiment module.
- Define a function called count_negative_words that takes two arguments: text (the text to analyze) and threshold (default value of -0.2, to filter negative words based on their sentiment score). The function does the following:

Initialize a SentimentIntensityAnalyzer object called sia. Tokenize the input text using nltk.word_tokenize and store the result in the tokens variable. Create an empty list called negative_words. Iterate over each token in tokens and calculate its sentiment score using sia.polarity_scores(token)["compound"]. If the sentiment score is less than or equal to the specified threshold, append the token to the negative_words list. Return the length of the negative_words list and the list itself.

- Apply the count_negative_words function to the "text" column of the DataFrame data and create two new columns, "negative_word_count" and "negative_words". The zip function is used to unpack the tuple returned by count_negative_words and assign the values to the appropriate columns.
- Preview the first few rows of the updated DataFrame data with the new columns "negative_word_count" and "negative_words".

Negative words from each personality

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate the mean negative word count per MBTI type
mean_negative_word_count = data.groupby('label')['negative_word_count'].mean()

# Set the Seaborn style to white for a minimalistic look
sns.set_style("white")

# Create a bar plot with pastel colors
plt.figure(figsize=(12, 6))
ax = sns.barplot(x=mean_negative_word_count.index, y=mean_negative_word_count.values, palette='pastel')

# Customize the plot
ax.set_title('Mean Negative Word Count per MBTI Type', fontsize=16)
ax.set_xlabel('MBTI Type', fontsize=14)
ax.set_ylabel('Mean Negative Word Count', fontsize=14)

# Remove the top and right spines for a cleaner look
sns.despine()

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



- Calculate the mean negative word count for each MBTI type
- Create a bar plot with pastel colors: plt.figure(figsize=(12, 6)): Set the figure size to 12 inches wide and 6 inches tall.

sns.barplot(): Create a bar plot. x=mean_negative_word_count.index: Set the x-axis values to be the MBTI types. y=mean_negative_word_count.values: Set the y-axis values to be the mean negative word count. palette='pastel': Use pastel colors for the bars.

- Remove the top and right spines for a cleaner look

In [ ]:
```
import nltk

nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('sentiwordnet')
from nltk.corpus import sentiwordnet as swn
from nltk.corpus import wordnet as wn
from nltk import word_tokenize, pos_tag

def count_positive_words(text):
    positive_words = []

    tokens = word_tokenize(text)
    tagged_tokens = pos_tag(tokens)

    for token, pos in tagged_tokens:
        wn_pos = get_wordnet_pos(pos)
        if wn_pos not in (wn.NOUN, wn.VERB, wn.ADJ, wn.ADV):
            continue

        synsets = wn.synsets(token, pos=wn_pos)
        if not synsets:
            continue

        sentiment = swn.senti_synset(synsets[0].name())
        if sentiment.pos_score() > sentiment.neg_score():
            positive_words.append(token)

    return len(positive_words), positive_words

# Helper function to map NLTK POS tags to WordNet POS tags
def get_wordnet_pos(treebank_tag):
    if treebank_tag.startswith('J'):
        return wn.ADJ
    elif treebank_tag.startswith('V'):
        return wn.VERB
    elif treebank_tag.startswith('N'):
        return wn.NOUN
    elif treebank_tag.startswith('R'):
        return wn.ADV
    else:
        return None
```

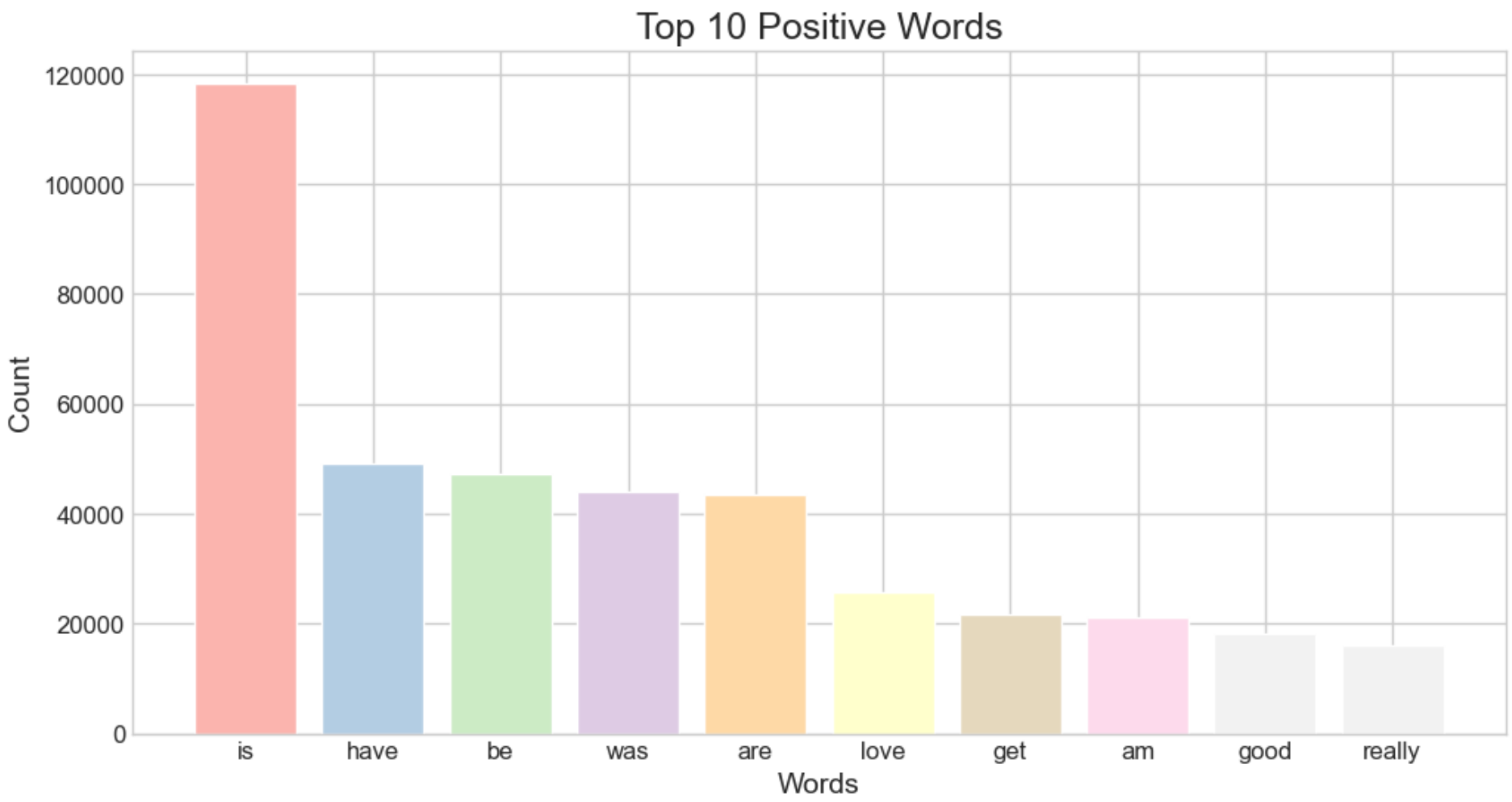
```
data["positive_word_count"], data["positive_words"] = zip(*data["text"].apply(count_positive_words))
print(data.head())
```

- Import necessary libraries and download required NLTK data
- This imports SentiWordNet, WordNet, and two useful functions from NLTK: word_tokenize (for tokenizing text into words) and pos_tag (for assigning part-of-speech tags to tokens).
- Define the count_positive_words function This function takes a text input, tokenizes it into words, and assigns part-of-speech tags to each token. Then, it iterates through each token and its associated part-of-speech.
- Process each token and identify positive words:For each token, this code converts the part-of-speech tag to a WordNet-compatible tag using the get_wordnet_pos() function. It then retrieves the WordNet synsets (synonym sets) for the token and its part-of-speech. Next, it gets the sentiment scores for the first synset (if any) using SentiWordNet. If the positive sentiment score is greater than the negative sentiment score, the token is considered a positive word and added to the positive_words list.
- Define the get_wordnet_pos helper function: This function converts the treebank part-of-speech tags used by NLTK's pos_tag function to WordNet-compatible part-of-speech tags.
- Apply the count_positive_words function to the dataset and create new columns: This code applies the count_positive_words function to the "text" column of the dataset and creates two new columns, "positive_word_count" and "positive_words", containing the number of positive words and the list of positive words for each row, respectively. Finally, it prints the first few rows of the dataset to show the new columns.

Top 10 Positive Words

In [ ]:
```
from collections import Counter

# Get the list of all positive words in the dataset
all_positive_words = [word for words_list in data["positive_words"] for word in words_list]

# Count the occurrences of each positive word
positive_word_counts = Counter(all_positive_words)

# Get the top 10 positive words
top_10_positive_words = positive_word_counts.most_common(10)
import matplotlib.pyplot as plt

# Set the minimal style
plt.style.use('seaborn-whitegrid')

# Set the pastel colors
colors = plt.cm.Pastel1(range(10))

# Prepare the data
words, counts = zip(*top_10_positive_words)

# Create the bar chart
fig, ax = plt.subplots(figsize=(12, 6))
ax.bar(words, counts, color=colors)

# Customize the chart
ax.set_title('Top 10 Positive Words', fontsize=18)
ax.set_xlabel('Words', fontsize=14)
ax.set_ylabel('Count', fontsize=14)
ax.tick_params(axis='both', which='major', labelsize=12)

# Show the chart
plt.show()
```

```
/var/folders/wm/_56vlq9x7plb_ljg3qsjr2nh0000gn/T/ipykernel_2721/2254649757.py:14: MatplotlibDeprecationWarning:

The seaborn styles shipped by Matplotlib are deprecated since 3.6, as they no longer correspond to the styles sh
ipped by seaborn. However, they will remain available as 'seaborn-v0_8-<style>'. Alternatively, directly use the
seaborn API instead.
```

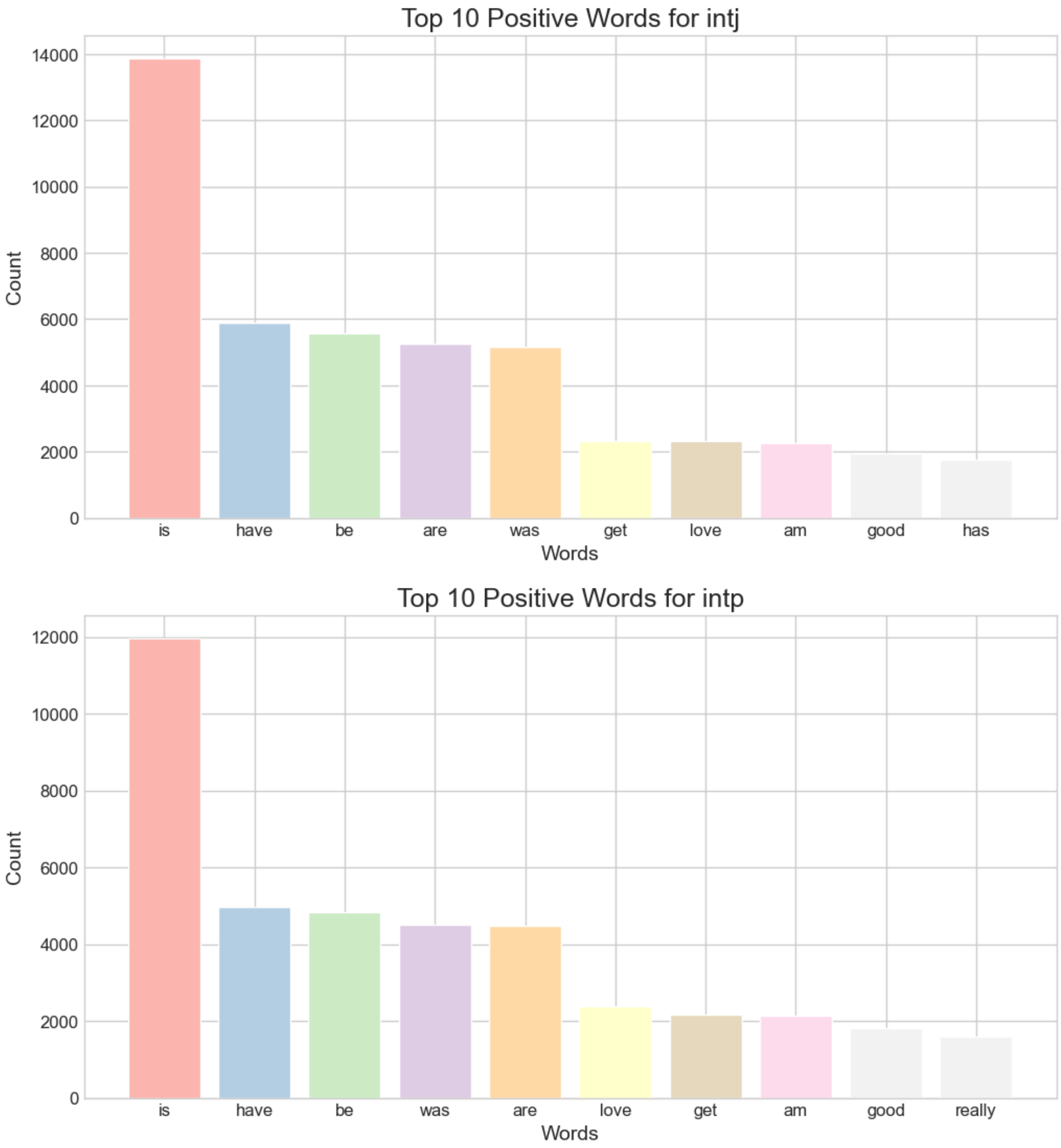


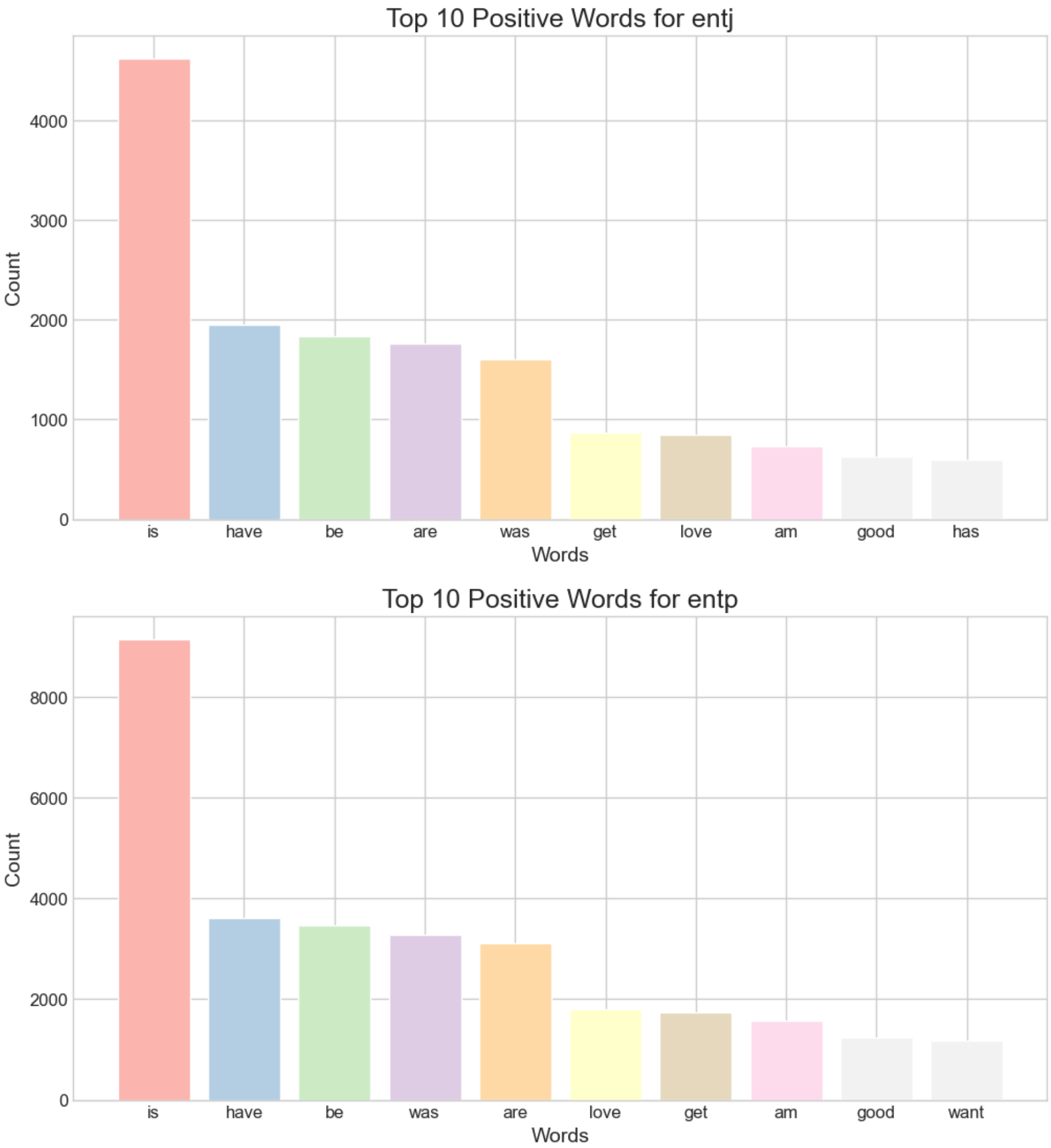
In summary, this code block creates and displays a bar chart of the top 10 positive words in the dataset, using a minimal style and pastel colors.

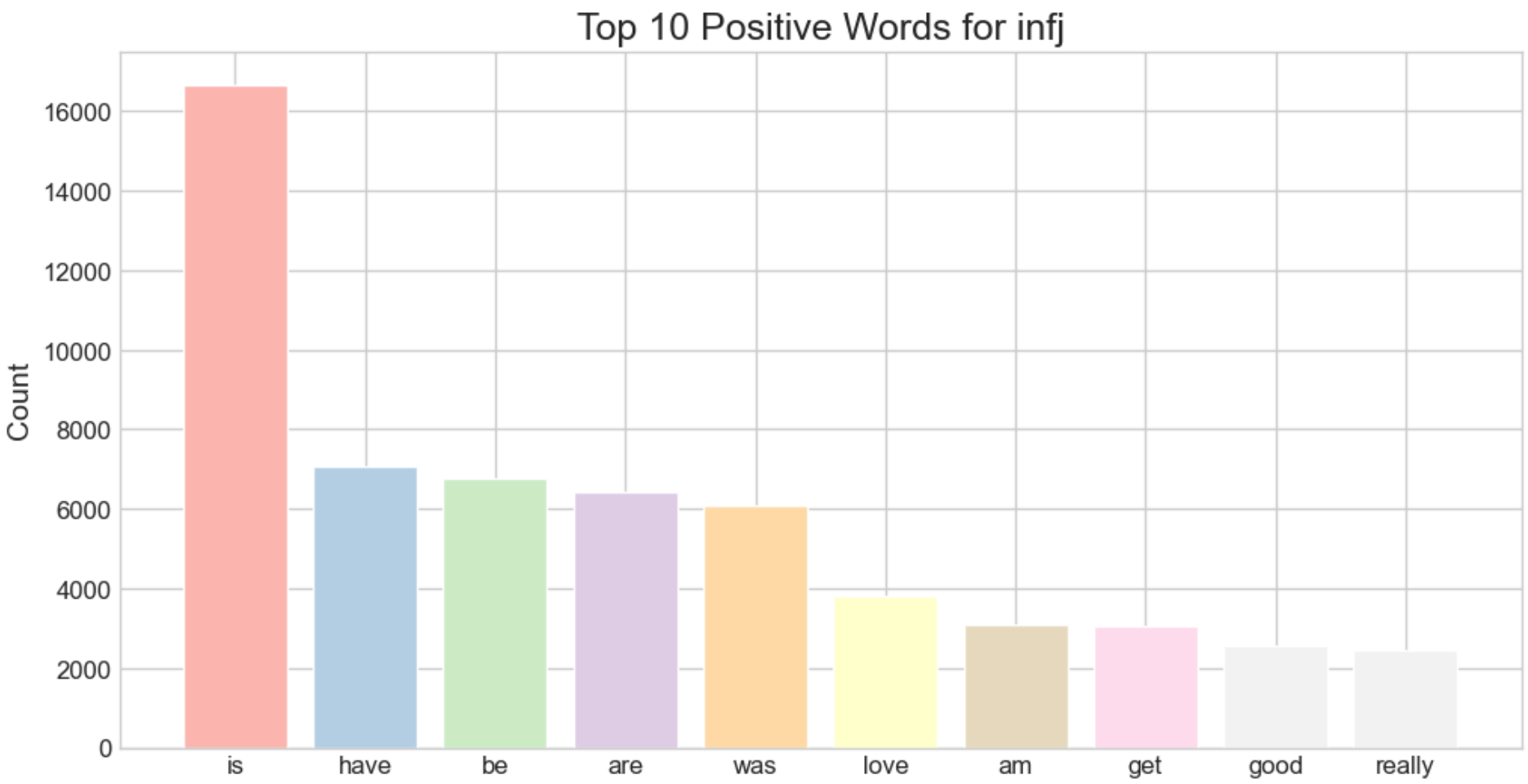
Positive words by MBTI Type

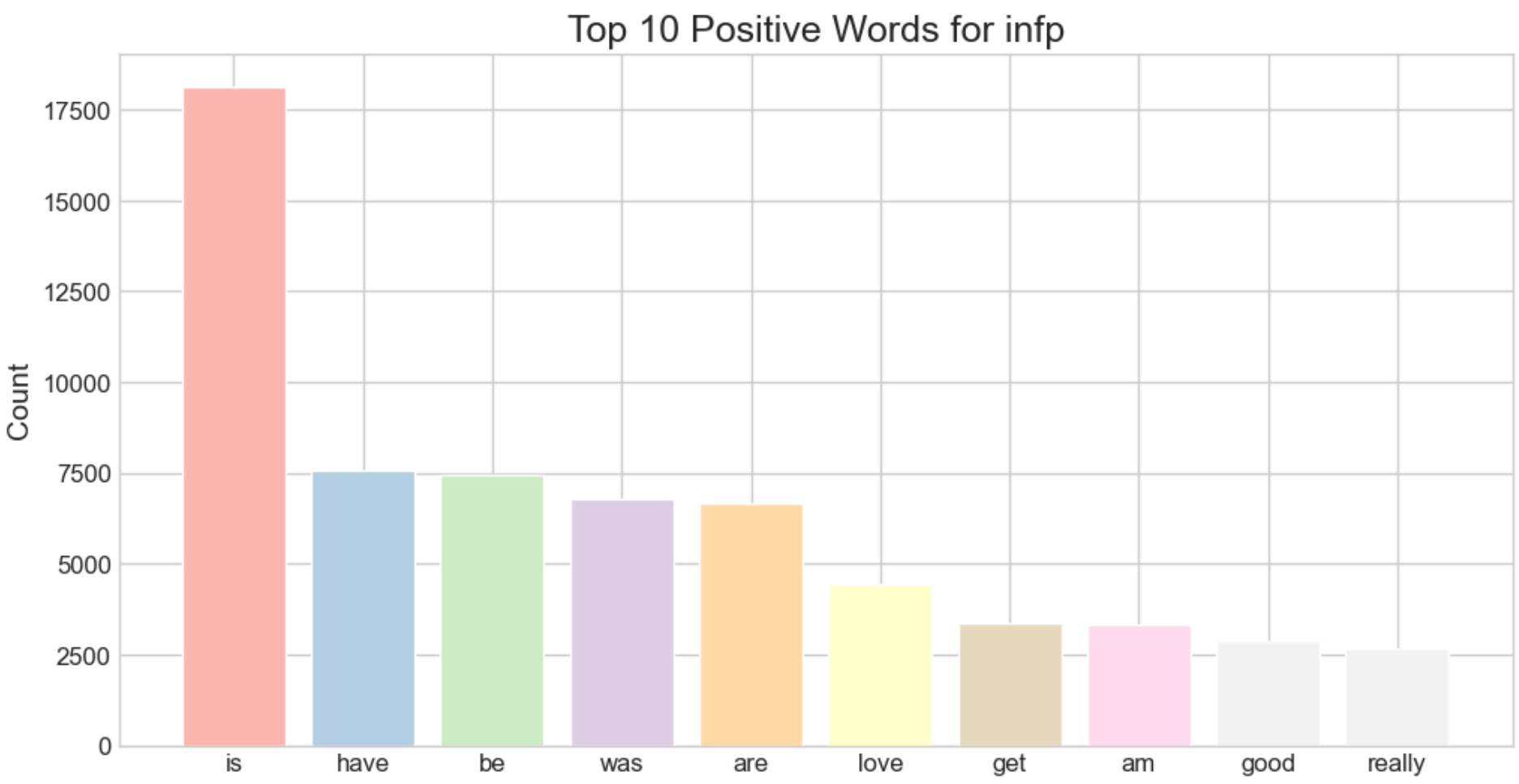
In [ ]:
```python
from collections import defaultdict

# Create a dictionary to store the positive words for each label
positive_words_by_label = defaultdict(list)

# Iterate through the dataset and add the positive words to the appropriate label list
for index, row in data.iterrows():
    label = row['label']
    positive_words = row['positive_words']
    positive_words_by_label[label].extend(positive_words)

# Calculate the top 10 positive words for each label
top_positive_words_by_label = {}
for label, words_list in positive_words_by_label.items():
    word_counts = Counter(words_list)
    top_positive_words_by_label[label] = word_counts.most_common(10)
# Set the minimal style
plt.style.use('seaborn-whitegrid')

def plot_top_positive_words(words_counts, label):
    # Set the pastel colors
    colors = plt.cm.Pastel1(range(10))

    # Prepare the data
    words, counts = zip(*words_counts)

    # Create the bar chart
    fig, ax = plt.subplots(figsize=(12, 6))
    ax.bar(words, counts, color=colors)

    # Customize the chart
    ax.set_title(f'Top 10 Positive Words for {label}', fontsize=18)
    ax.set_xlabel('Words', fontsize=14)
    ax.set_ylabel('Count', fontsize=14)
    ax.tick_params(axis='both', which='major', labelsize=12)

    # Show the chart
    plt.show()

# Create a bar chart for each MBTI type
for label, words_counts in top_positive_words_by_label.items():
    plot_top_positive_words(words_counts, label)
```

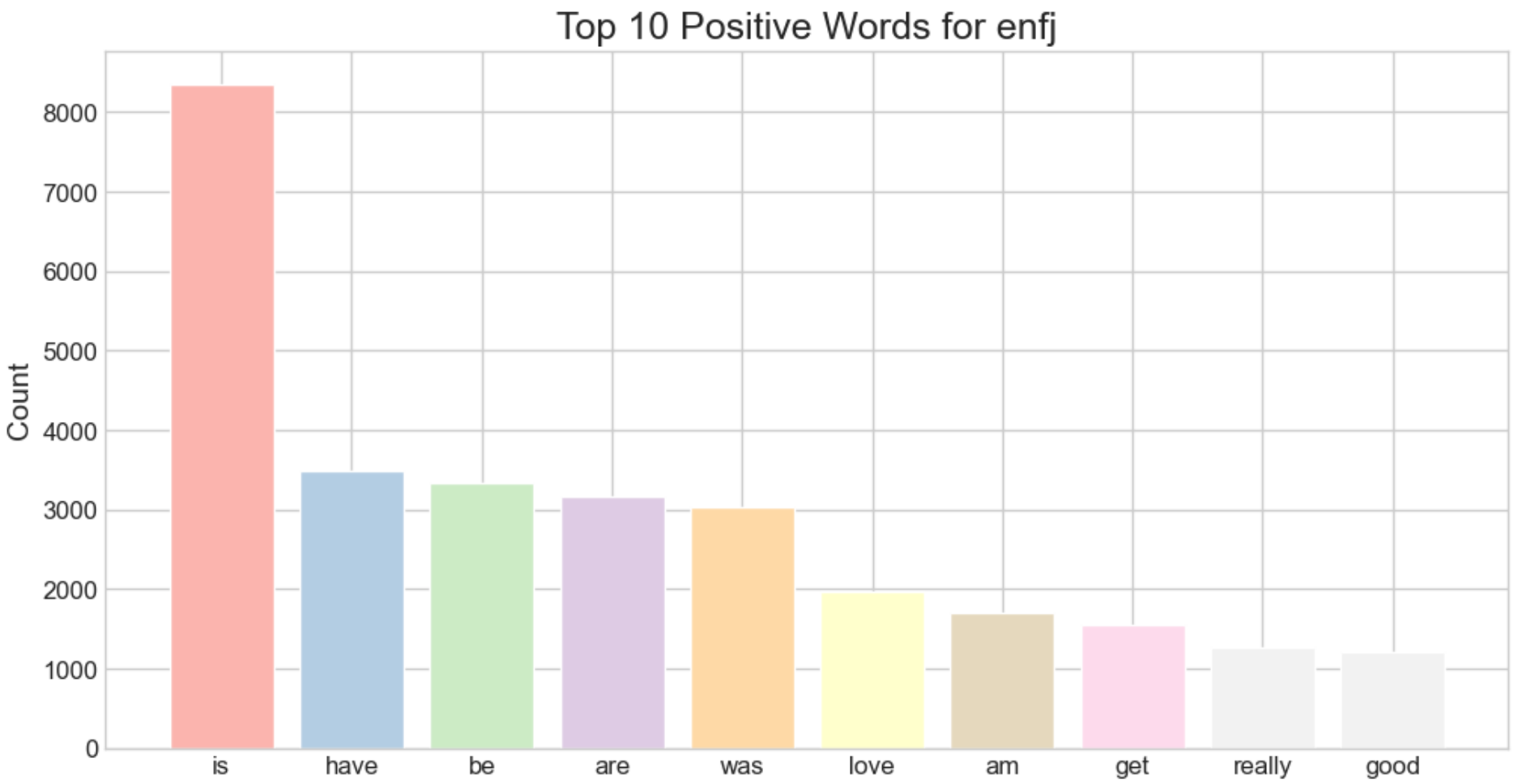
```
/var/folders/wm/_56vlq9x7plb_ljg3qsjr2nh0000gn/T/ipykernel_2721/1849289743.py:18: MatplotlibDeprecationWarning:

The seaborn styles shipped by Matplotlib are deprecated since 3.6, as they no longer correspond to the styles sh
ipped by seaborn. However, they will remain available as 'seaborn-v0_8-<style>'. Alternatively, directly use the
seaborn API instead.
```

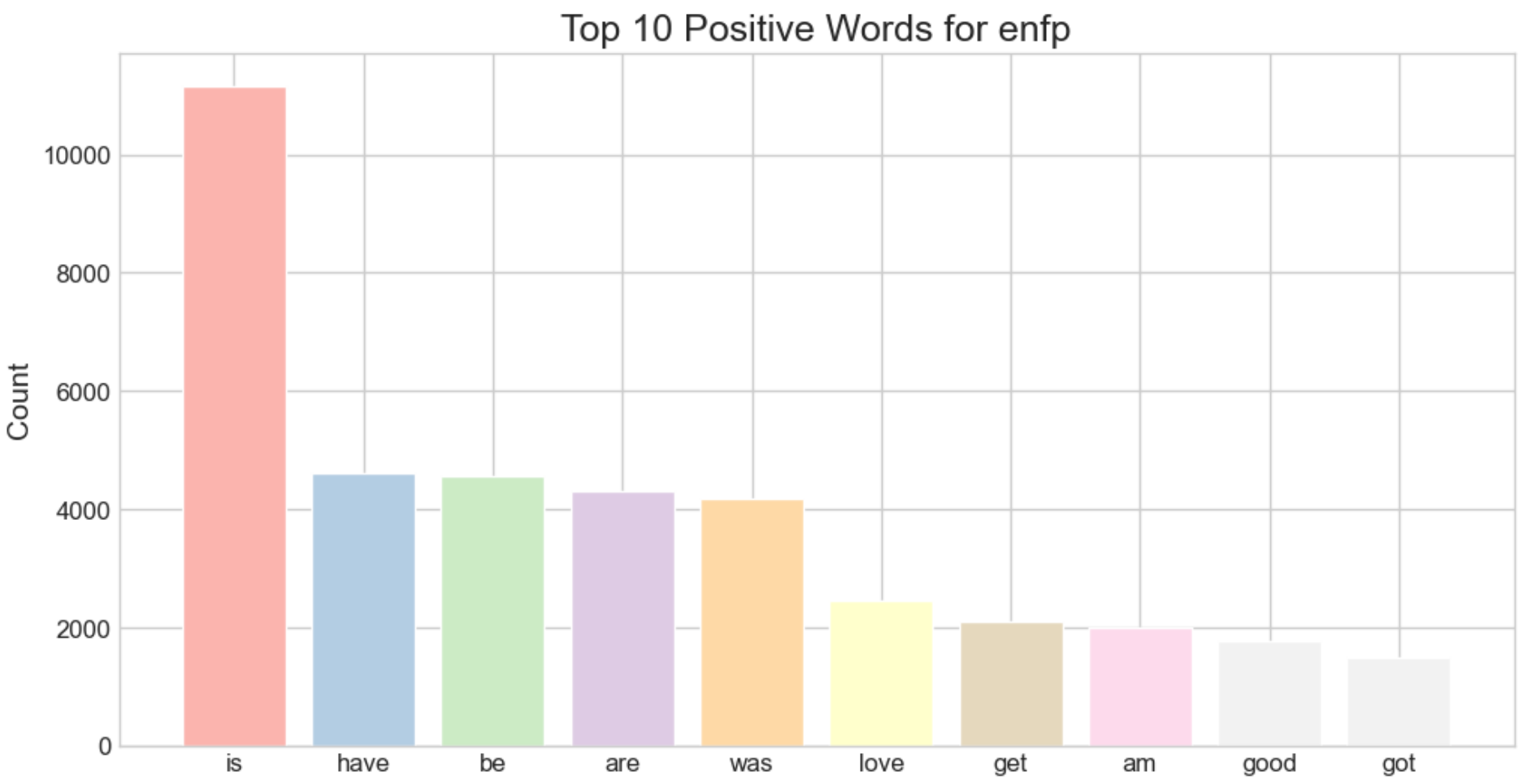


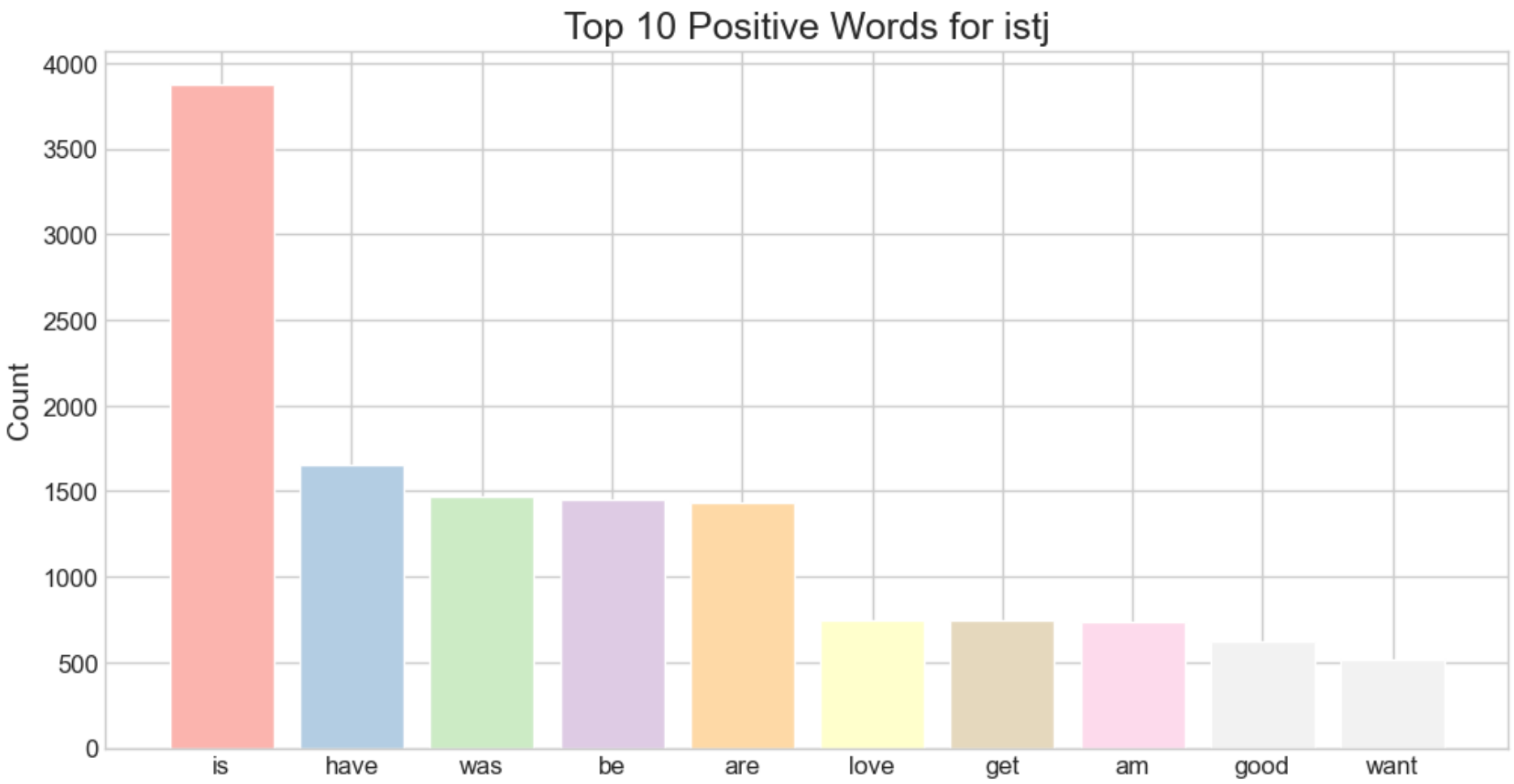

Top 10 Positive Words for entj
Count
4000
3000
2000
1000
0
is
have
be
are
was
get
love
am
good
has
Words
Top 10 Positive Words for entp
Count
8000
6000
4000
2000
0
is
have
be
was
are
love
get
am
good
want
Words

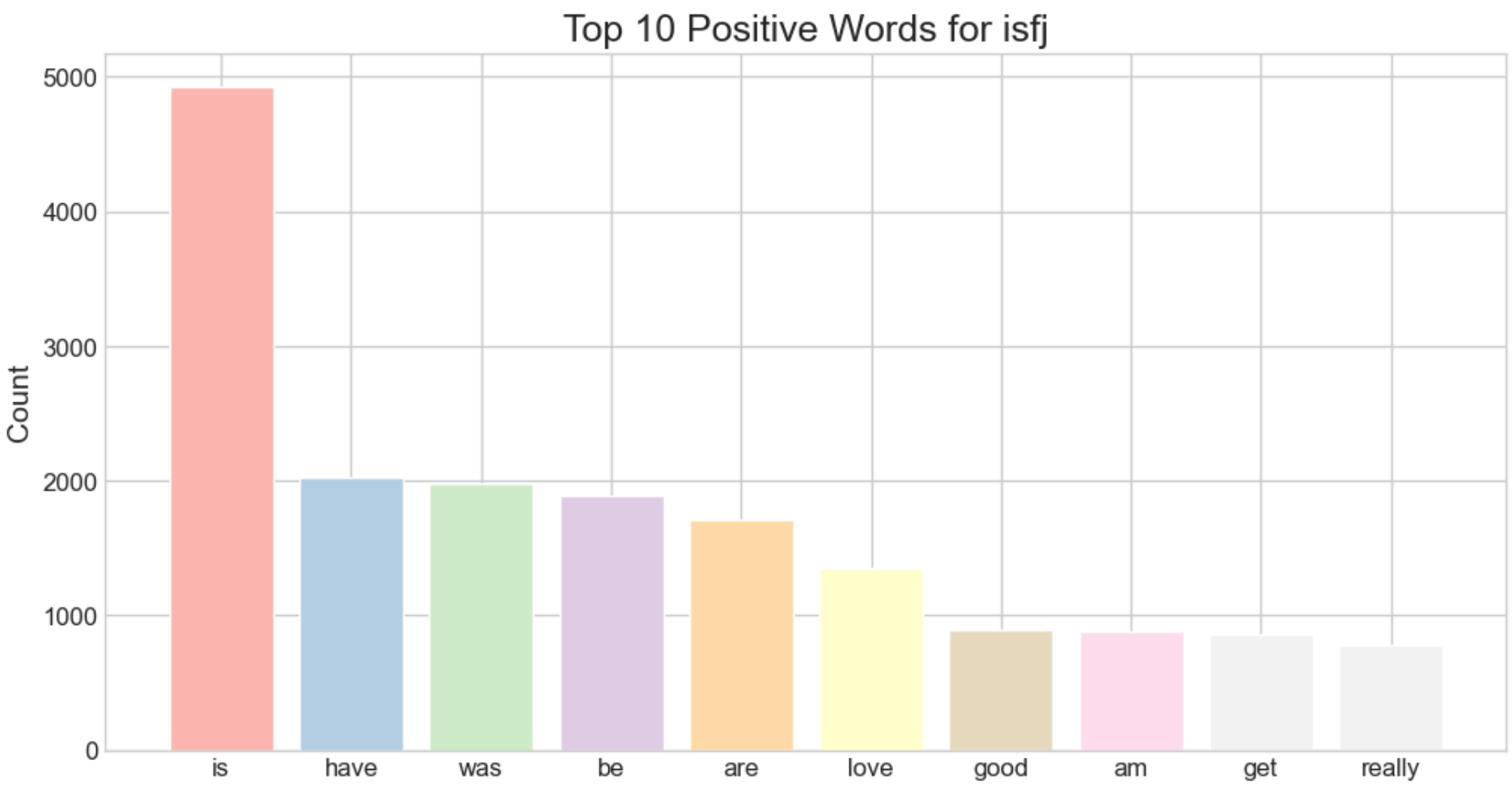

Top 10 Positive Words for infj
Count
16000
14000
12000
10000
8000
6000
4000
2000
0
is
have
be
are
was
love
am
get
good
really
Words

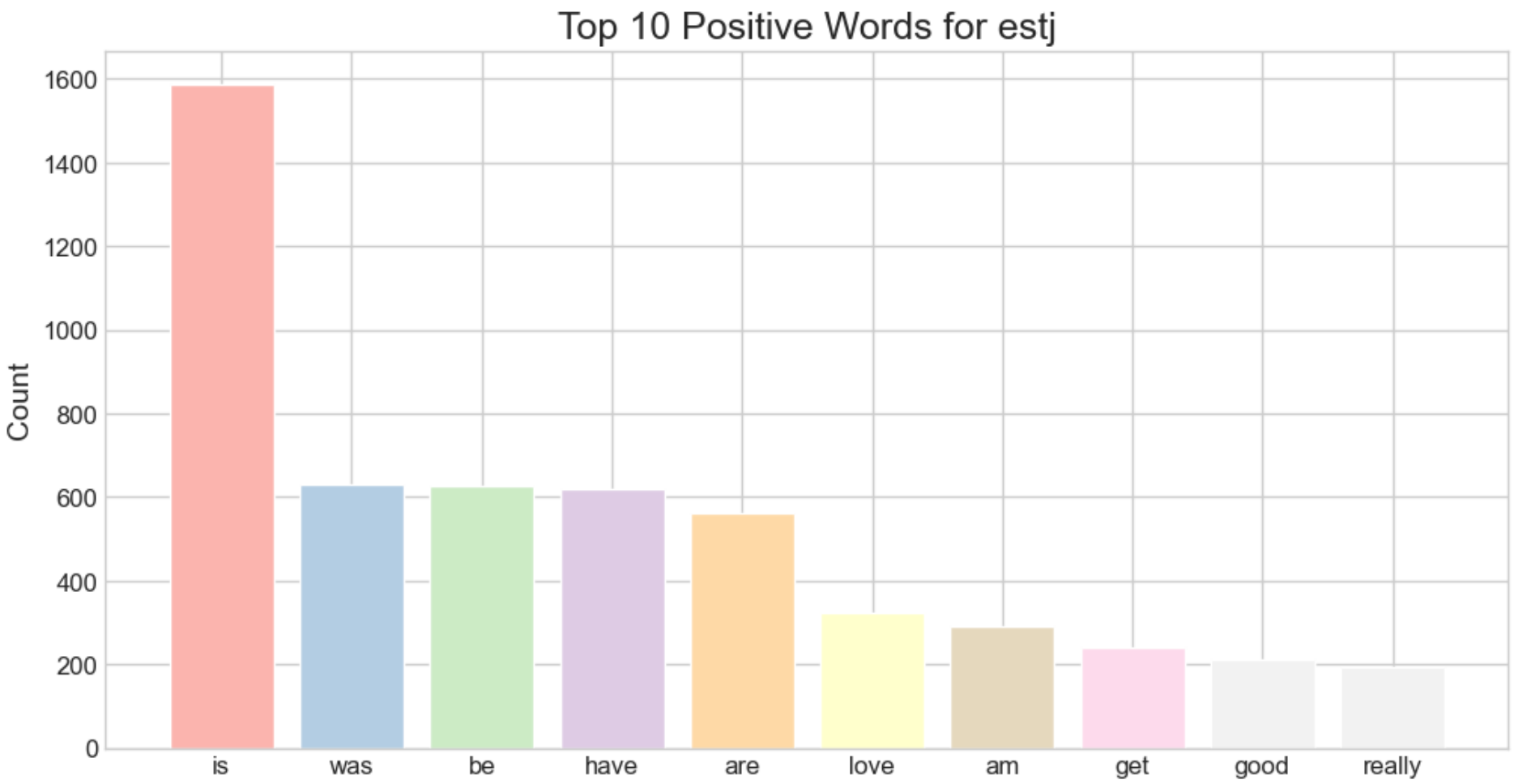

Top 10 Positive Words for infp
Count
17500
15000
12500
10000
7500
5000
2500
0
is
have
be
was
are
love
get
am
good
really
Words

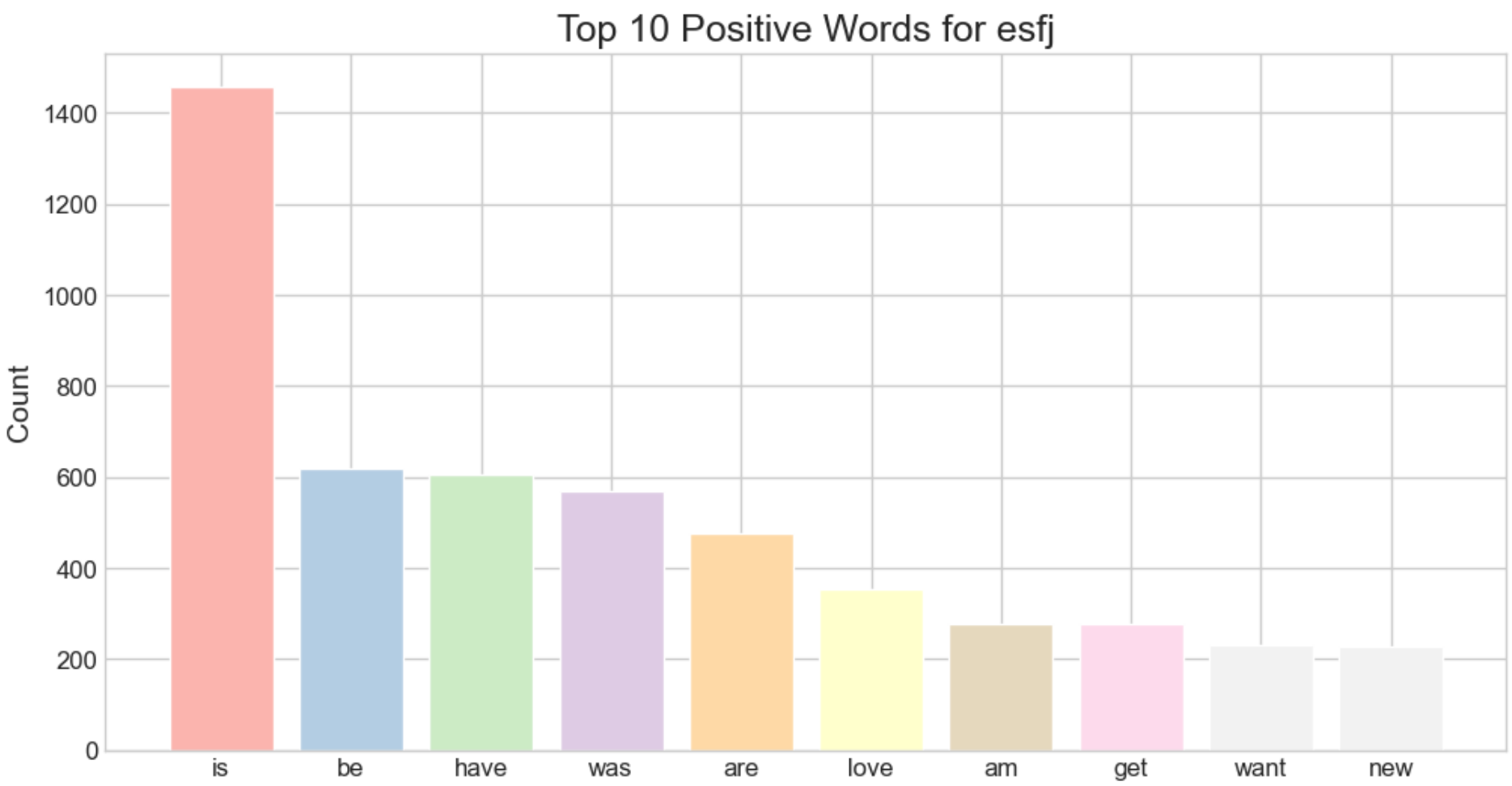

Top 10 Positive Words for enfj
Count
8000
7000
6000
5000
4000
3000
2000
1000
0
is
have
be
are
was
love
am
get
really
good
Words

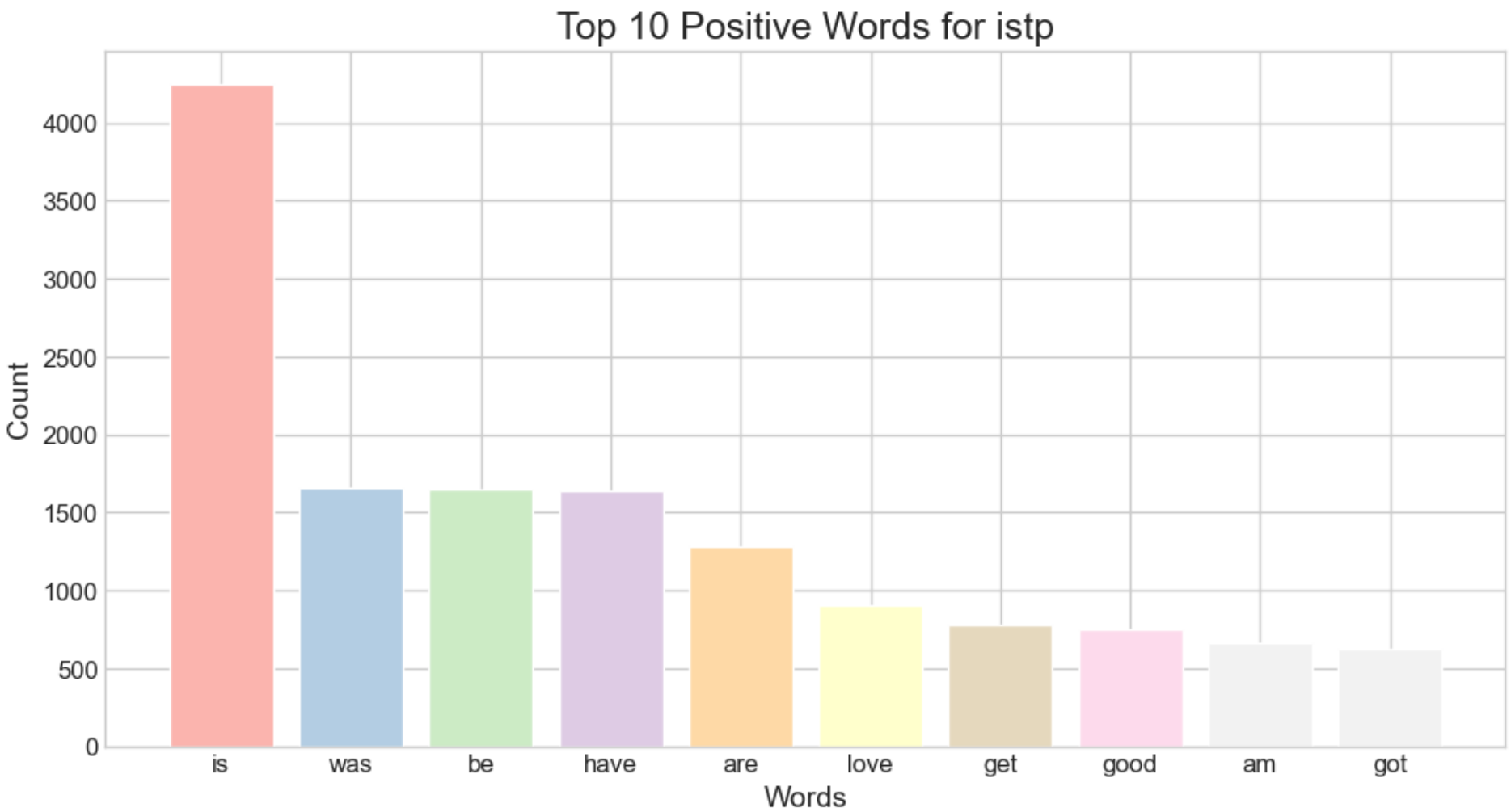

Top 10 Positive Words for enfp
Count
10000
8000
6000
4000
2000
0
is
have
be
are
was
love
get
am
good
got
Words

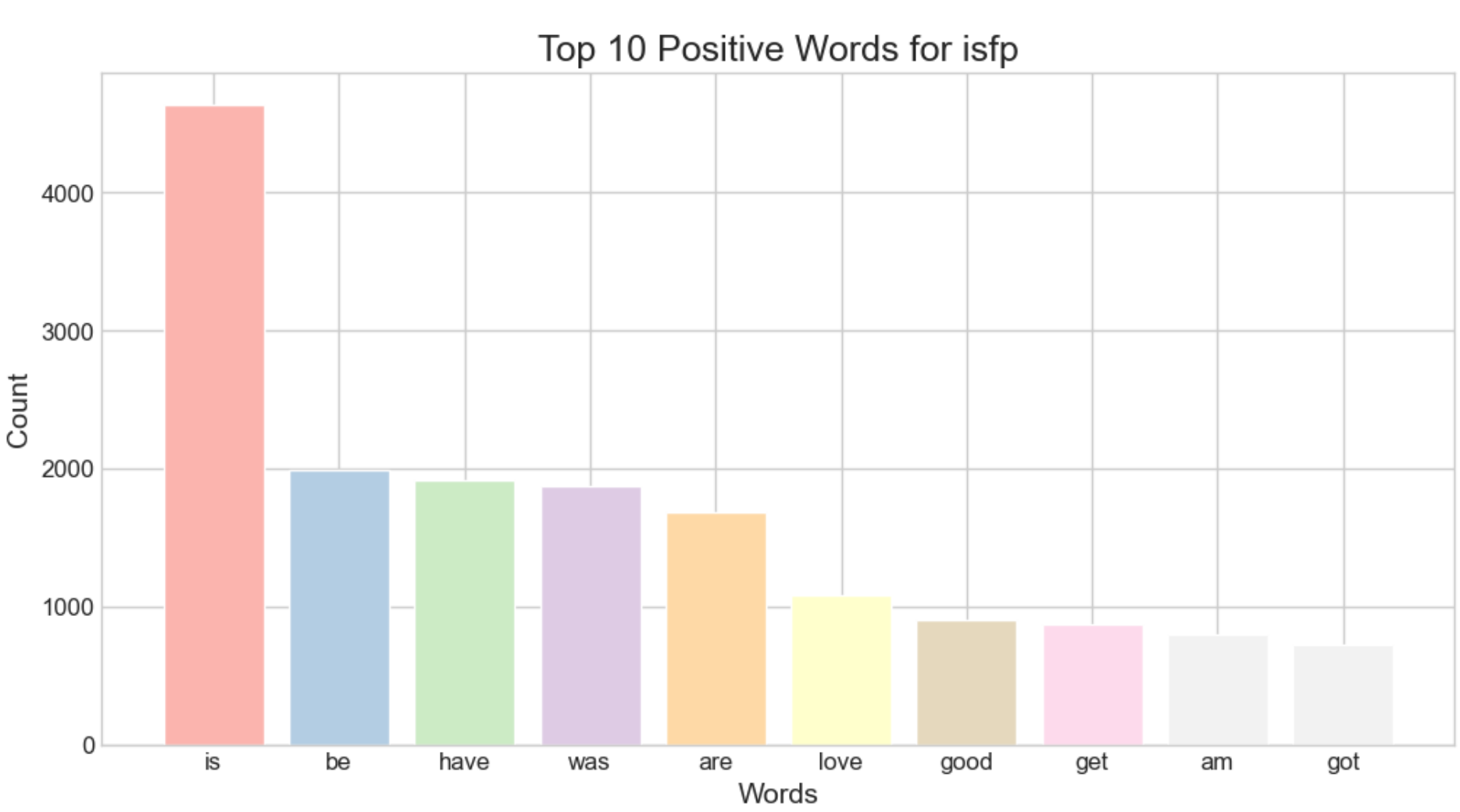

Top 10 Positive Words for istj
Count
4000
3500
3000
2500
2000
1500
1000
500
0
is
have
was
be
are
love
get
am
good
want
Words

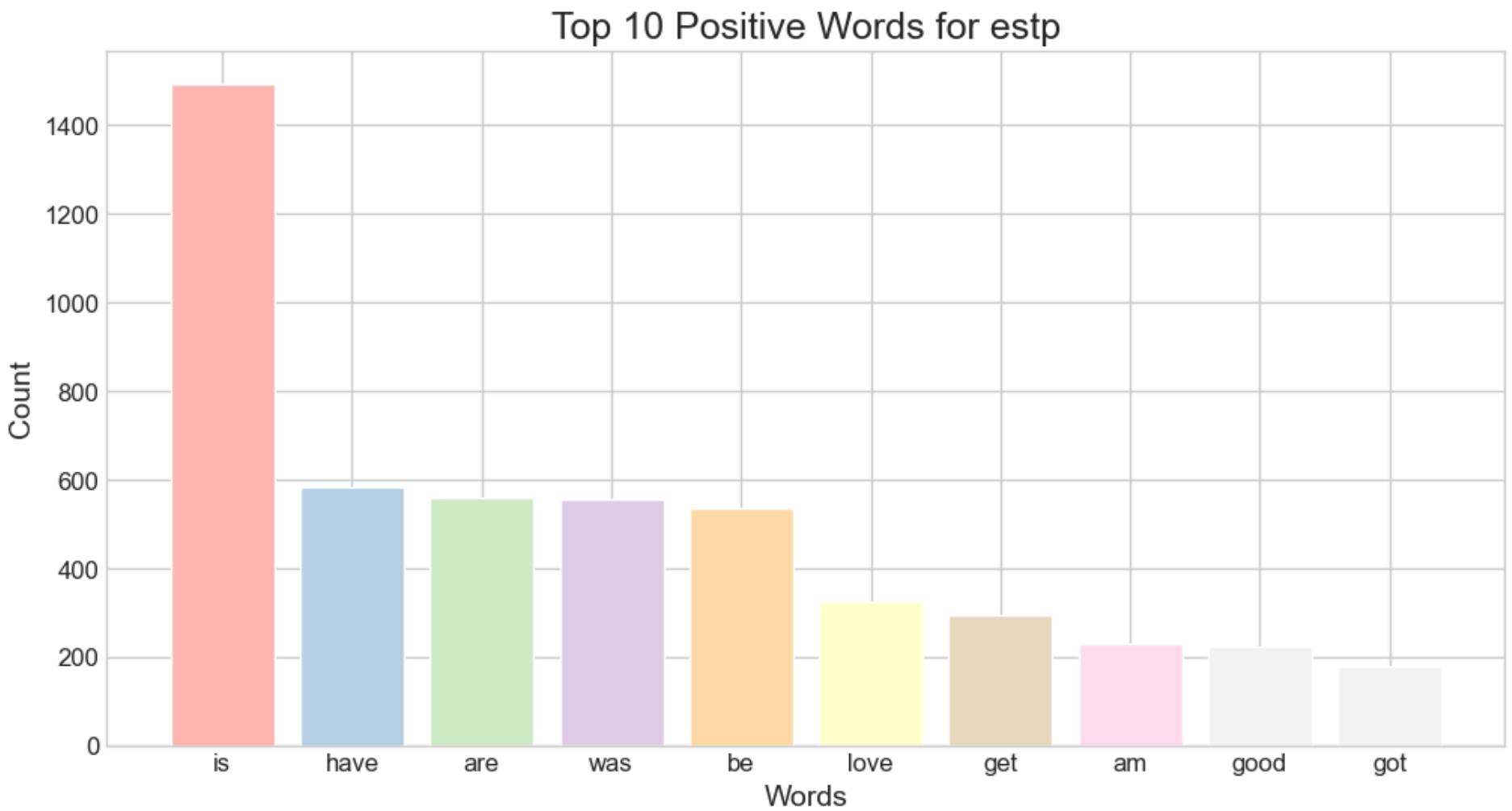

Top 10 Positive Words for isfj
Count
5000
4000
3000
2000
1000
0
is
have
was
be
are
love
good
am
get
really
Words

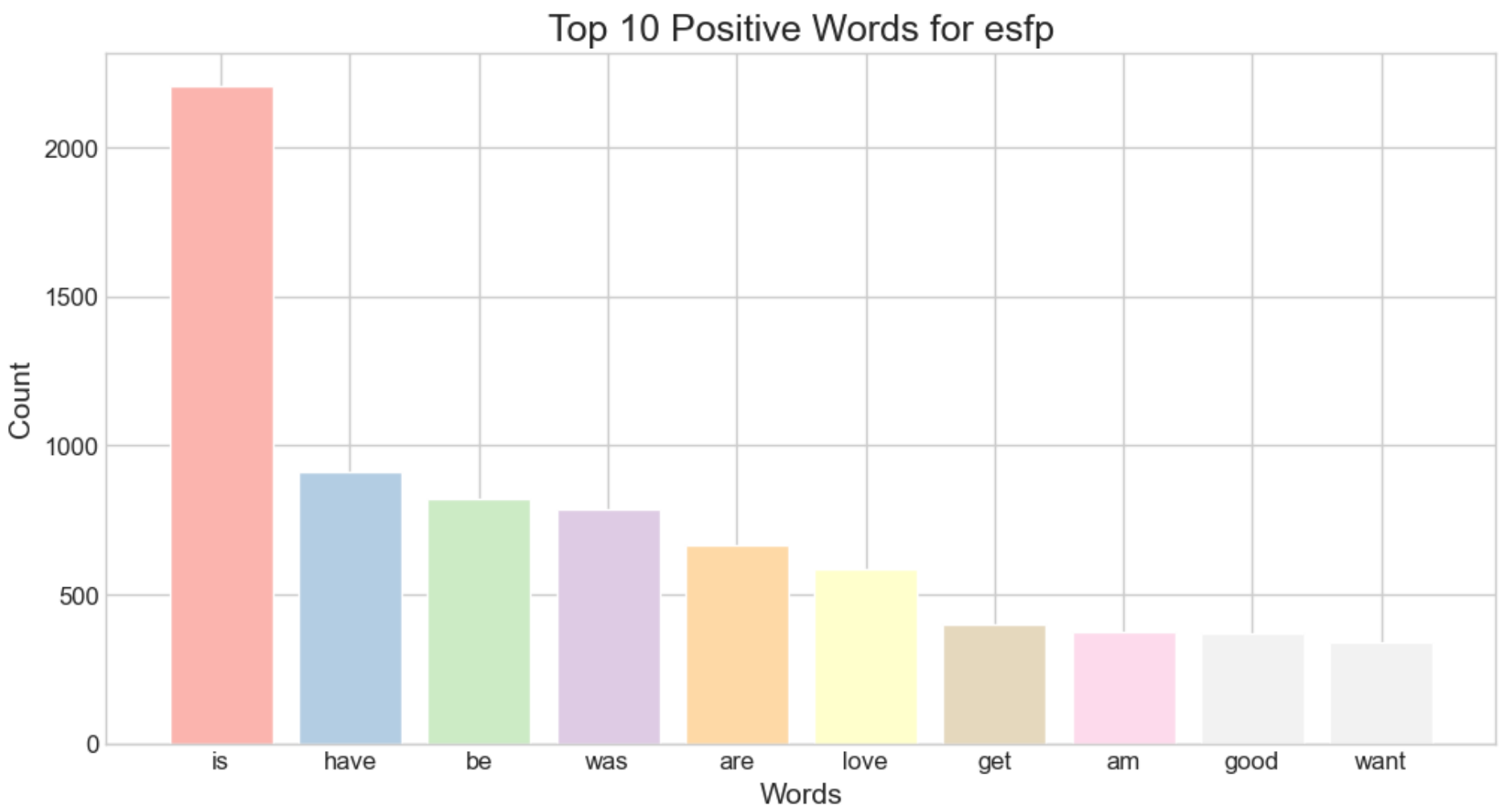

Top 10 Positive Words for estj
Count
1600
1400
1200
1000
800
600
400
200
0
is
was
be
have
are
love
am
get
good
really
Words


Top 10 Positive Words for esfj
Count
1400
1200
1000
800
600
400
200
0
is
be
have
was
are
love
am
get
want
new
Words


Top 10 Positive Words for istp
Count
4000
3500
3000
2500
2000
1500
1000
500
0
is
was
be
have
are
love
get
good
am
got
Words

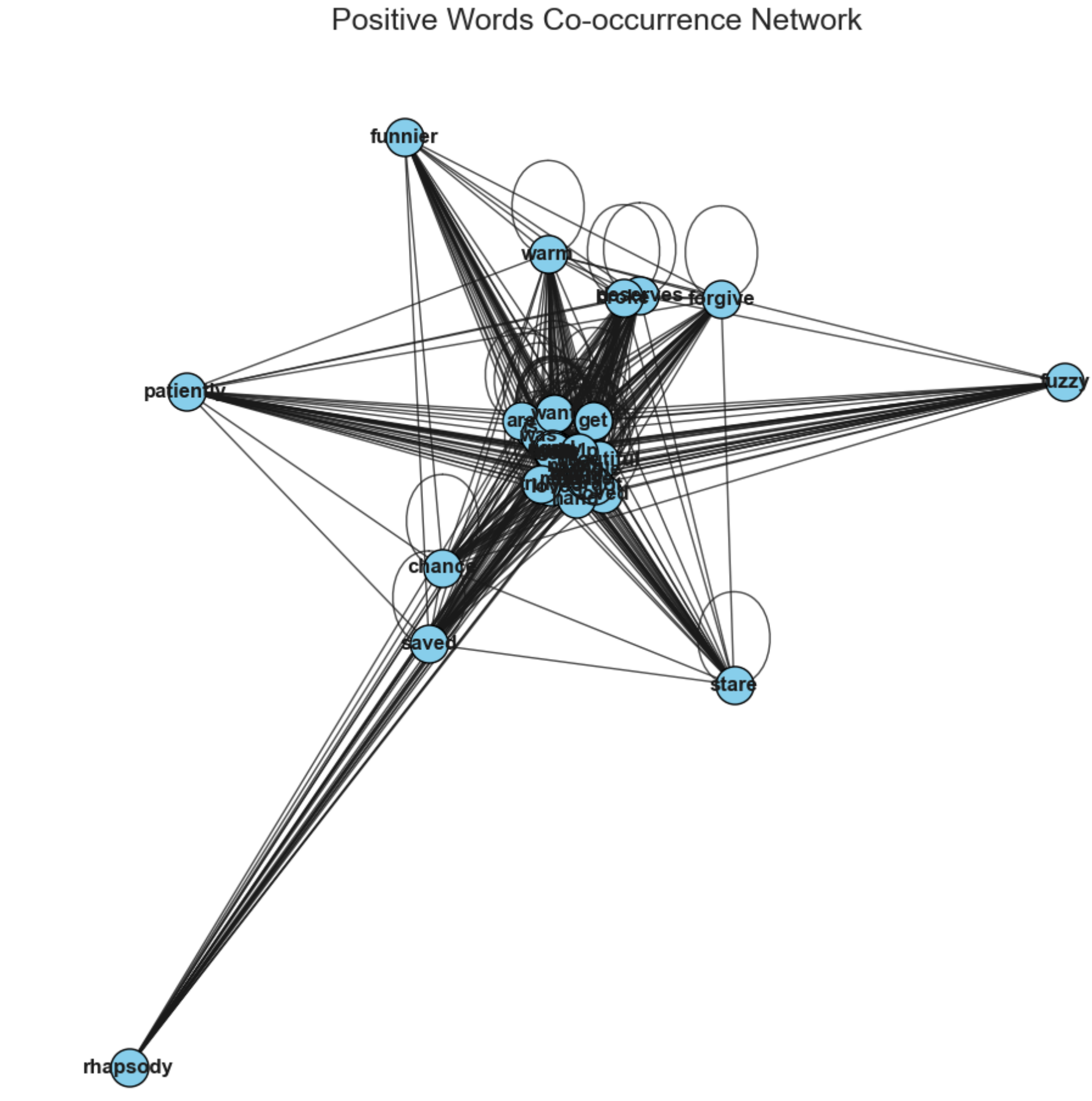

Top 10 Positive Words for isfp
Count
4000
3000
2000
1000
0
is
be
have
was
are
love
good
get
am
got
Words





- Import the defaultdict class from the collections module. defaultdict is a subclass of the built-in dict class that overrides the **missing**() method, providing a default value for a nonexistent key.
- Create a dictionary called positive_words_by_label to store the positive words for each label (MBTI type) using defaultdict(list). This means that if a key (label) is not present in the dictionary, its default value will be an empty list.
- Iterate through the dataset using a for loop, where index represents the row index and row is a pandas Series containing the row data. For each row, retrieve the label (MBTI type) and the list of positive words, and then extend the list of positive words associated with the label in the positive_words_by_label dictionary.
- Calculate the top 10 positive words for each label by iterating through the positive_words_by_label dictionary. Use the Counter class from the collections module to count the occurrences of each word in the list, then store the 10 most common words and their counts in the top_positive_words_by_label dictionary.
- Set the minimal style for the plots by using the seaborn-whitegrid style.
- Define a function called plot_top_positive_words that takes two arguments: words_counts, which is a list of tuples containing the top 10 positive words and their counts, and label, which is the MBTI type. This function creates a bar chart of the top 10 positive words for the given MBTI type using pastel colors and customizes the chart with title, labels, and tick parameters.
- Finally, iterate through the top_positive_words_by_label dictionary and call the plot_top_positive_words function for each MBTI type to create and display the bar charts.

In [ ]:
```
import networkx as nx
import itertools

# Set the co-occurrence threshold
threshold = 5

# Initialize the graph
G = nx.Graph()

# Iterate through the texts in the dataset
for text in data['text']:
    # Tokenize the text
    tokens = nltk.word_tokenize(text)

    # Keep only the positive words
    positive_tokens = [token.lower() for token in tokens if token.lower() in positive_words]

    # Update the co-occurrence counts in the graph
    for pair in itertools.combinations(positive_tokens, 2):
        if G.has_edge(pair[0], pair[1]):
            G[pair[0]][pair[1]]['weight'] += 1
        else:
            G.add_edge(pair[0], pair[1], weight=1)

# Filter out edges with low co-occurrence counts
edges_to_remove = [(u, v) for u, v, weight in G.edges(data='weight') if weight < threshold]
G.remove_edges_from(edges_to_remove)

# Remove isolated nodes
G.remove_nodes_from(list(nx.isolates(G)))

plt.figure(figsize=(12, 12))

# Set the node positions using the spring layout algorithm
pos = nx.spring_layout(G, seed=42)

# Draw nodes
nx.draw_networkx_nodes(G, pos, node_color='skyblue', edgecolors='black', node_size=500)

# Draw edges
nx.draw_networkx_edges(G, pos, width=1, alpha=0.8)

# Draw labels
nx.draw_networkx_labels(G, pos, font_size=12, font_family='sans-serif', font_weight='bold')

# Customize the plot
plt.axis('off')
plt.title('Positive Words Co-occurrence Network', fontsize=18)
plt.show()
```

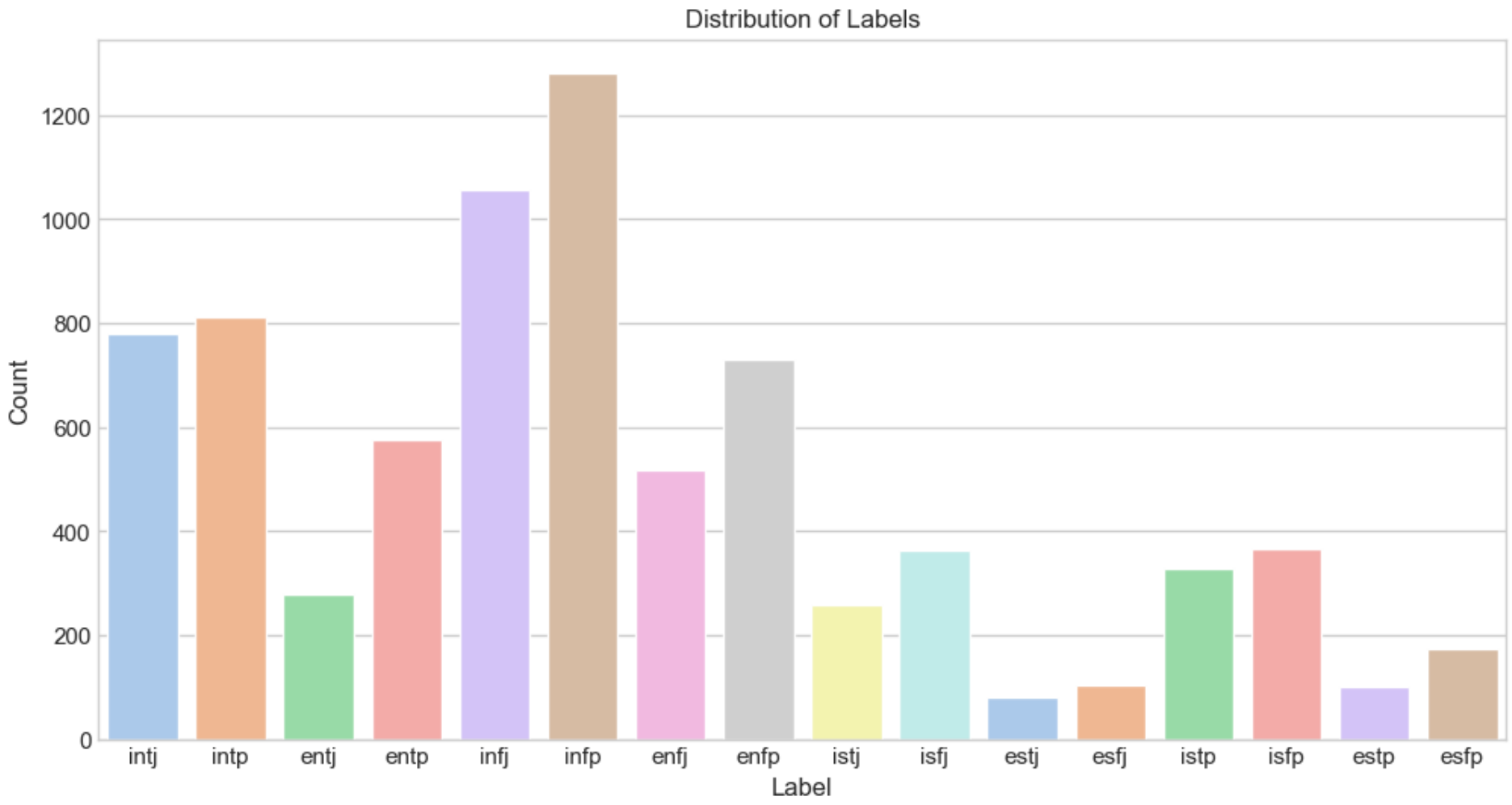


- Set the co-occurrence threshold. Only word pairs with a co-occurrence count equal to or greater than this threshold will be included in the graph.
- Initialize an empty graph using the NetworkX library
- Iterate through the texts in the dataset, tokenize each text, and retain only the positive words.
- Update the co-occurrence counts in the graph. For each pair of positive words in a text, increment the weight of the edge connecting them.
- Filter out edges with low co-occurrence counts (below the threshold) and remove isolated nodes.
- Create a figure for the plot and set the node positions using the spring layout algorithm.
- Draw nodes, edges, and labels on the graph.
- The resulting graph shows the relationships between positive words that co-occur frequently in the dataset. Nodes represent positive words, and edges represent co-occurrences between pairs of words. The graph only includes word pairs with co-occurrence counts equal to or greater than the threshold.

In [ ]:
```
# Display the variables and data types in a table format
print(data.dtypes)

# Display dataset information
print(data.info())

# Display dataset summary statistics
print(data.describe())


# Display count of missing values in each variable
print(data.isnull().sum().to_frame().rename(columns={0: "Count of Missing Values"}))
```

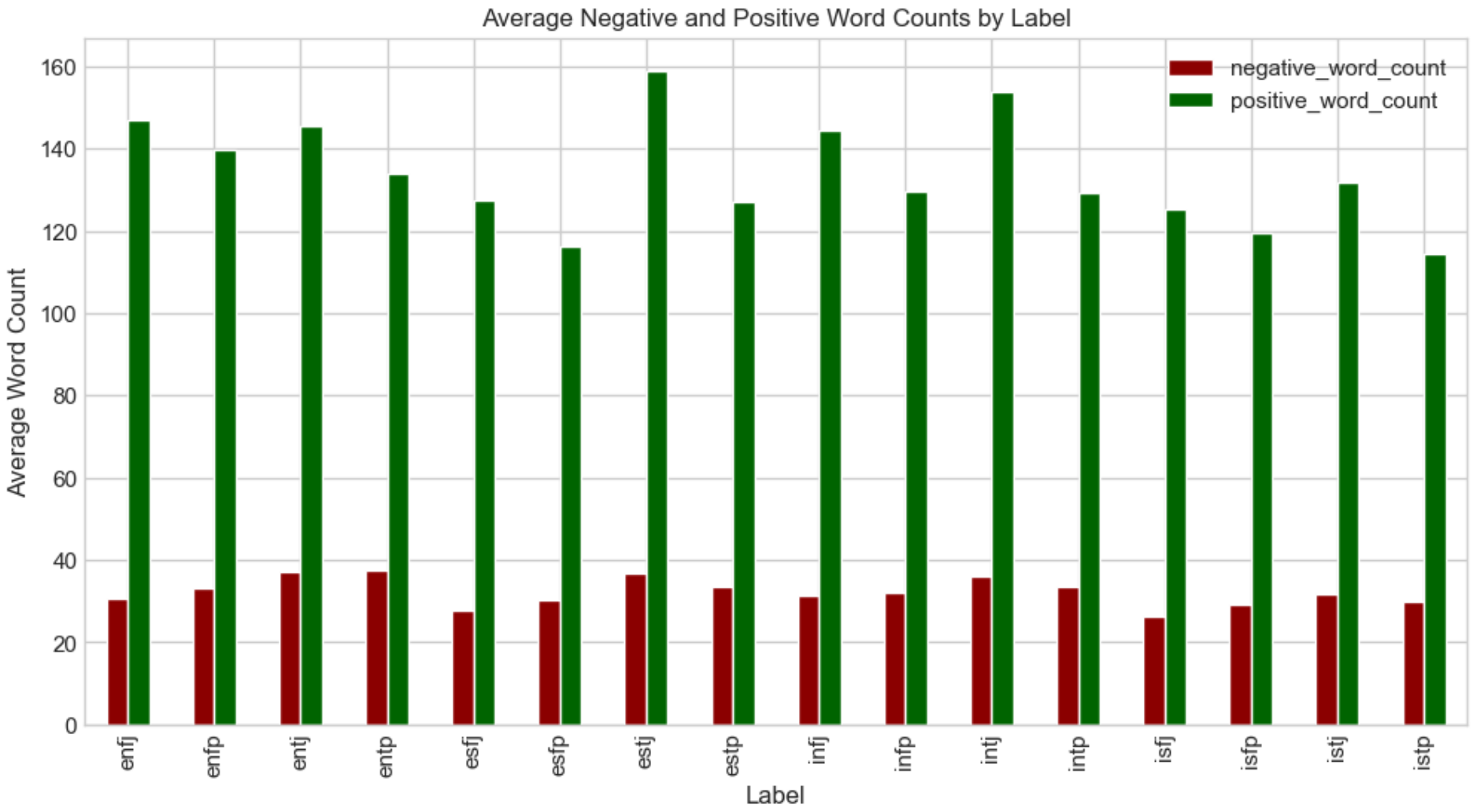
```
number                    int64
text                     object
label                    object
Description              object
word_count                int64
text_length               int64
cumulative_text_length    int64
negative_word_count       int64
negative_words           object
positive_word_count       int64
positive_words           object
dtype: object
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7811 entries, 0 to 7810
Data columns (total 11 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   number                  7811 non-null   int64
 1   text                    7811 non-null   object
 2   label                   7811 non-null   object
 3   Description             7811 non-null   object
 4   word_count              7811 non-null   int64
 5   text_length             7811 non-null   int64
 6   cumulative_text_length  7811 non-null   int64
 7   negative_word_count     7811 non-null   int64
 8   negative_words          7811 non-null   object
 9   positive_word_count     7811 non-null   int64
 10  positive_words          7811 non-null   object
dtypes: int64(6), object(5)
memory usage: 671.4+ KB
None
            number   word_count  text_length  cumulative_text_length  \
count  7811.000000  7811.000000  7811.000000            7.811000e+03
mean   3905.000000  1298.170273  1298.170273            5.110661e+06
std    2254.985809   652.298395   652.298395            2.909564e+06
min       0.000000   201.000000   201.000000            2.351000e+03
25%    1952.500000   834.000000   834.000000            2.591354e+06
50%    3905.000000  1210.000000  1210.000000            5.175169e+06
75%    5857.500000  1679.000000  1679.000000            7.615728e+06
max    7810.000000  4344.000000  4344.000000            1.014001e+07

       negative_word_count  positive_word_count
count          7811.000000          7811.000000
mean             32.529766           135.650365
std              20.951636            79.133755
min               0.000000             1.000000
25%              17.000000            76.000000
50%              29.000000           122.000000
75%              44.000000           180.000000
max             198.000000           602.000000
                        Count of Missing Values
number                                        0
text                                          0
label                                         0
Description                                   0
word_count                                    0
text_length                                   0
cumulative_text_length                        0
negative_word_count                           0
negative_words                                0
positive_word_count                           0
positive_words                                0
```

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Distribution of labels
plt.figure(figsize=(12, 6))
sns.countplot(x='label', data=data, palette='pastel')
plt.title('Distribution of Labels')
plt.xlabel('Label')
plt.ylabel('Count')
plt.show()
```



In [ ]:
```
# Average negative and positive word counts by label
average_word_counts = data.groupby('label')[['negative_word_count', 'positive_word_count']].mean()

print(average_word_counts)

# Plot the average negative and positive word counts by label
average_word_counts.plot(kind='bar', figsize=(12, 6), color=['darkred', 'darkgreen'])
plt.title('Average Negative and Positive Word Counts by Label')
plt.xlabel('Label')
plt.ylabel('Average Word Count')
plt.show()
```

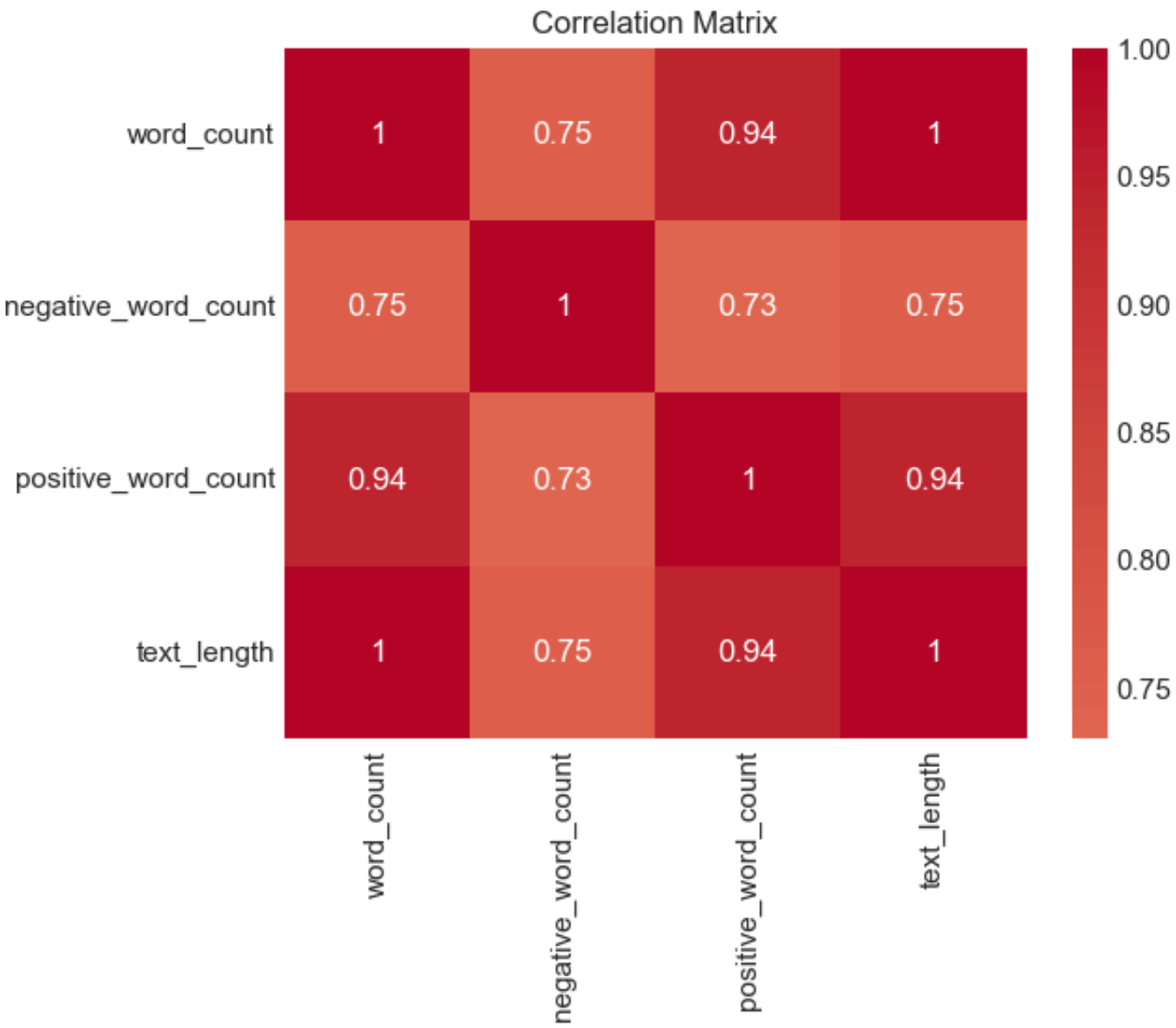
```
       negative_word_count  positive_word_count
label
enfj             30.654440           147.069498
enfp             33.001372           139.884774
entj             36.992832           145.329749
entp             37.325823           133.902946
esfj             27.571429           127.285714
esfp             30.189655           116.149425
estj             36.617284           158.913580
estp             33.690000           127.060000
infj             31.499527           144.302744
infp             32.025741           129.503900
intj             35.910371           153.738796
intp             33.472256           129.315660
isfj             26.406593           125.398352
isfp             29.084469           119.365123
istj             31.633205           131.667954
istp             29.935780           114.354740
```



- Calculate the average negative and positive word counts for each MBTI type (label) in the dataset. Here, data.groupby('label') groups the dataset by the MBTI types (label), and [['negative_word_count', 'positive_word_count']].mean() calculates the mean of the negative and positive word counts for each MBTI type.
- Create a bar plot to visualize the average negative and positive word counts for each MBTI type.
- Set the plot's title, x-axis label, and y-axis label. These lines set the plot's title to "Average Negative and Positive Word Counts by Label", the x-axis label to "Label", and the y-axis label to "Average Word Count".

In [ ]:
```
# Correlations between word counts, negative/positive word counts, and text length
correlations = data[['word_count', 'negative_word_count', 'positive_word_count', 'text_length']].corr()

print(correlations)

# Plot the correlation matrix
sns.heatmap(correlations, annot=True, cmap='coolwarm', center=0)
plt.title('Correlation Matrix')
plt.show()
```

```
                     word_count  negative_word_count  positive_word_count  \
word_count             1.000000             0.751445             0.944844
negative_word_count    0.751445             1.000000             0.729572
positive_word_count    0.944844             0.729572             1.000000
text_length            1.000000             0.751445             0.944844

                     text_length
word_count              1.000000
negative_word_count     0.751445
positive_word_count     0.944844
text_length             1.000000
```



- correlations = data[['word_count', 'negative_word_count', 'positive_word_count', 'text_length']].corr()This line calculates the correlation between the specified columns (word_count, negative_word_count, positive_word_count, and text_length) in the dataset. The corr() function computes the pairwise correlation of columns, excluding null values.
- print(correlations)

This line prints the correlation matrix, which shows the correlation coefficients between each pair of features. The correlation coefficient ranges from -1 to 1. A value close to 1 means that the two features have a strong positive correlation, a value close to -1 indicates a strong negative correlation, and a value close to 0 means there is little or no correlation between the features.

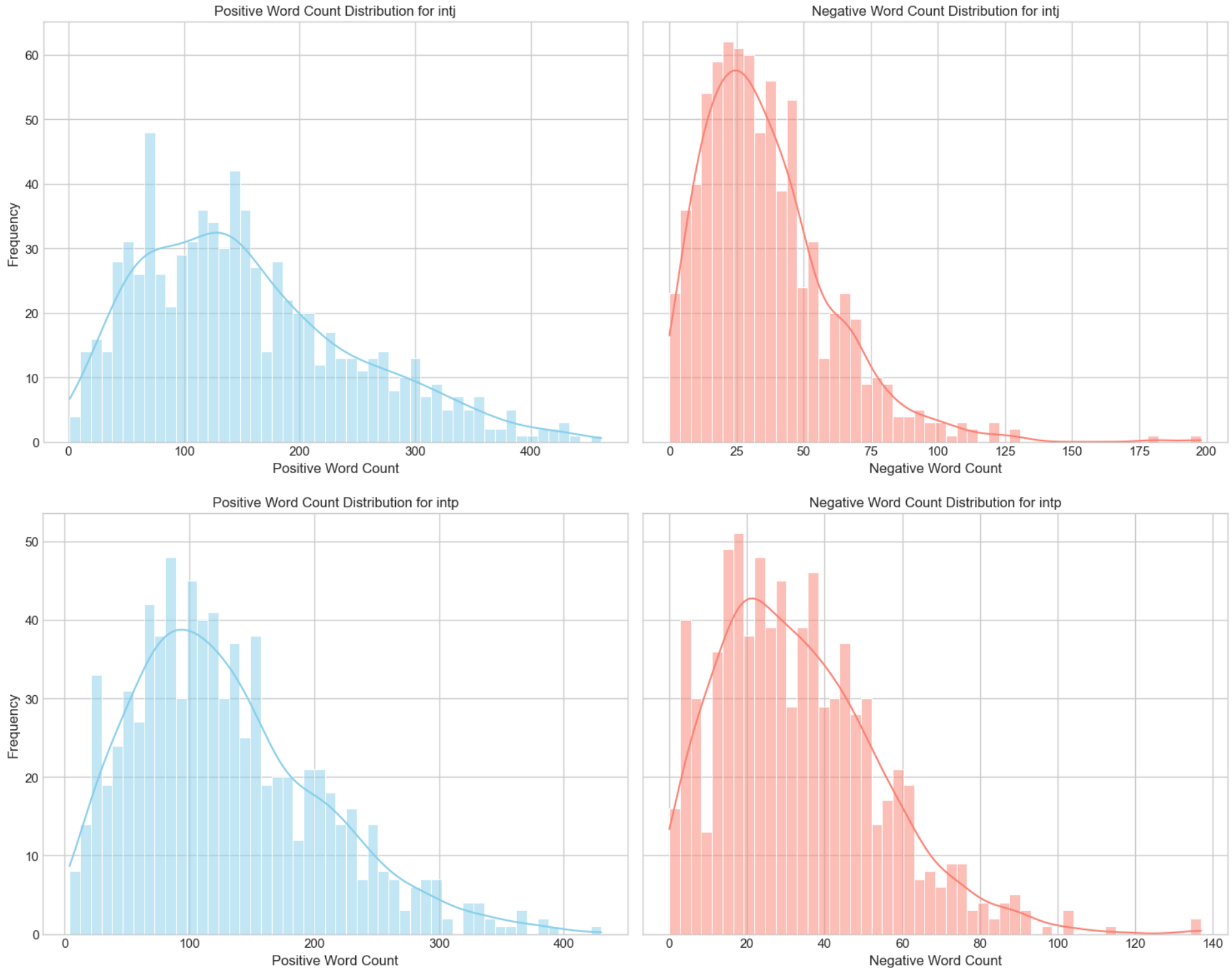
- sns.heatmap(correlations, annot=True, cmap='coolwarm', center=0)

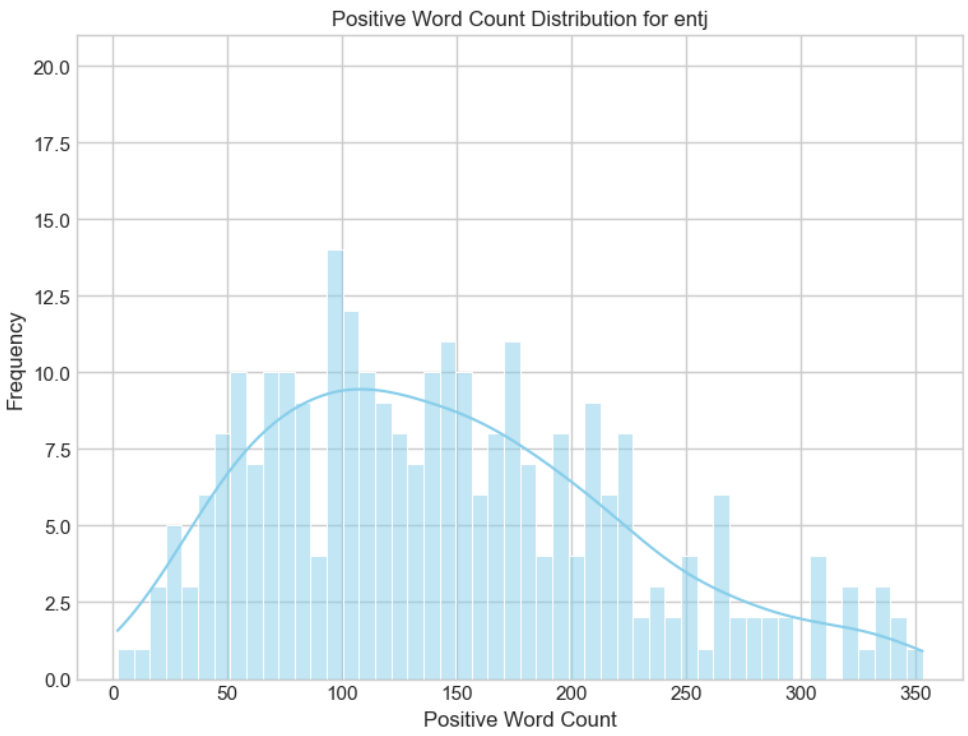
This line creates a heatmap using the Seaborn library to visualize the correlation matrix. The annot=True parameter adds the correlation coefficients to each cell in the heatmap. The cmap='coolwarm' parameter sets the color palette for the heatmap to a blue-to-red gradient, with blue representing negative correlations and red representing positive correlations. The center=0 parameter ensures that the color scale is centered around 0, making it easier to distinguish between positive and negative correlations.

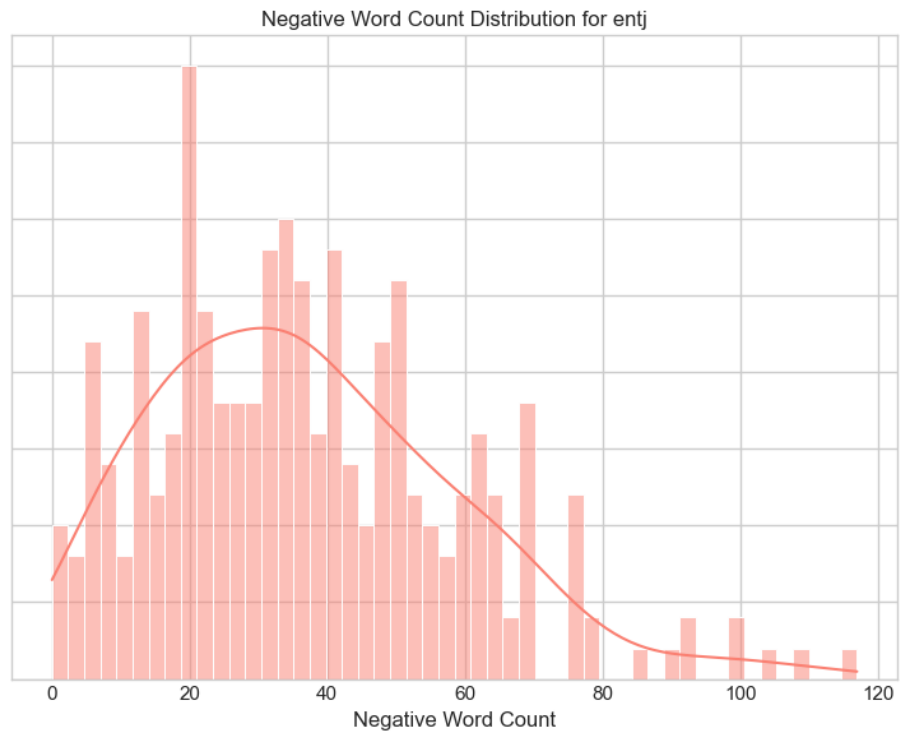
- plt.title('Correlation Matrix')

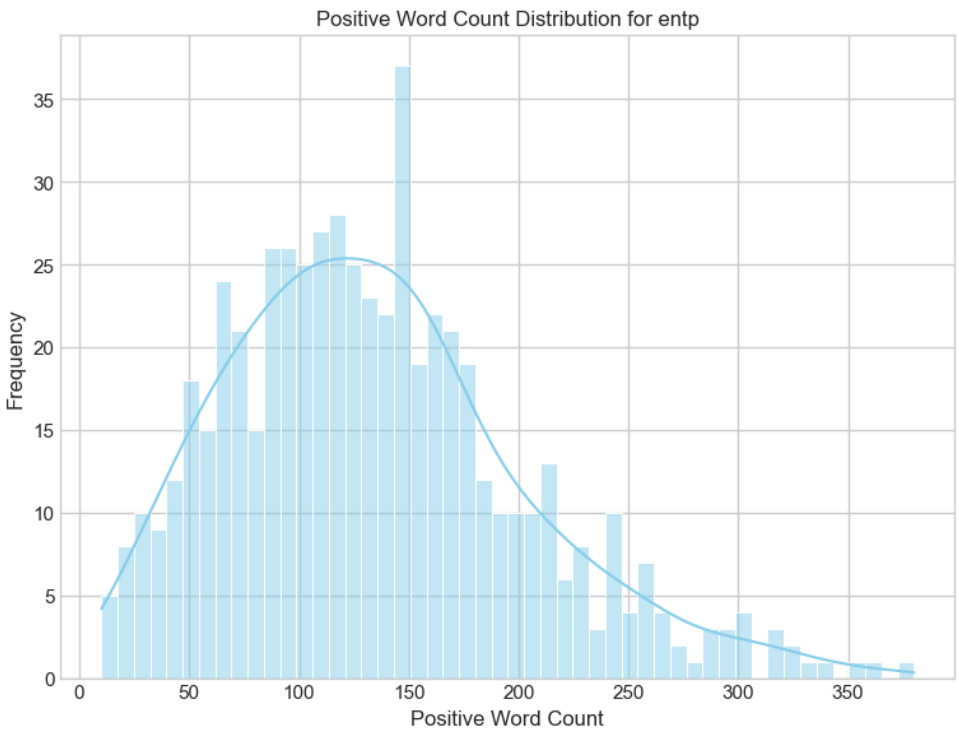
This line adds a title to the heatmap plot, labeling it as 'Correlation Matrix'.

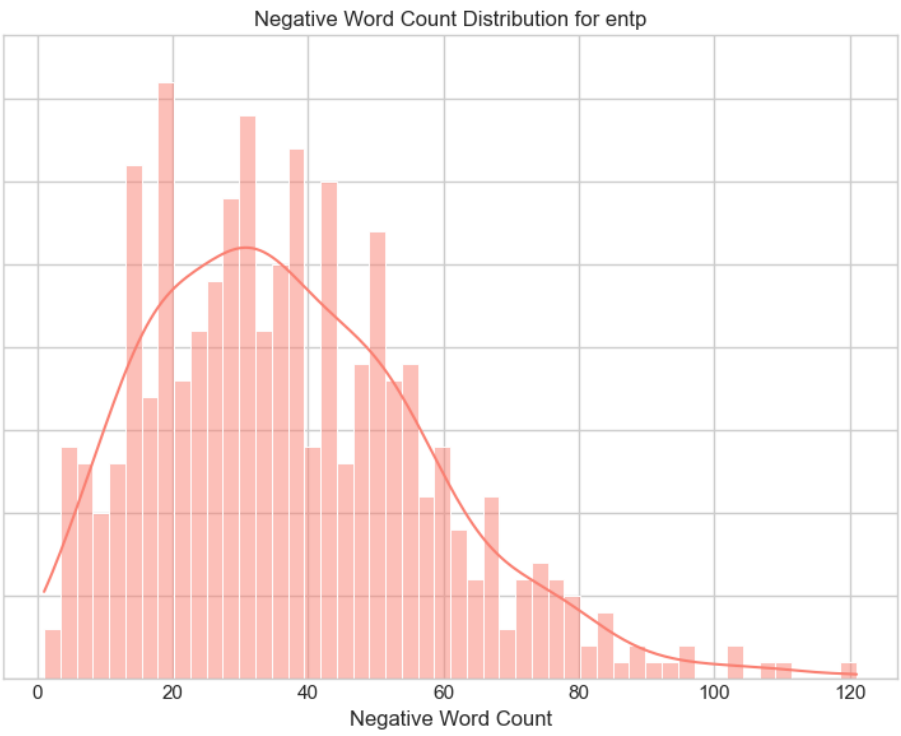
In [ ]:
```
import seaborn as sns

def plot_word_count_distribution_side_by_side(data, label, color_dict):
    fig, axes = plt.subplots(1, 2, figsize=(15, 6), sharey=True)

    sns.histplot(data=data[data['label'] == label], x='positive_word_count', kde=True, color=color_dict['positive
    axes[0].set_title(f'Positive Word Count Distribution for {label}')
    axes[0].set_xlabel('Positive Word Count')
    axes[0].set_ylabel('Frequency')

    sns.histplot(data=data[data['label'] == label], x='negative_word_count', kde=True, color=color_dict['negative
    axes[1].set_title(f'Negative Word Count Distribution for {label}')
    axes[1].set_xlabel('Negative Word Count')
    axes[1].set_ylabel('Frequency')

    plt.tight_layout()
    plt.show()

labels = data['label'].unique()
color_dict = {'positive': 'skyblue', 'negative': 'salmon'}

for label in labels:
    plot_word_count_distribution_side_by_side(data, label, color_dict)
```

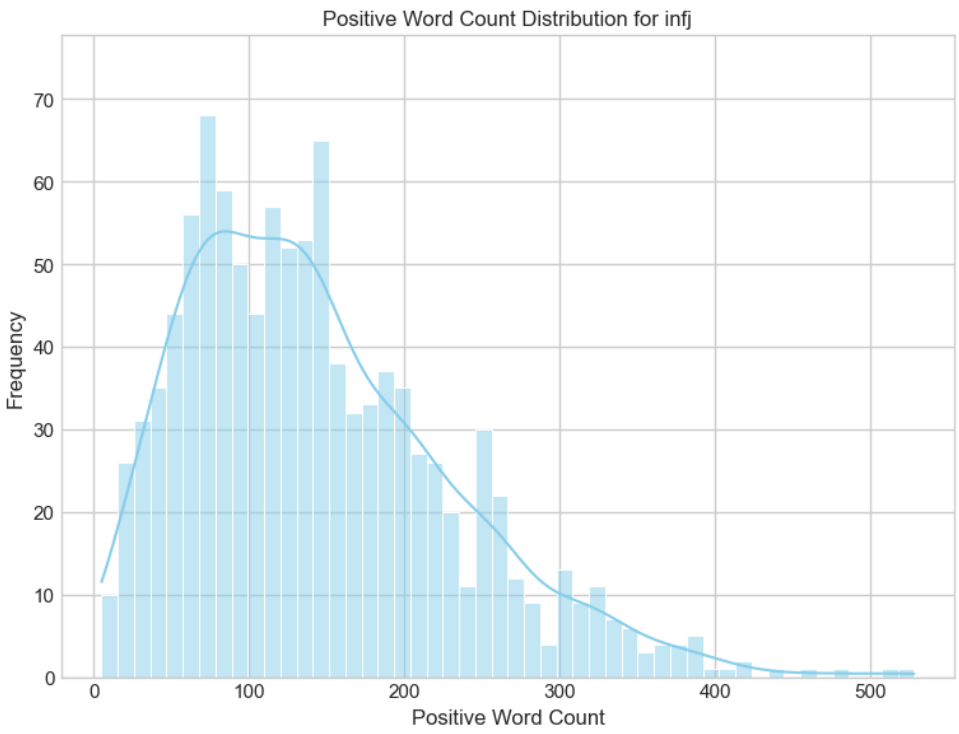


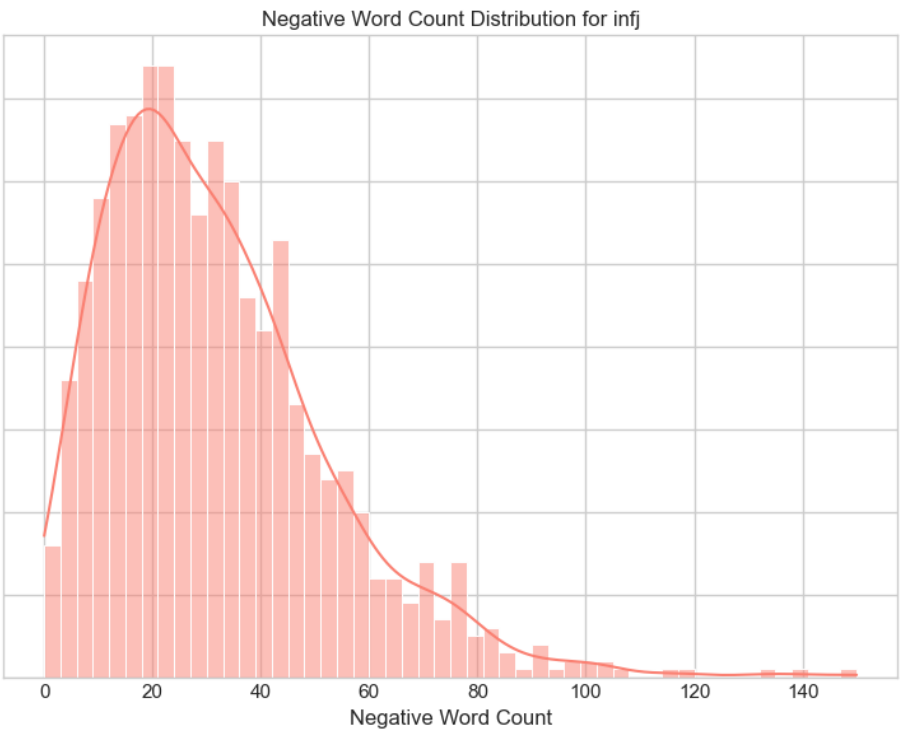

Positive Word Count Distribution for entj
Frequency
Positive Word Count

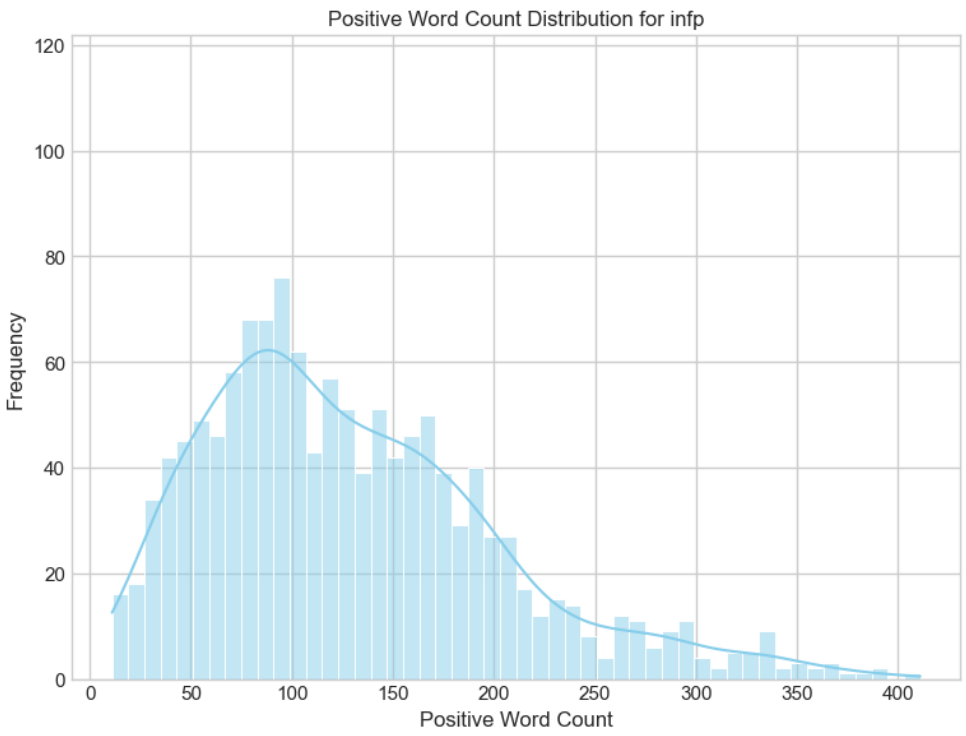

Negative Word Count Distribution for entj
Negative Word Count

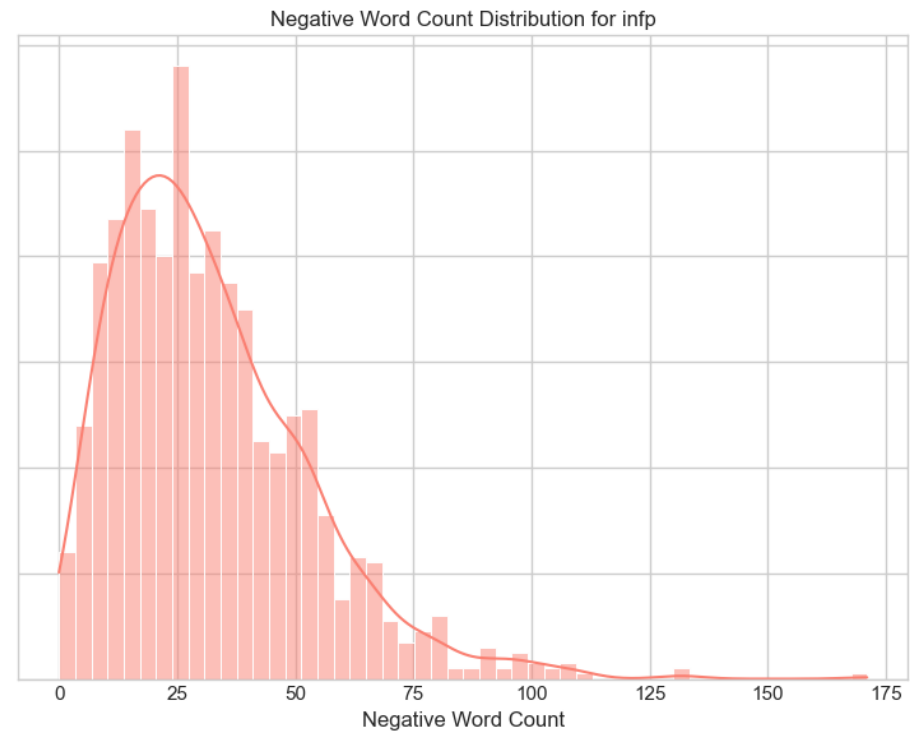

Positive Word Count Distribution for entp
Frequency
Positive Word Count

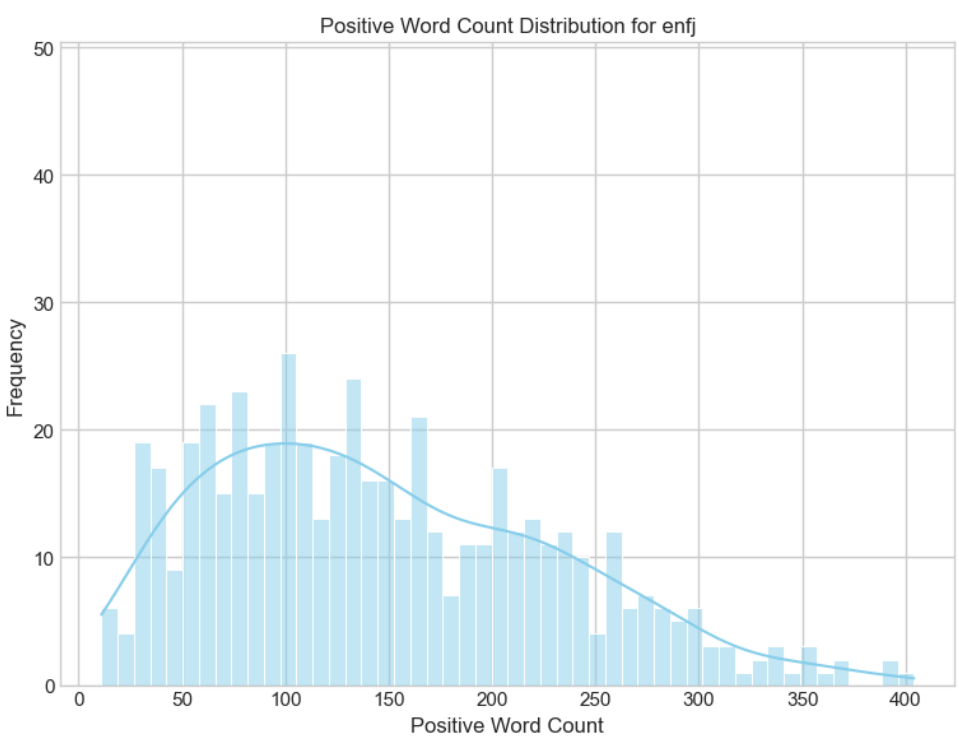

Negative Word Count Distribution for entp
Negative Word Count

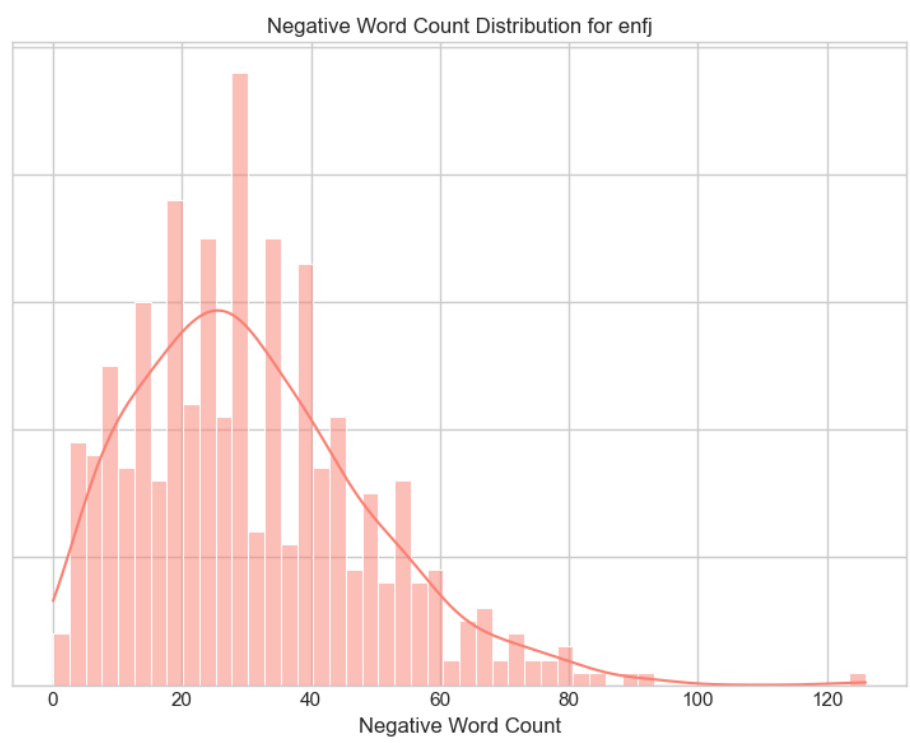

Positive Word Count Distribution for infj
Frequency
Positive Word Count

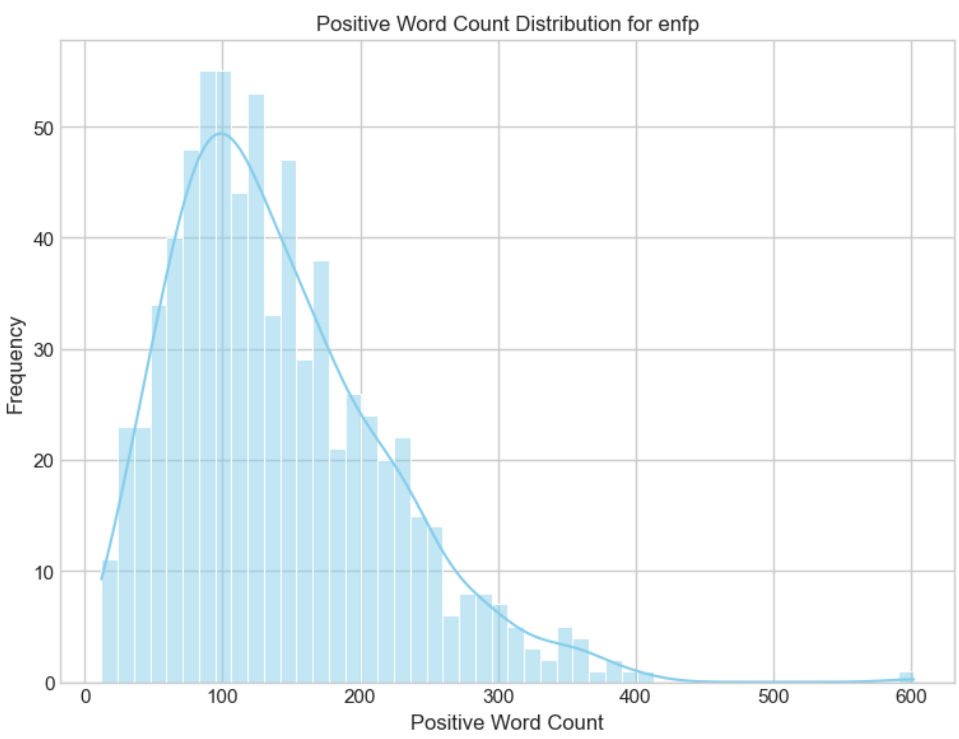

Negative Word Count Distribution for infj
Negative Word Count

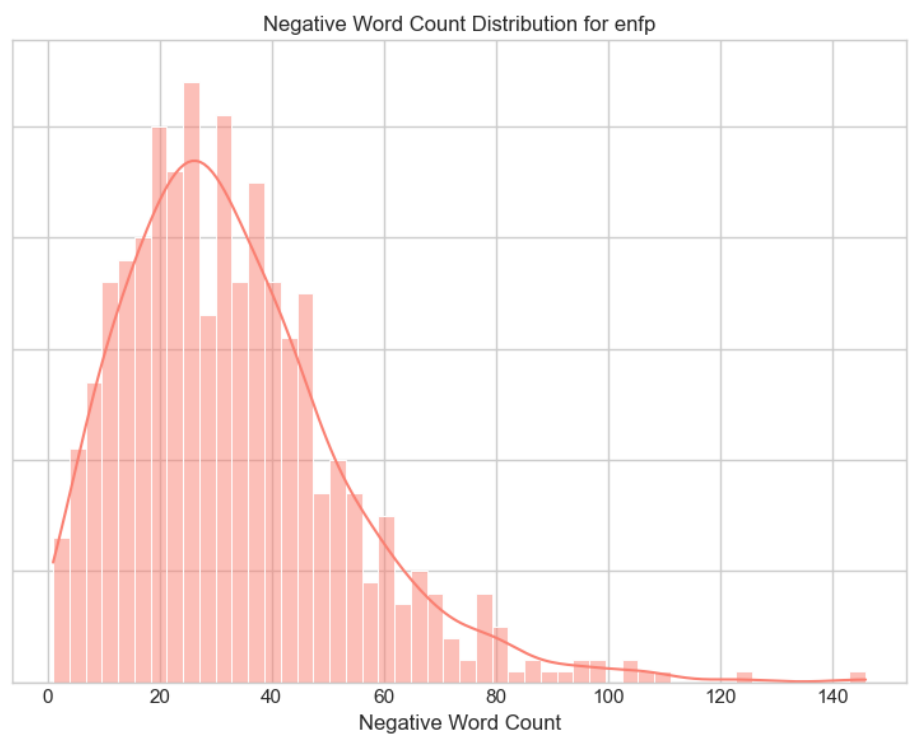

Positive Word Count Distribution for infp
Frequency
Positive Word Count

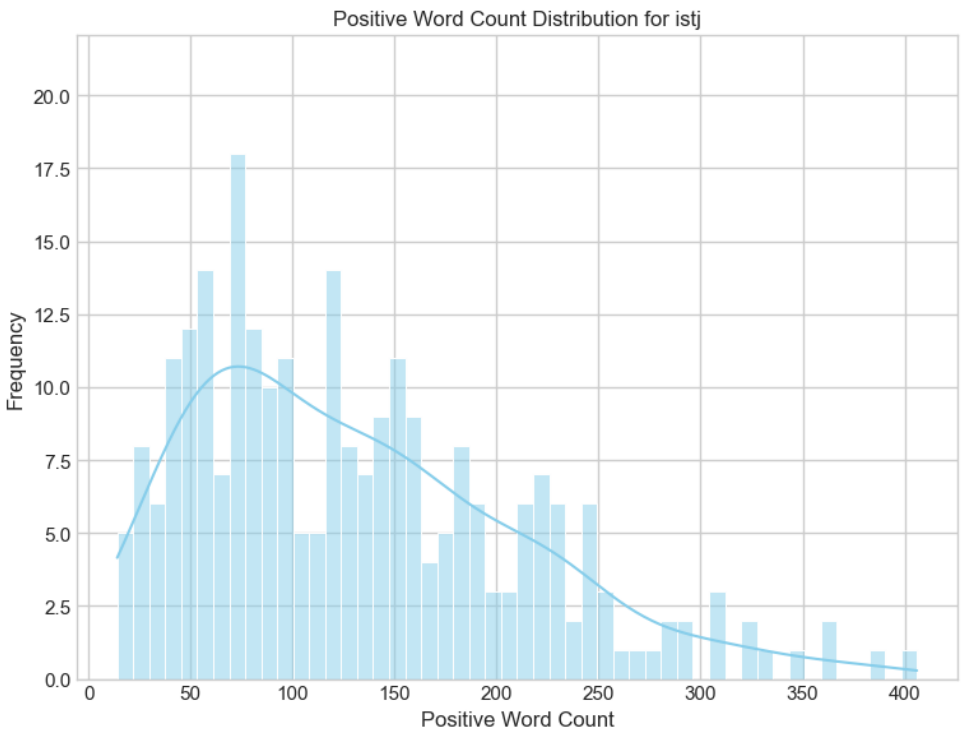

Negative Word Count Distribution for infp
Negative Word Count

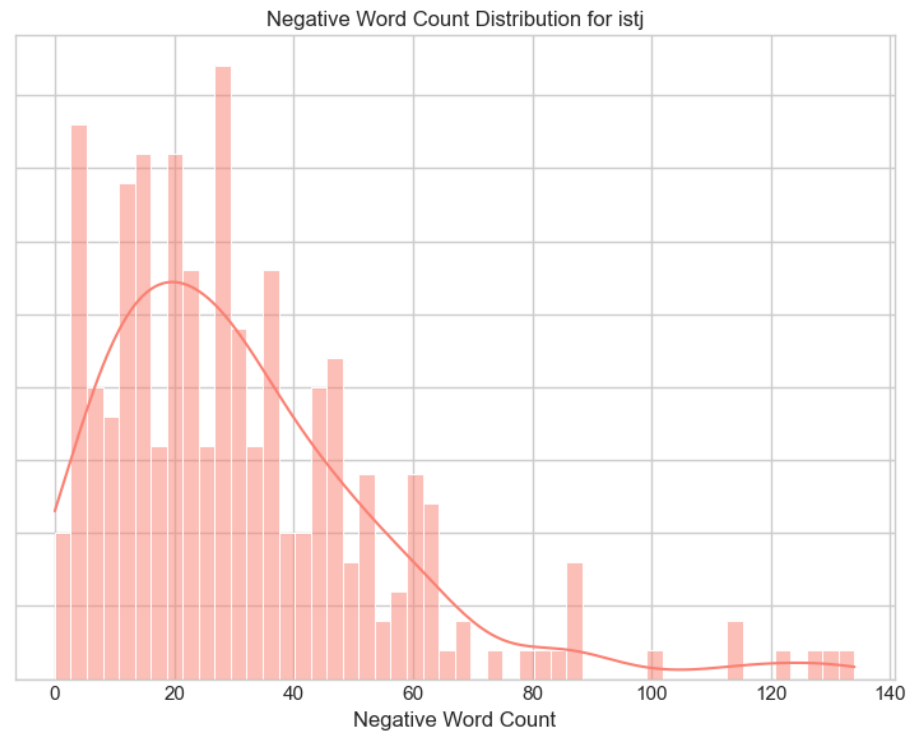

Positive Word Count Distribution for enfj
Frequency
Positive Word Count

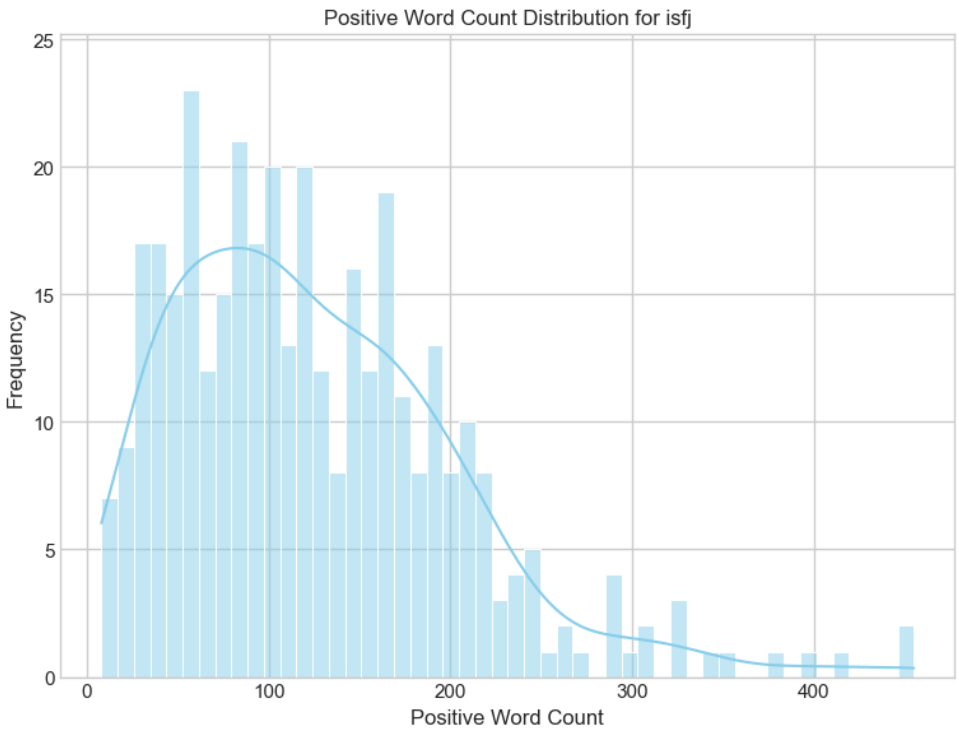

Negative Word Count Distribution for enfj
Negative Word Count

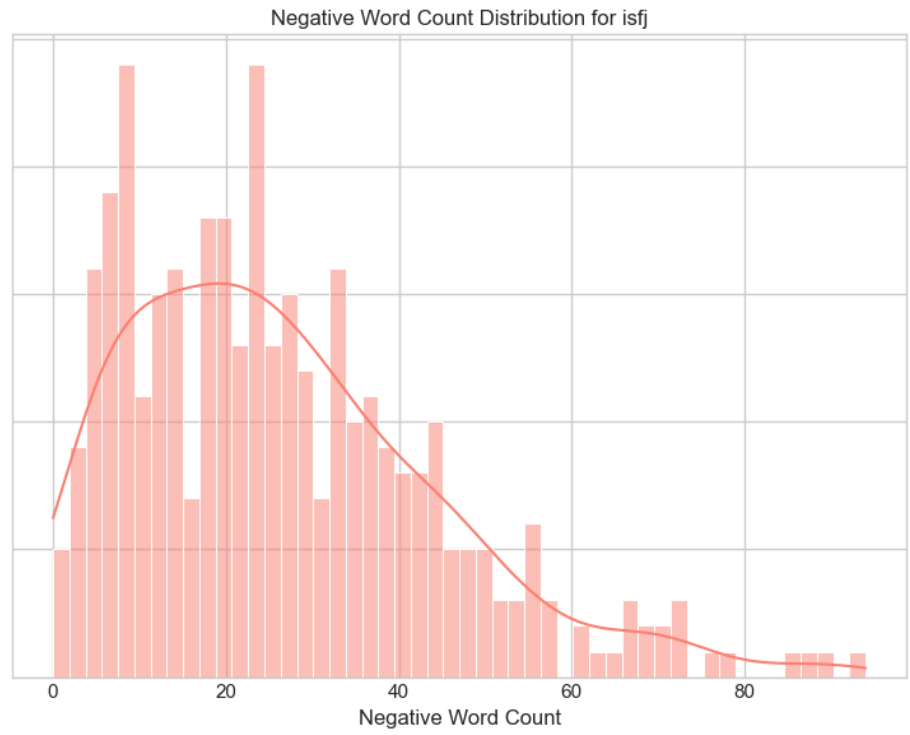

Positive Word Count Distribution for enfp
Frequency
Positive Word Count

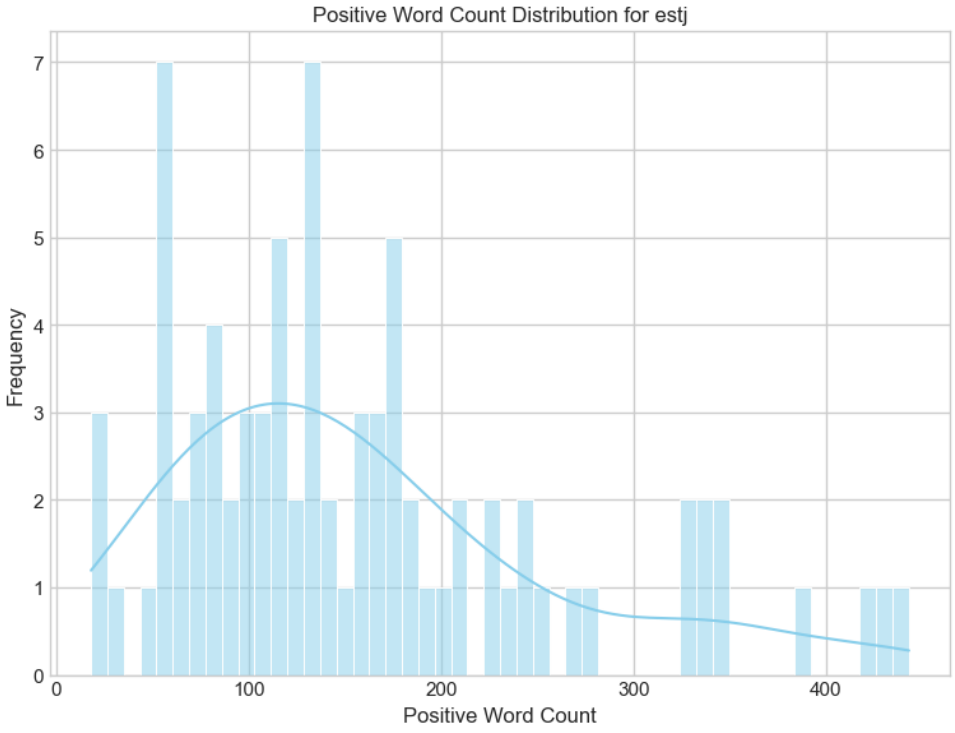

Negative Word Count Distribution for enfp
Negative Word Count

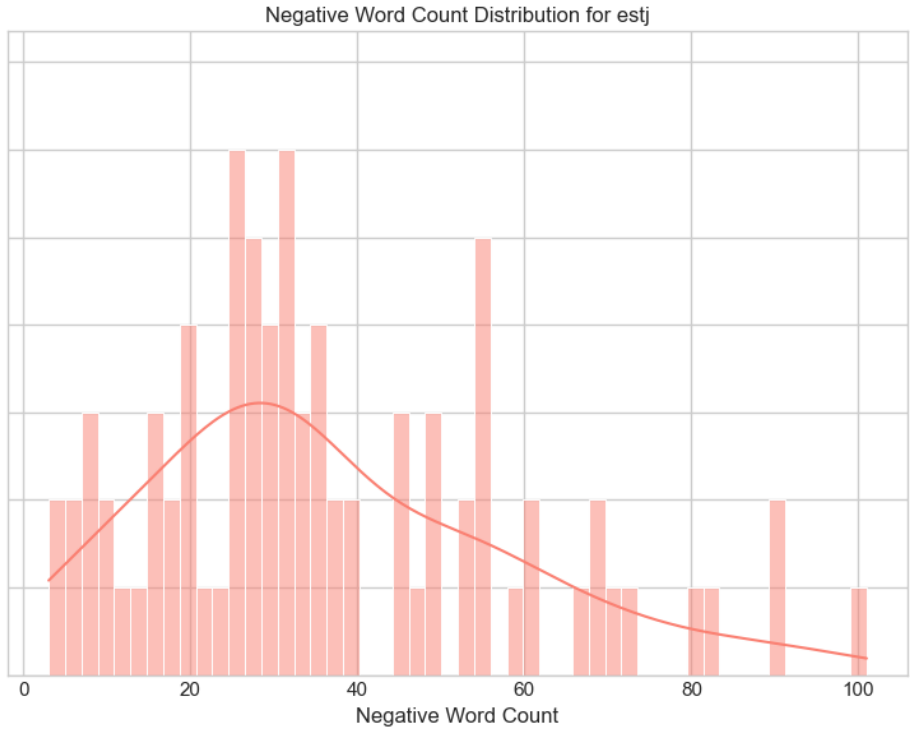

Positive Word Count Distribution for istj
Frequency
Positive Word Count

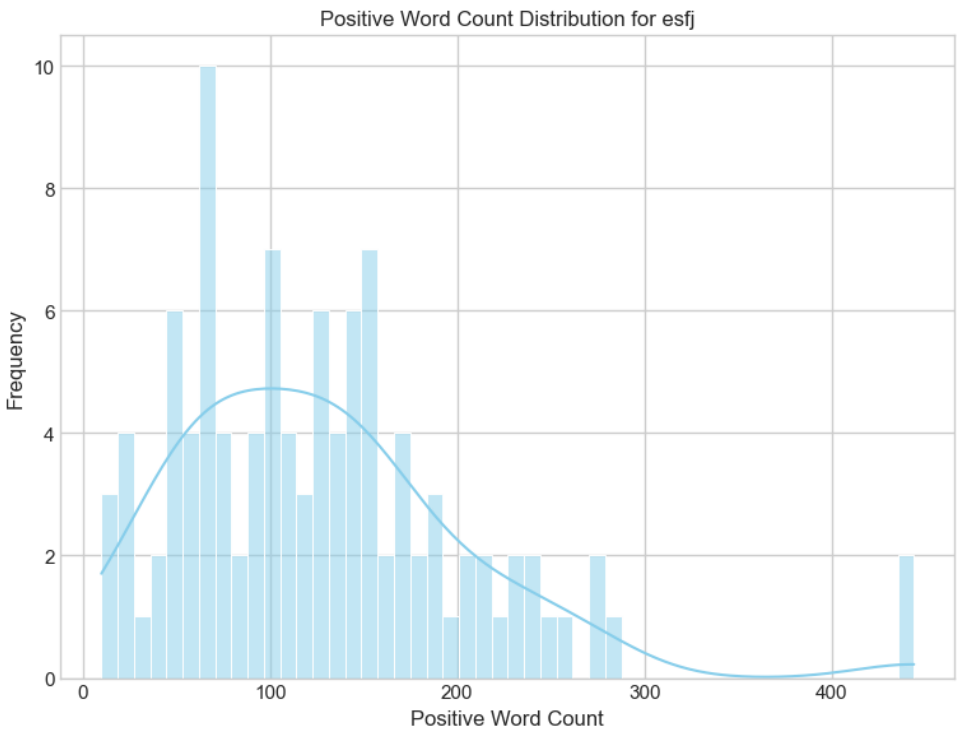

Negative Word Count Distribution for istj
Negative Word Count

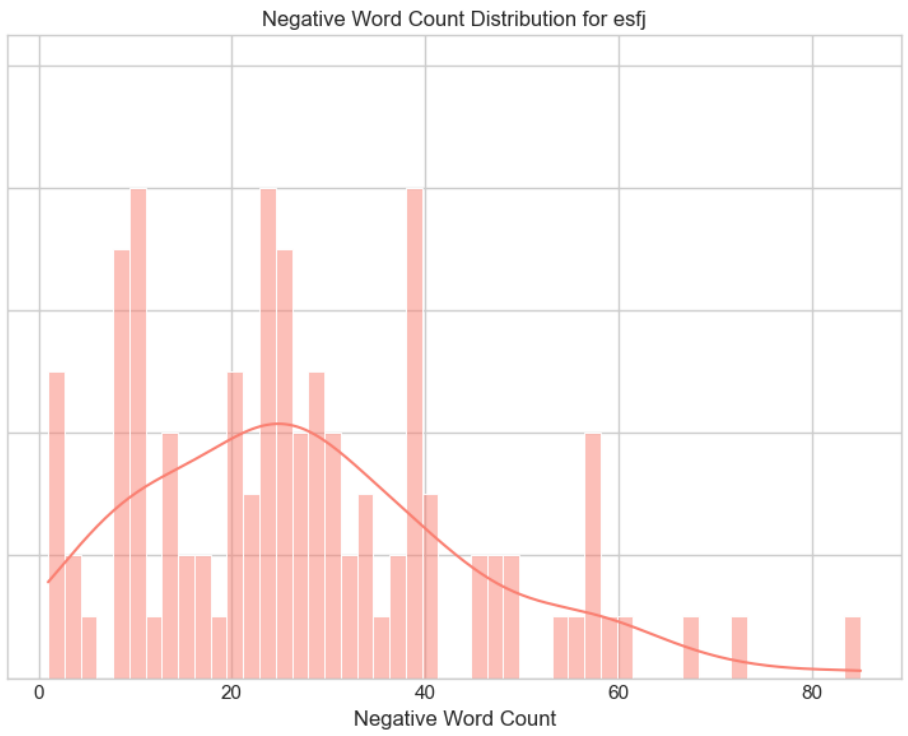

Positive Word Count Distribution for isfj
Frequency
Positive Word Count

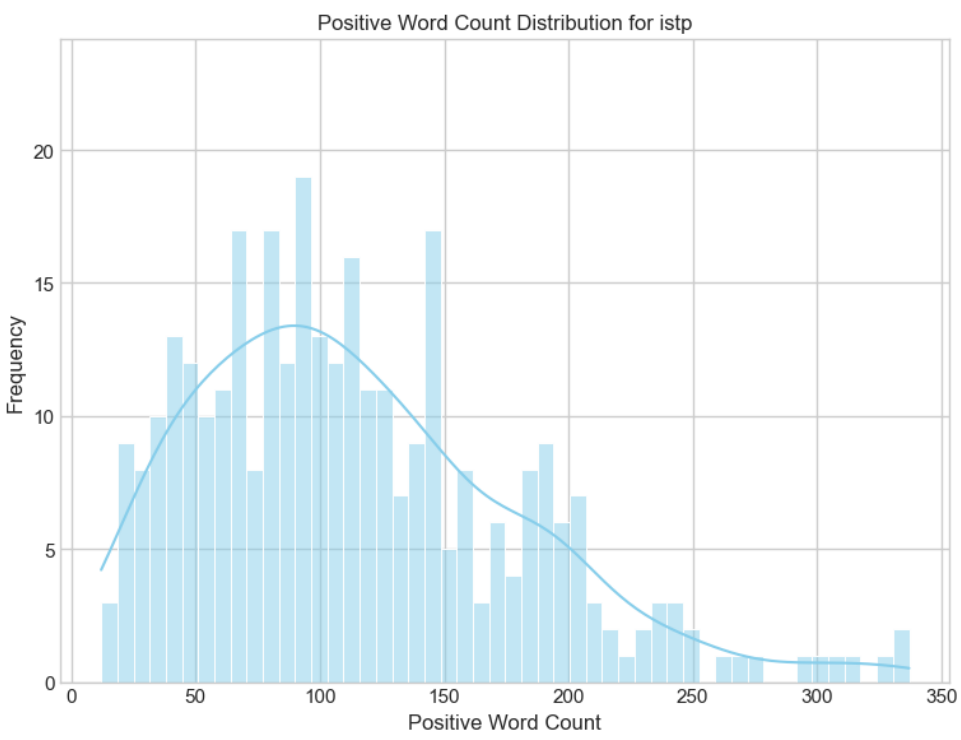

Negative Word Count Distribution for isfj
Negative Word Count

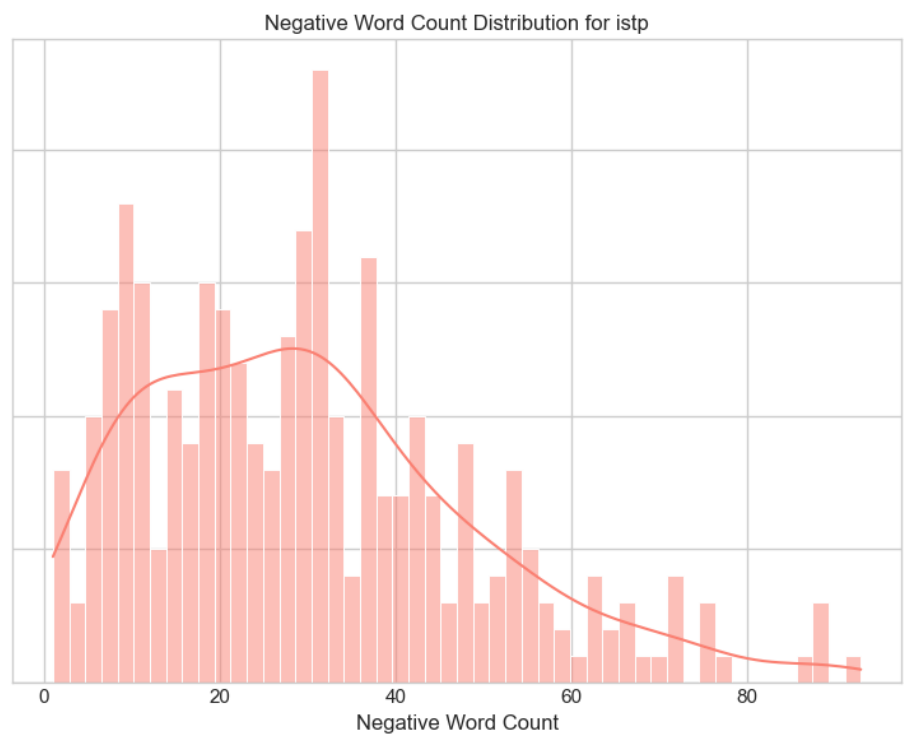

Positive Word Count Distribution for estj
Frequency
Positive Word Count

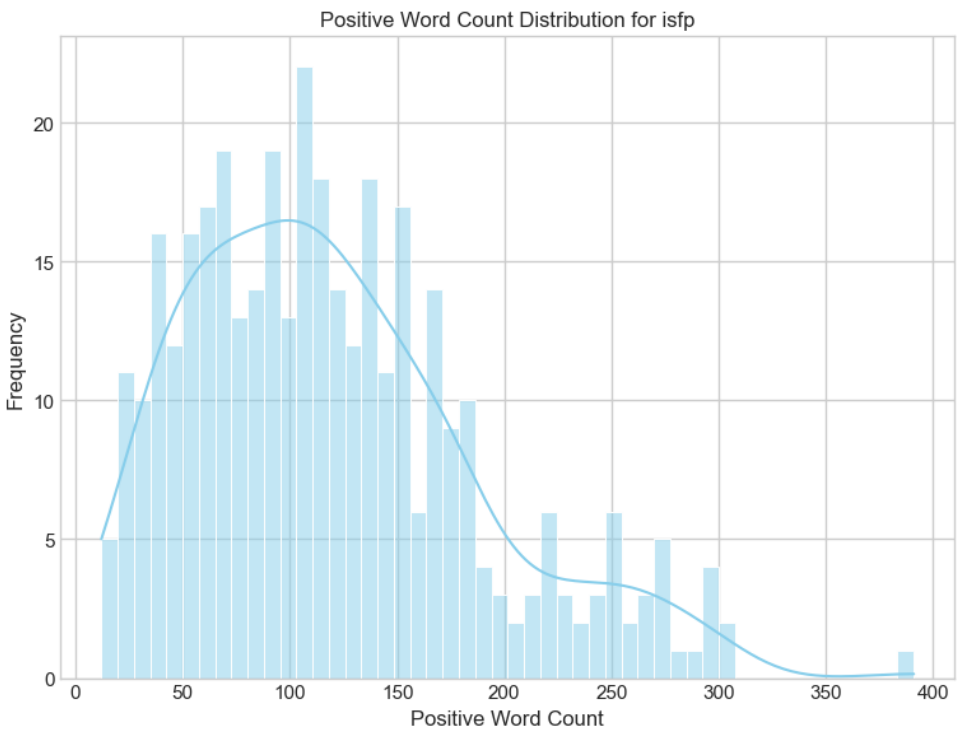

Negative Word Count Distribution for estj
Negative Word Count

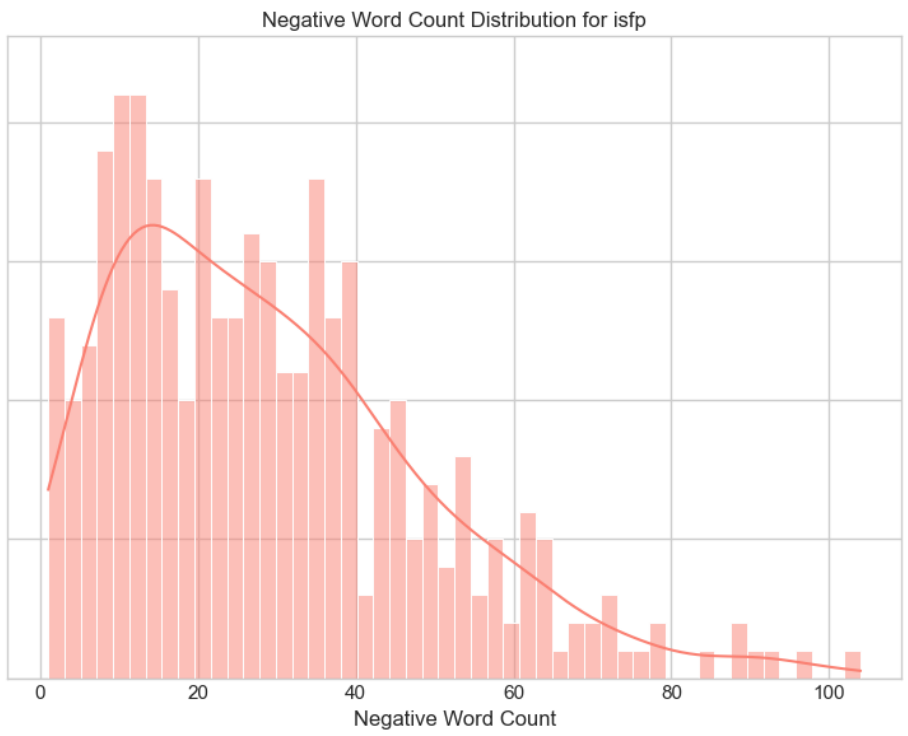

Positive Word Count Distribution for esfj
Frequency
Positive Word Count
0
2
4
6
8
10
0
100
200
300
400

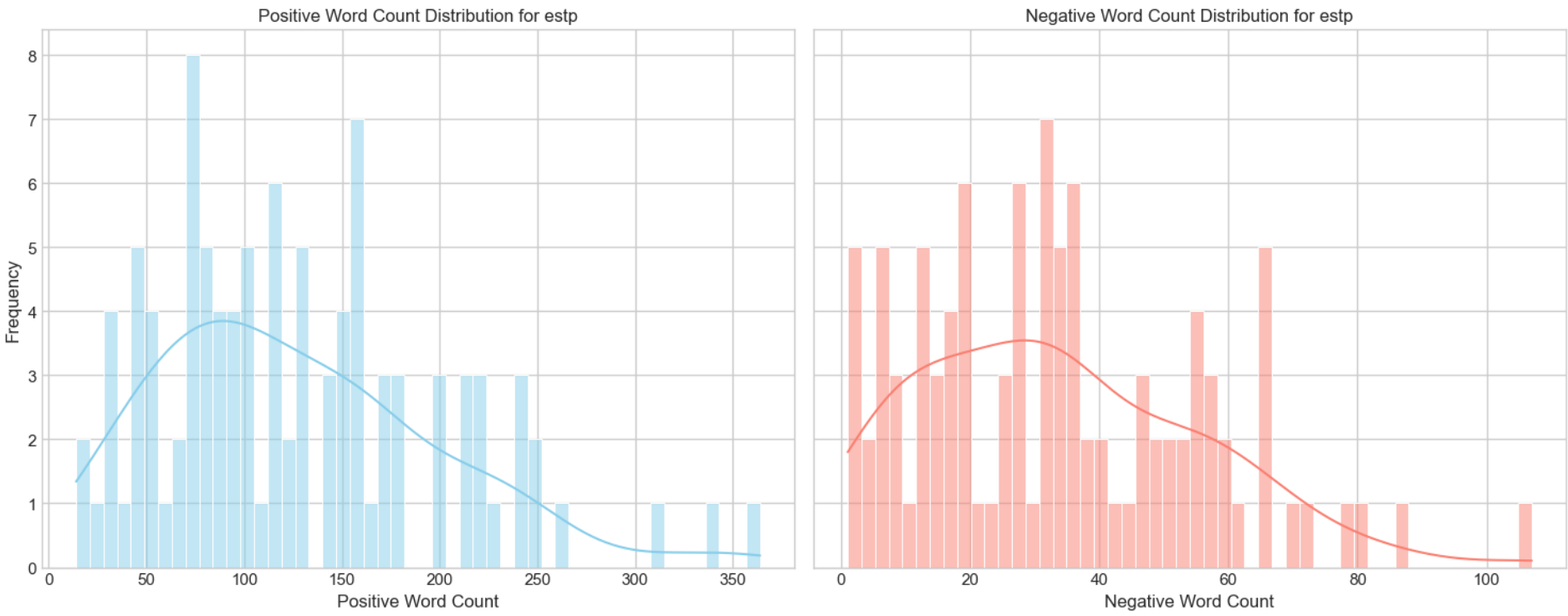

Negative Word Count Distribution for esfj
Negative Word Count
0
20
40
60
80

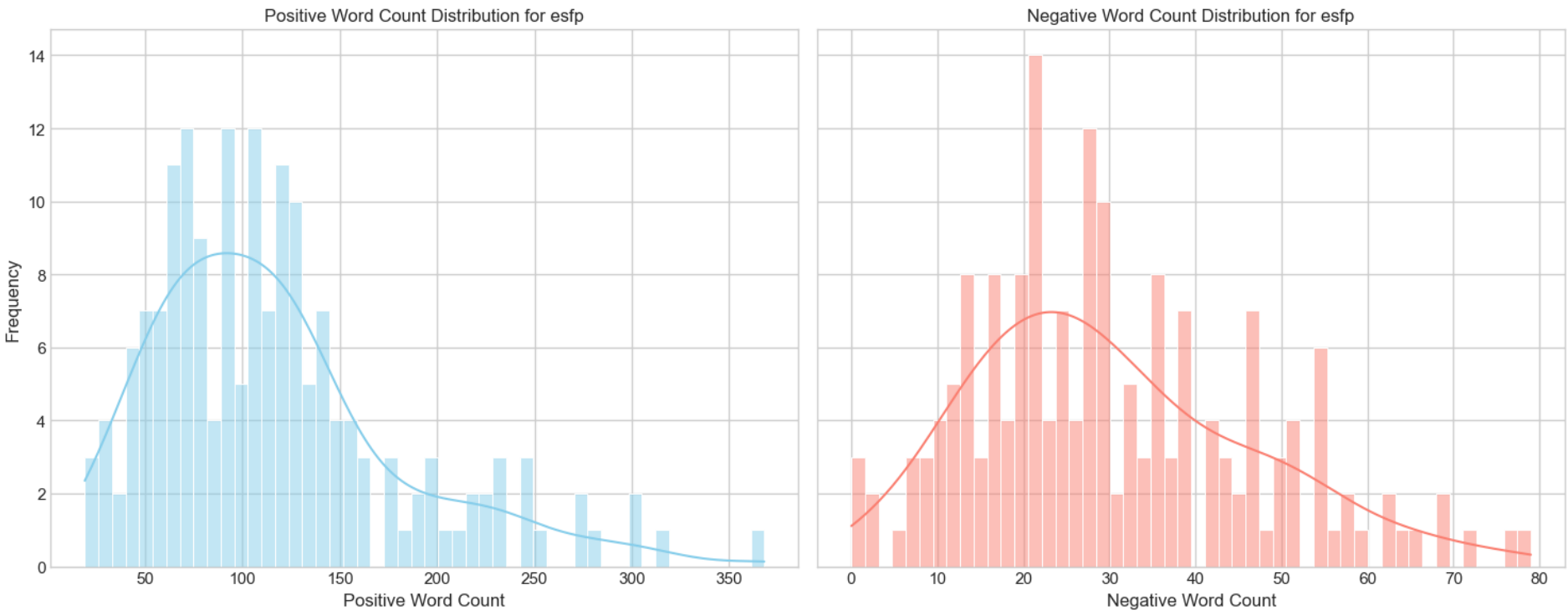

Positive Word Count Distribution for istp
Frequency
Positive Word Count
0
5
10
15
20
0
50
100
150
200
250
300
350


Negative Word Count Distribution for istp
Negative Word Count
0
20
40
60
80


Positive Word Count Distribution for isfp
Frequency
Positive Word Count
0
5
10
15
20
0
50
100
150
200
250
300
350
400


Negative Word Count Distribution for isfp
Negative Word Count
0
20
40
60
80
100





- Define the function plot_word_count_distribution_side_by_side() with input parameters data, label, and color_dict
- Create a 1x2 subplot figure with shared y-axis using plt.subplots(). This will create two side-by-side plots.
- Create a histogram plot of the positive word count distribution for the given label (MBTI type) using the seaborn sns.histplot() function. Set the plot title, x-axis label, and y-axis label.
- Similarly, create a histogram plot of the negative word count distribution for the given label (MBTI type). Set the plot title, x-axis label, and y-axis label.
- Define labels as the unique MBTI types found in the dataset and color_dict to store the colors for positive and negative word count plots.
- Iterate over each label (MBTI type) and call the plot_word_count_distribution_side_by_side() function to generate the side-by-side histograms for each MBTI type.