```
from google.colab import files

# Upload the zip file
uploaded = files.upload()
```

Choose Files 1.zip

- **1.zip**(application/x-zip-compressed) - 93586 bytes, last modified: 9/23/2024 - 100% done

```
Saving 1.zip to 1.zip
```

```
import zipfile
import os

# Assuming the uploaded file name is 'data.zip'
zip_file_path = '1.zip'

# Create a directory to extract the files
extraction_path = '/content/data'
os.makedirs(extraction_path, exist_ok=True)

# Extract the zip file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extraction_path)

# List the extracted files
extracted_files = os.listdir(extraction_path)
print(extracted_files)
```

```
['emotion_data', 'transcript_data', 'transcripts']
```

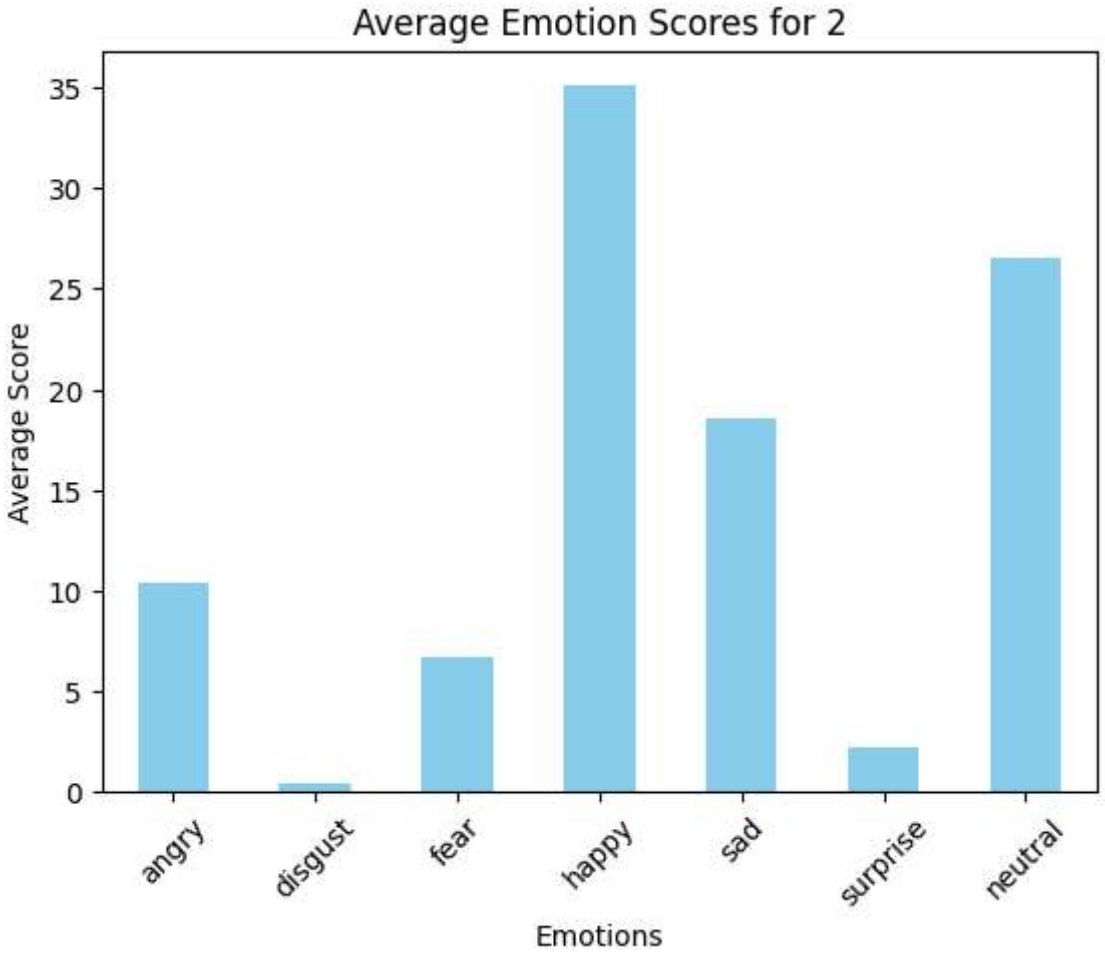
```
import os
import pandas as pd

# Define the path to the Emotion folder
emotion_data_path = os.path.join(extraction_path, 'emotion_data')

# Load emotion, gaze, and metadata for each candidate
```

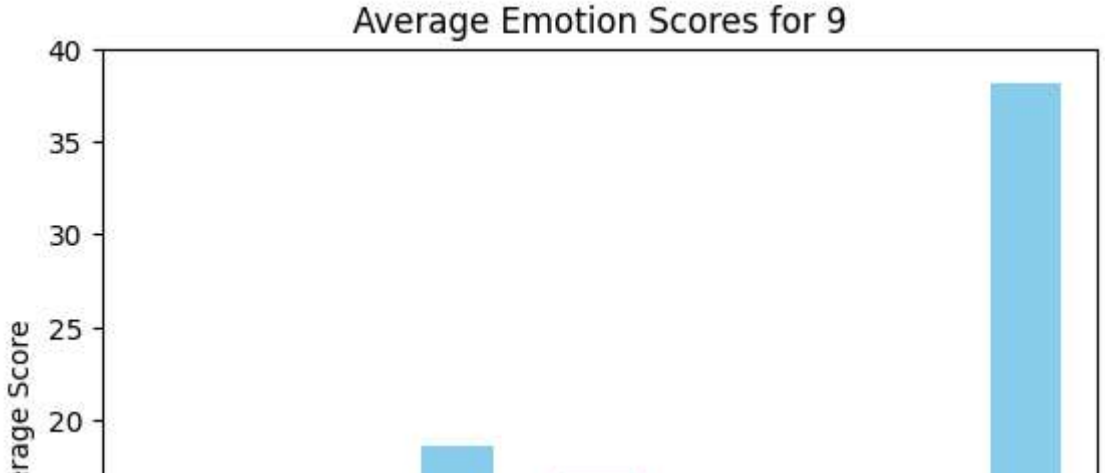
```
emotion_dfs = {}
gaze_dfs = {}
metadata_dfs = {}

for candidate_folder in os.listdir('/content/data/emotion_data'):
    candidate_path = os.path.join('/content/data/emotion_data', candidate_folder)
    if os.path.isdir(candidate_path):
        # Load emotion, gaze, and metadata data from CSV files
        emotion_file = os.path.join(candidate_path, 'emotion.csv')  # Adjust filename
        gaze_file = os.path.join(candidate_path, 'gaze.csv')        # Adjust filename
        metadata_file = os.path.join(candidate_path, 'metadata.csv')    # Adjust filename

        # Read the CSV files into DataFrames
        emotion_dfs[candidate_folder] = pd.read_csv(emotion_file)
        gaze_dfs[candidate_folder] = pd.read_csv(gaze_file)
        metadata_dfs[candidate_folder] = pd.read_csv(metadata_file)

# Display a sample of emotion data for one candidate
print(emotion_dfs[list(emotion_dfs.keys())[0]].head())
```

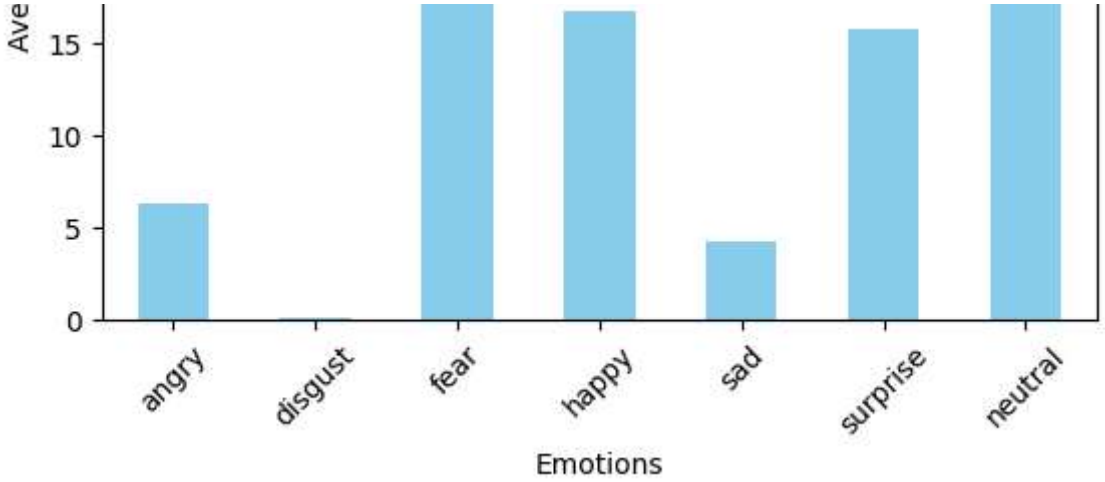
```
                              movie_id  image_seq      angry   disgust  \
0  baa26895-85b2-465b-a972-649b41d9870e          0   4.903760  0.000024
1  baa26895-85b2-465b-a972-649b41d9870e          1   0.179621  0.000185
2  baa26895-85b2-465b-a972-649b41d9870e          2  10.126300  0.087004
3  baa26895-85b2-465b-a972-649b41d9870e          3  37.344900  0.427457
4  baa26895-85b2-465b-a972-649b41d9870e          4   0.003088  0.000003

       fear     happy       sad   surprise    neutral dominant_emotion
0  1.847580   2.55923  48.79130   0.033327  41.864800              sad
1  0.055258  93.56640   6.18999   0.001184   0.007402            happy
2  6.057070  42.70380  19.81920  15.360900   5.845700            happy
3  2.784040  16.53680  35.73190   0.534506   6.640390            angry
4  0.002681  98.51810   1.47585   0.000055   0.000212            happy
```

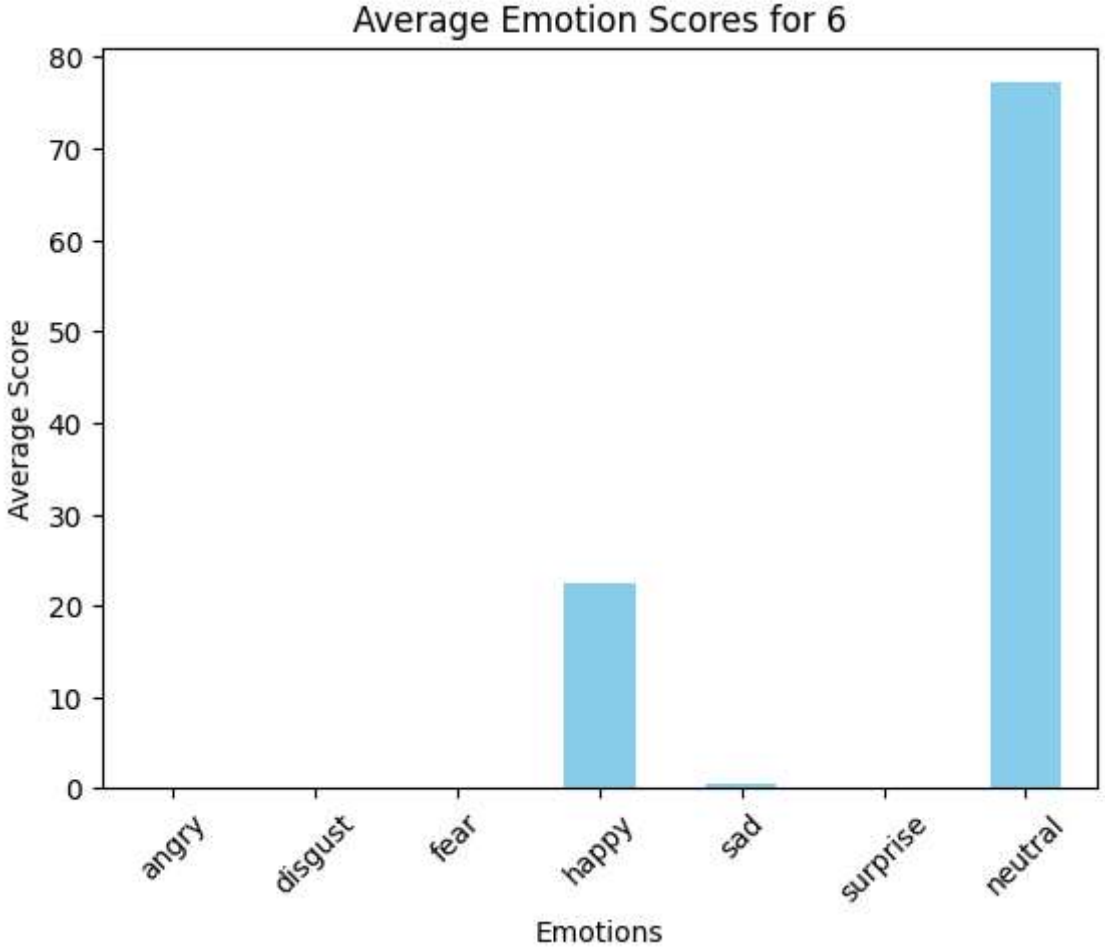
```
import matplotlib.pyplot as plt
import seaborn as sns

# Function to plot emotion distributions
def plot_emotion_distribution(emotion_df, candidate_id):
```

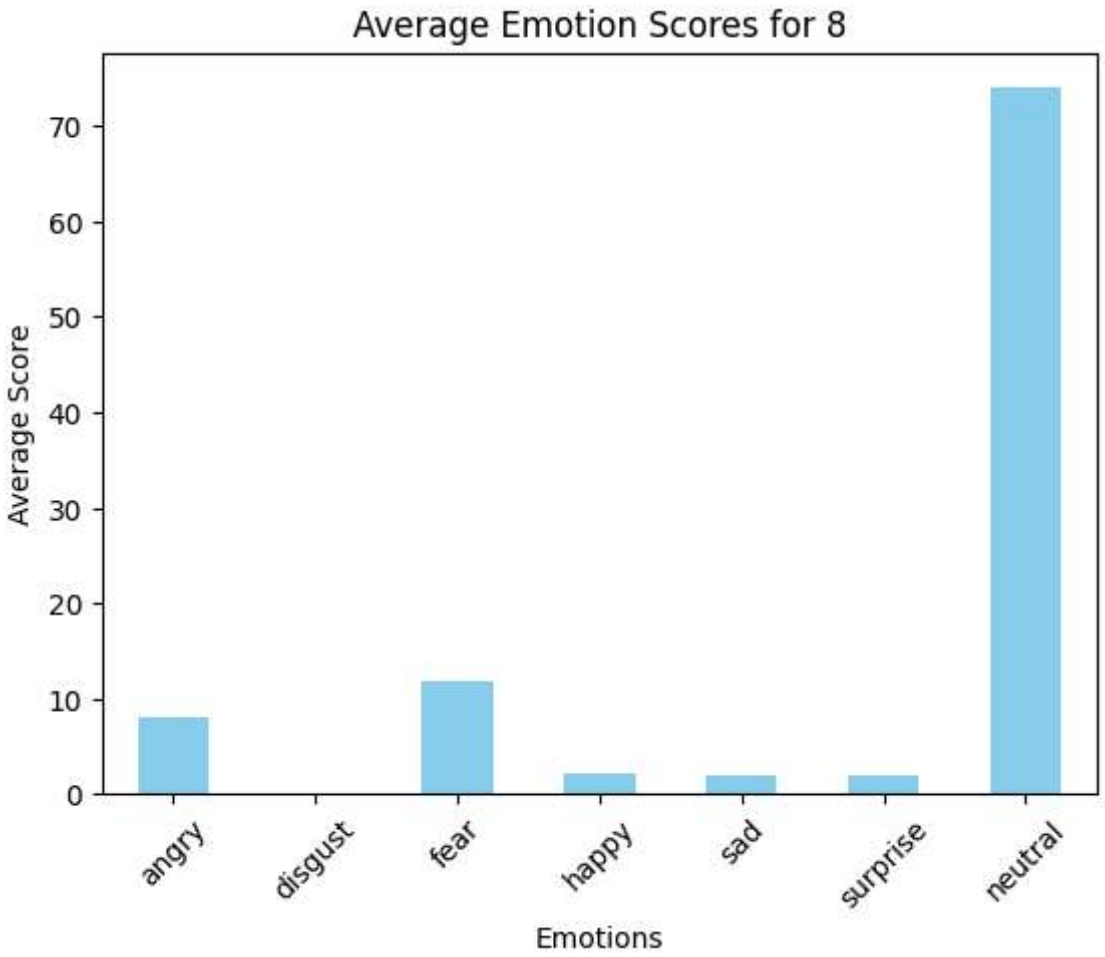
```
        emotion_cols = ['angry', 'disgust', 'fear', 'happy', 'sad', 'surprise', 'neutral']
        emotion_df[emotion_cols].mean().plot(kind='bar', color='skyblue')
        plt.title(f'Average Emotion Scores for {candidate_id}')
        plt.ylabel('Average Score')
        plt.xlabel('Emotions')
        plt.xticks(rotation=45)
        plt.show()

    # Analyze for each candidate
    for candidate_id, df in emotion_dfs.items():
        plot_emotion_distribution(df, candidate_id)
```

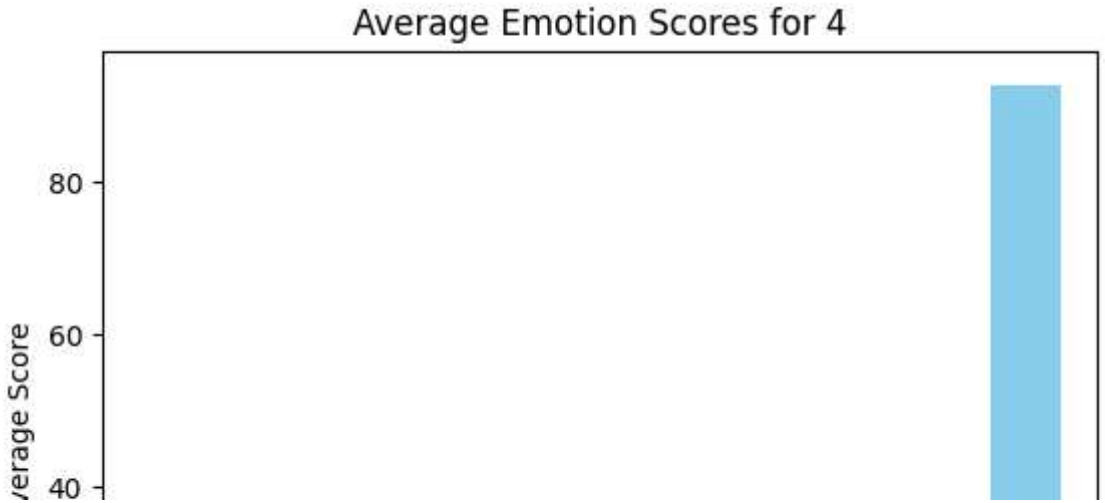

Average Emotion Scores for 2
Average Score
0
5
10
15
20
25
30
35
angry
disgust
fear
happy
sad
surprise
neutral
Emotions

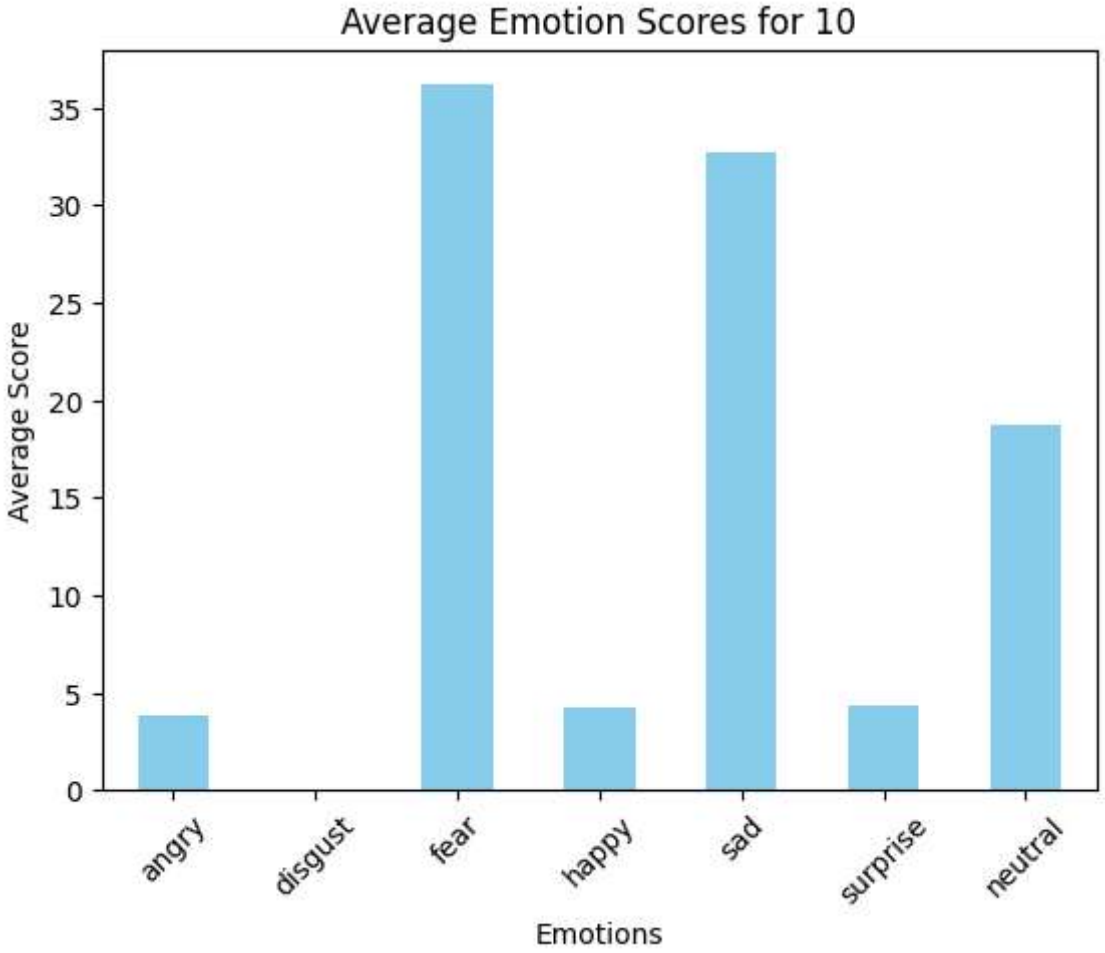

Average Emotion Scores for 9
20
25
30
35
40


Ave
15
10
5
0
angry
disgust
fear
happy
sad
surprise
neutral
Emotions

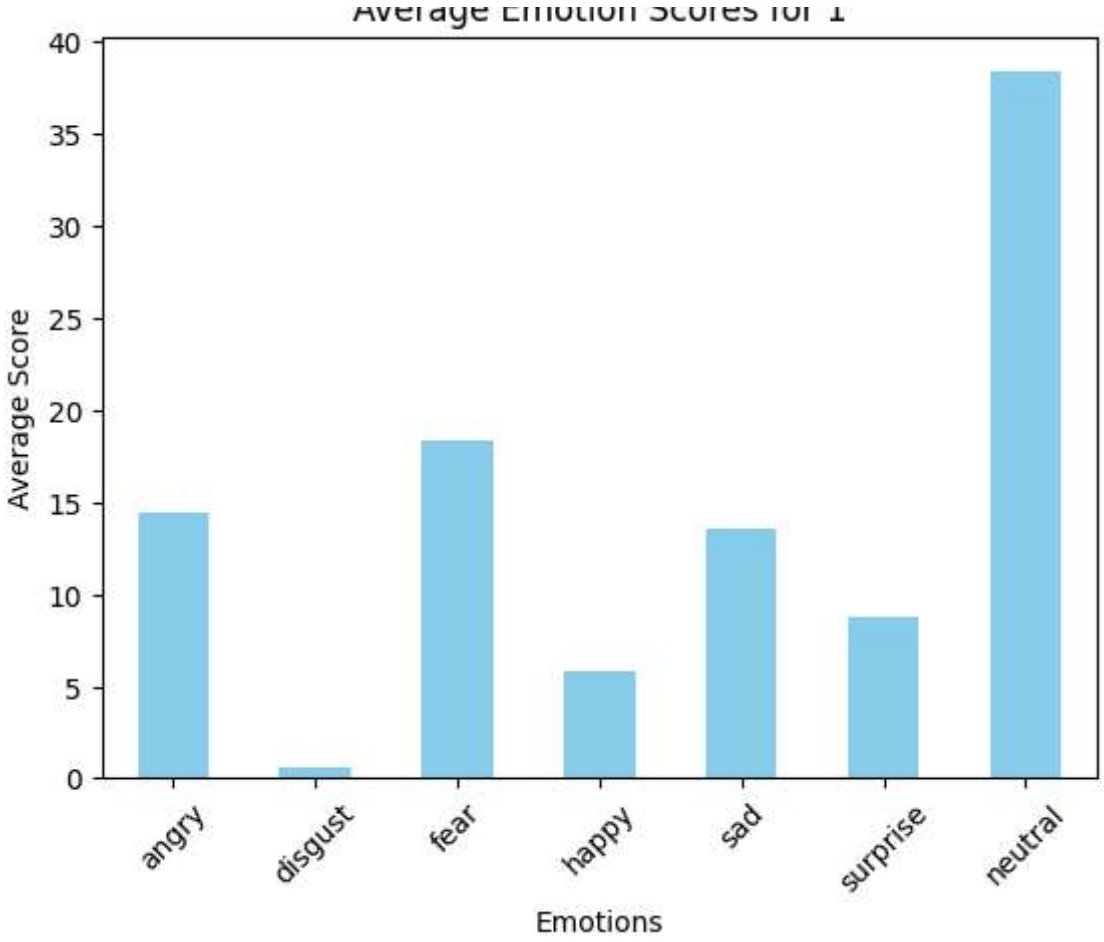

Average Emotion Scores for 6
Average Score
80
70
60
50
40
30
20
10
0
angry
disgust
fear
happy
sad
surprise
neutral
Emotions

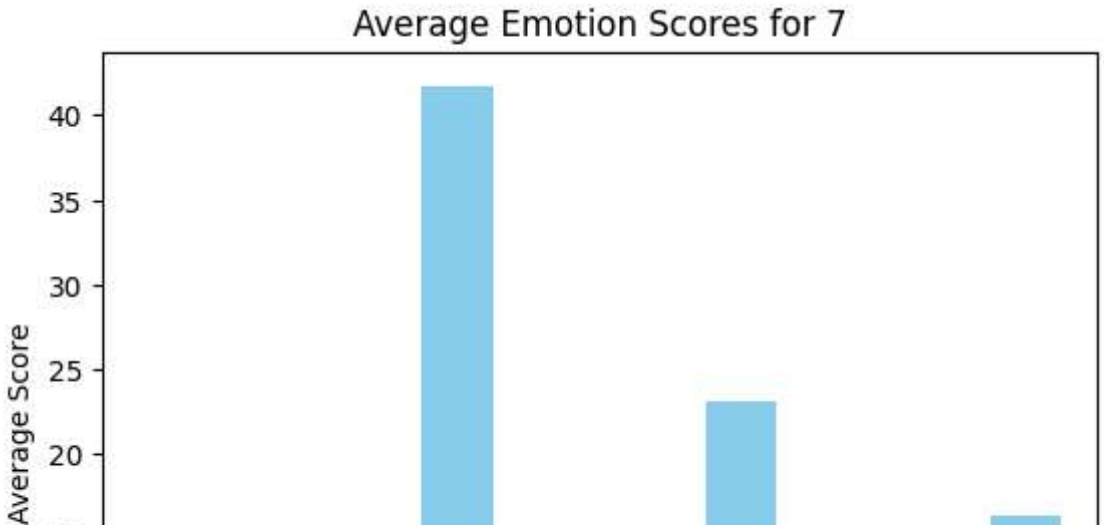

Average Emotion Scores for 8
Average Score
0
10
20
30
40
50
60
70
angry
disgust
fear
happy
sad
surprise
neutral
Emotions


Average Emotion Scores for 4
erage Score
40
60
80

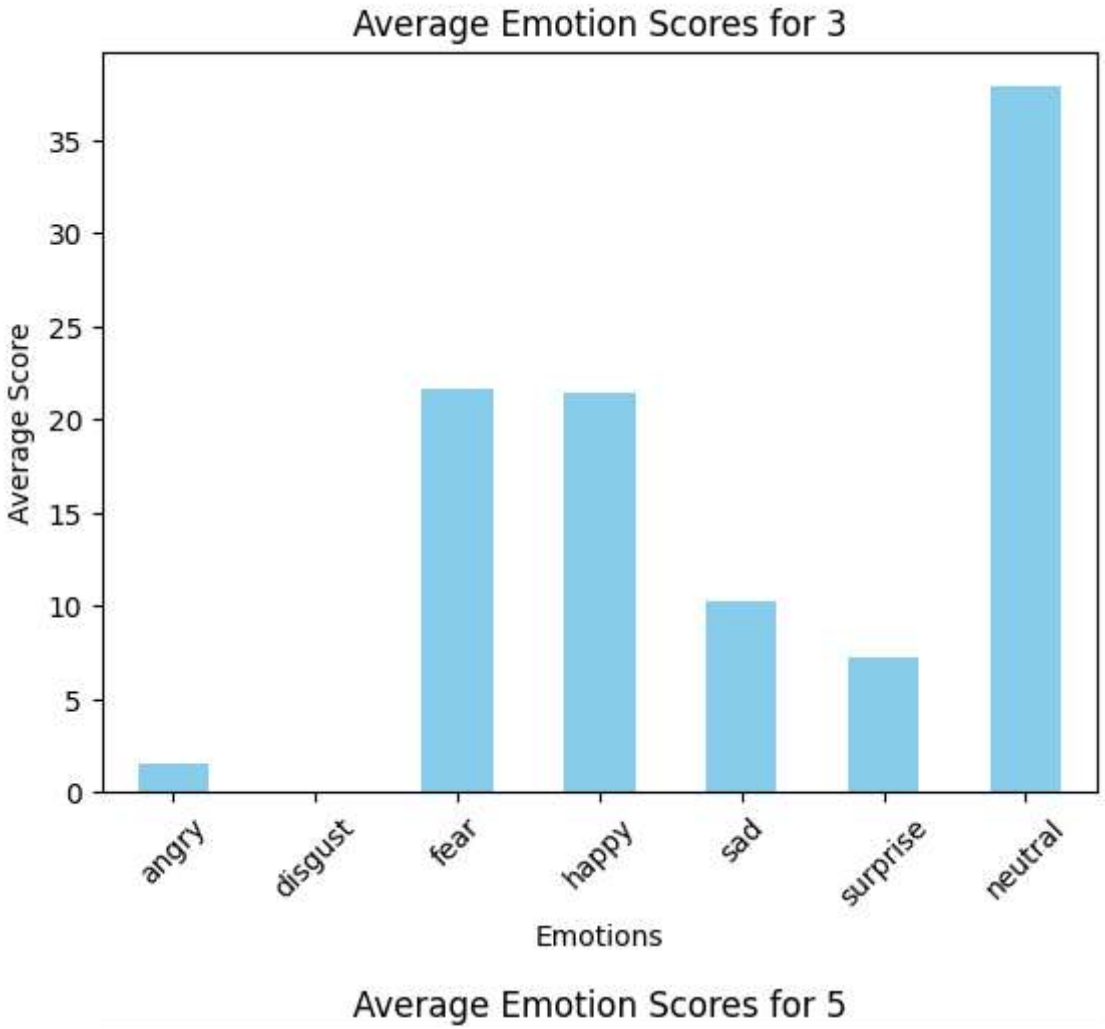


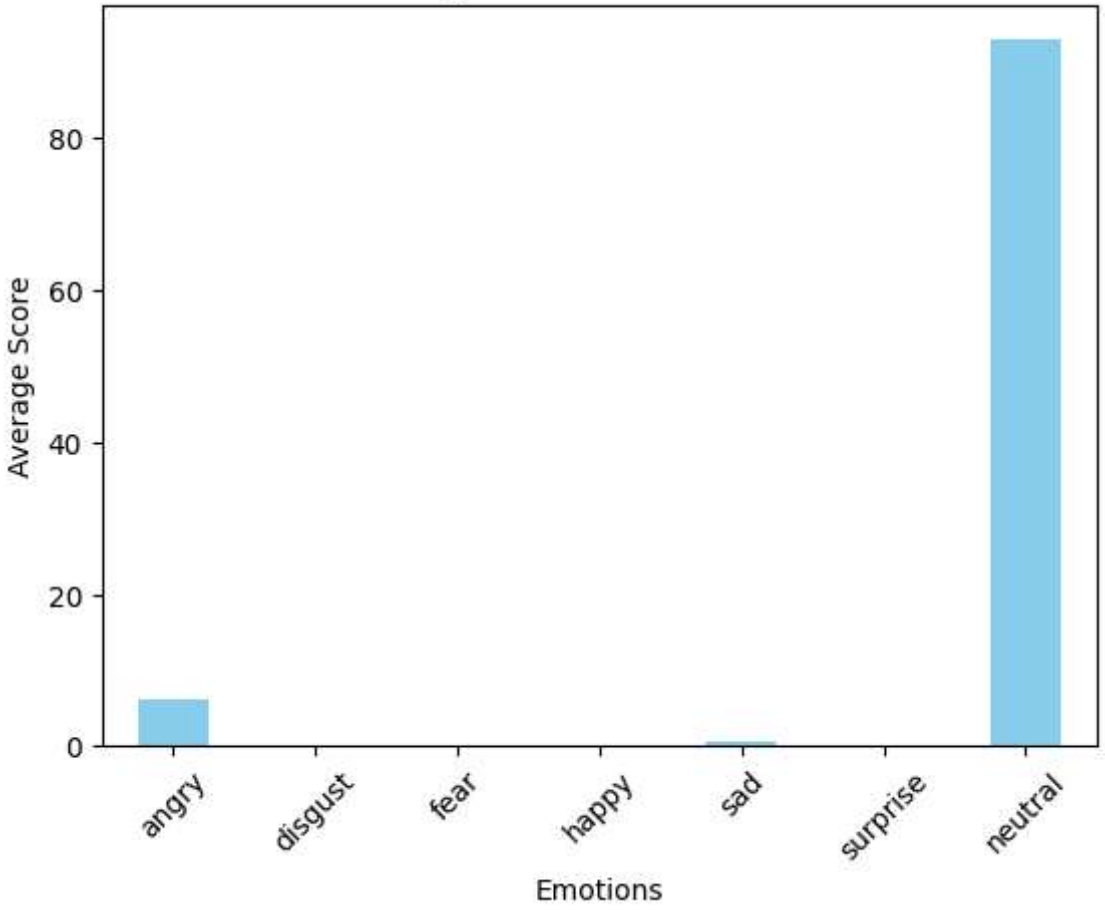
Average Emotion Scores for 1


Average Score
Emotions
angry
disgust
fear
happy
sad
surprise
neutral
0
5
10
15
20
25
30
35
40

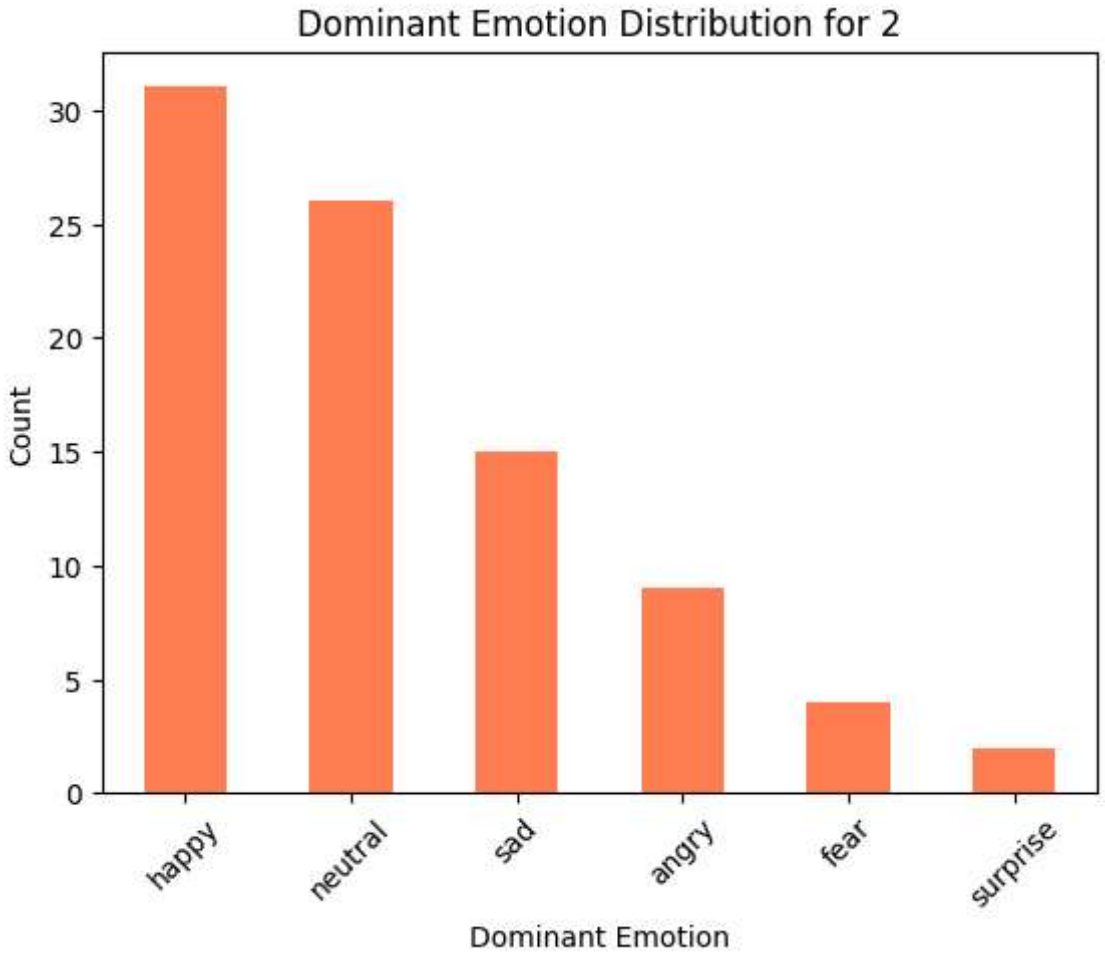

Average Emotion Scores for 7
Average Score
20
25
30
35
40

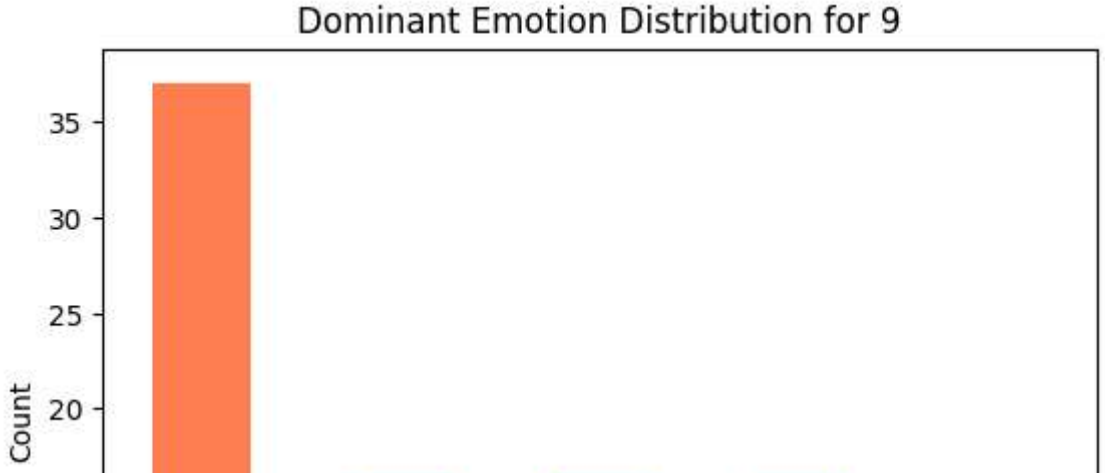

15
10
5
0
angry
disgust
fear
happy
sad
surprise
neutral
Emotions

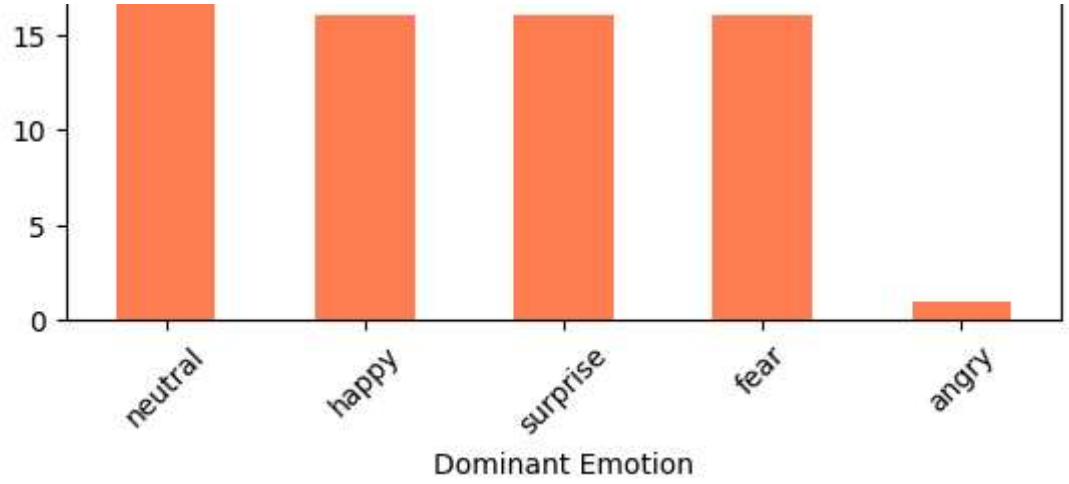

Average Emotion Scores for 3
Average Score
35
30
25
20
15
10
5
0
angry
disgust
fear
happy
sad
surprise
neutral
Emotions
Average Emotion Scores for 5

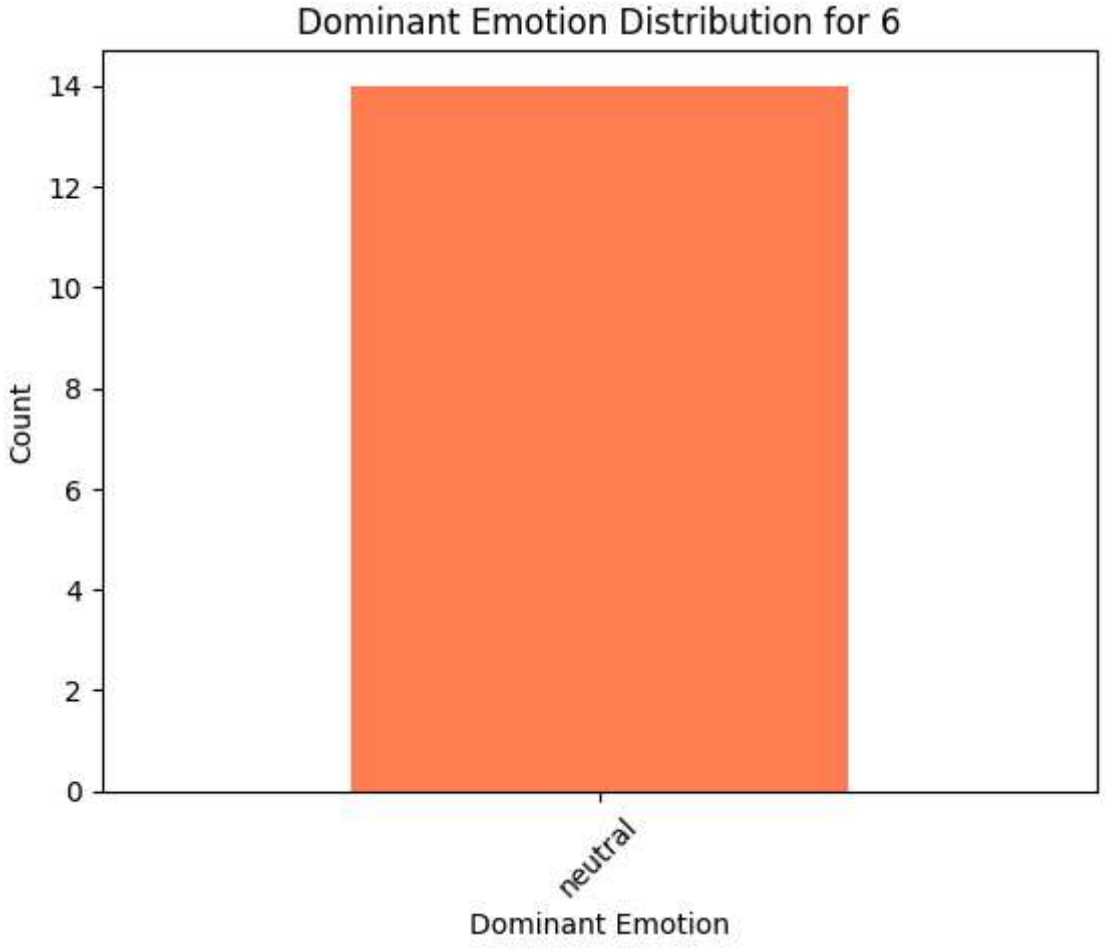

Average Score
0
20
40
60
80
angry
disgust
fear
happy
sad
surprise
neutral
Emotions

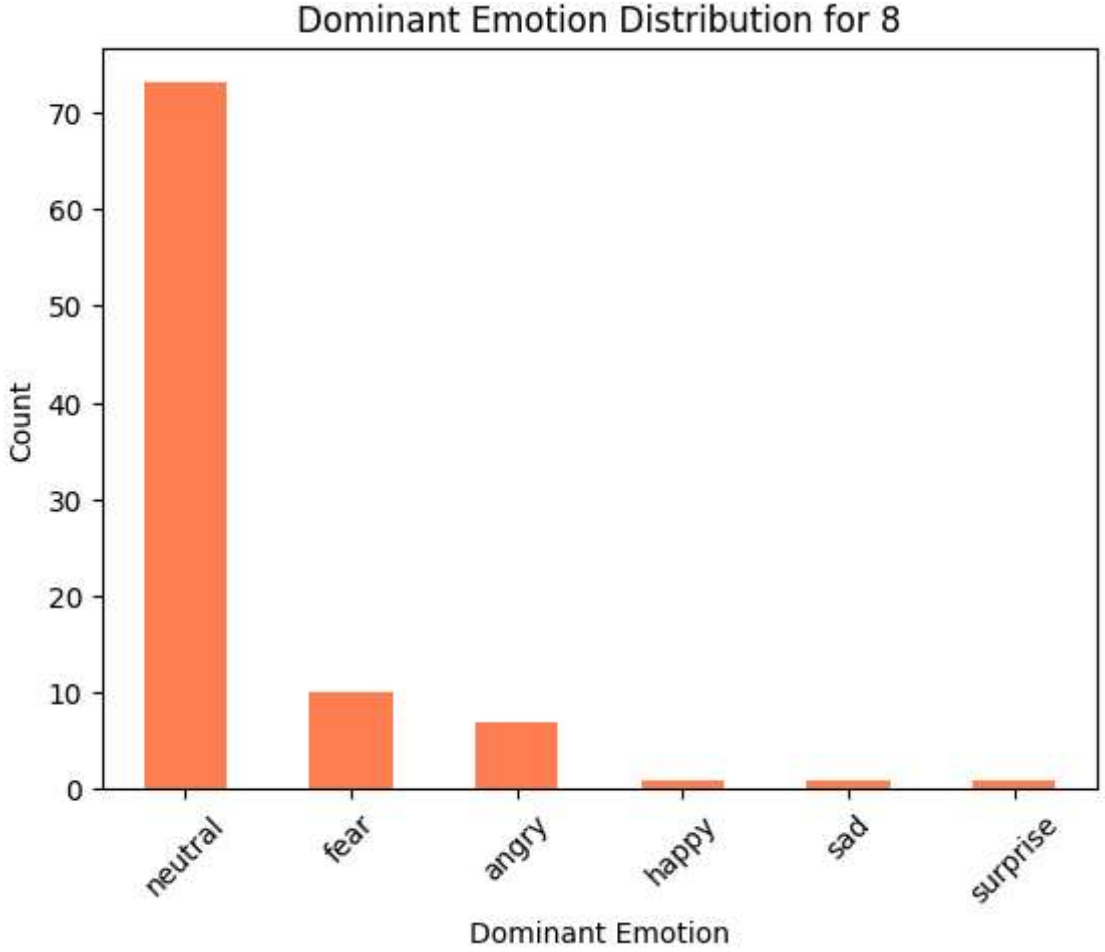
```
# Function to analyze the dominant emotion
def dominant_emotion_analysis(emotion_df, candidate_id):
    dominant_counts = emotion_df['dominant_emotion'].value_counts()
    dominant_counts.plot(kind='bar', color='coral')
    plt.title(f'Dominant Emotion Distribution for {candidate_id}')
    plt.ylabel('Count')
    plt.xlabel('Dominant Emotion')
    plt.xticks(rotation=45)
    plt.show()

# Analyze for each candidate
for candidate_id, df in emotion_dfs.items():
    dominant_emotion_analysis(df, candidate_id)
```

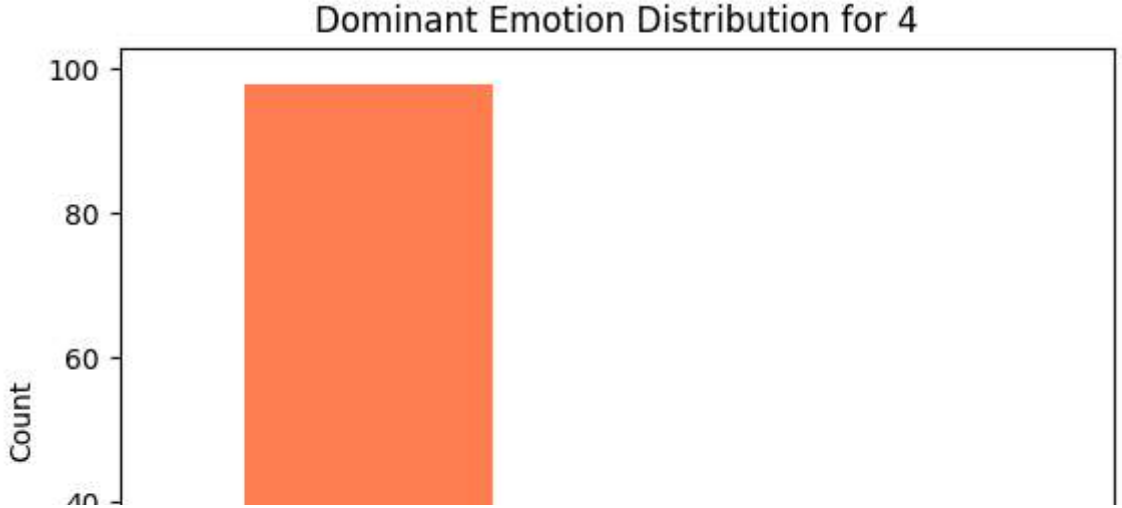

Dominant Emotion Distribution for 2
Count
0
5
10
15
20
25
30
happy
neutral
sad
angry
fear
surprise
Dominant Emotion

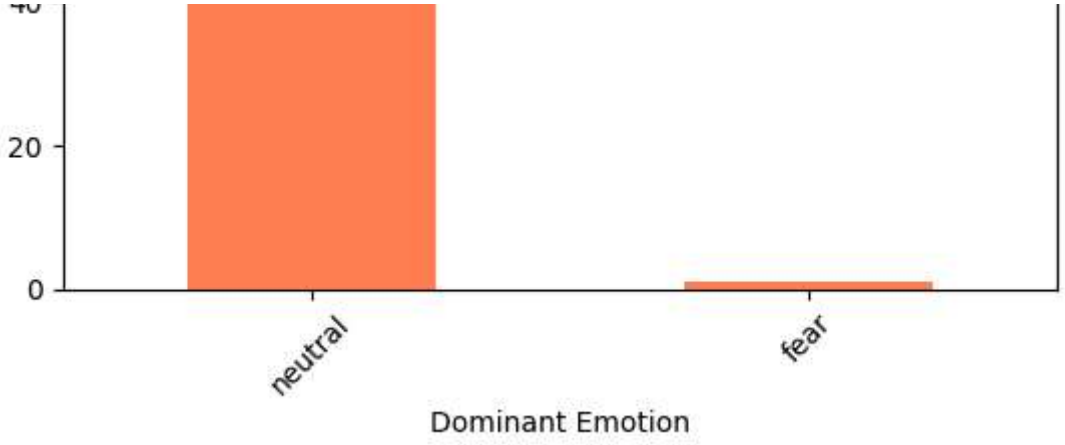

Dominant Emotion Distribution for 9
Count
20
25
30
35

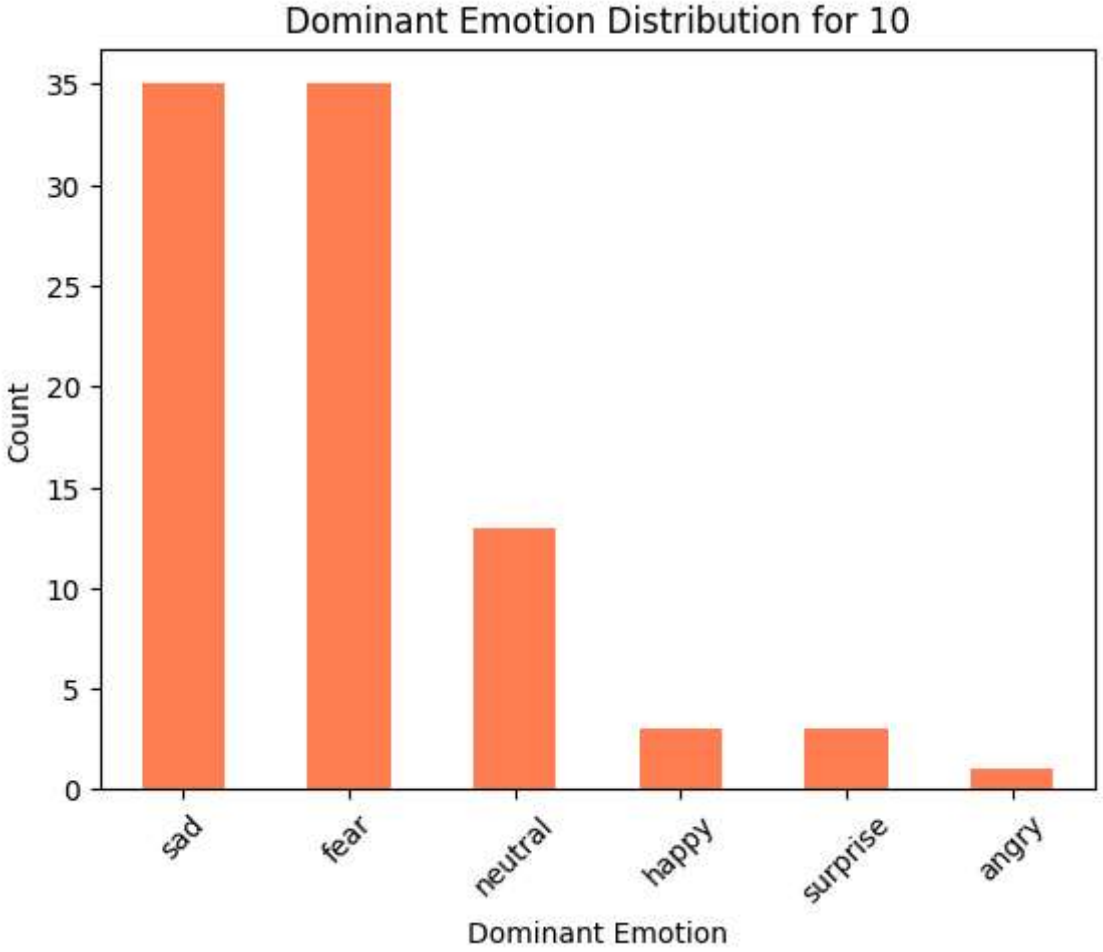

15
10
5
0
neutral
happy
surprise
fear
angry
Dominant Emotion


Dominant Emotion Distribution for 6
14
12
10
8
6
4
2
0
Count
neutral
Dominant Emotion

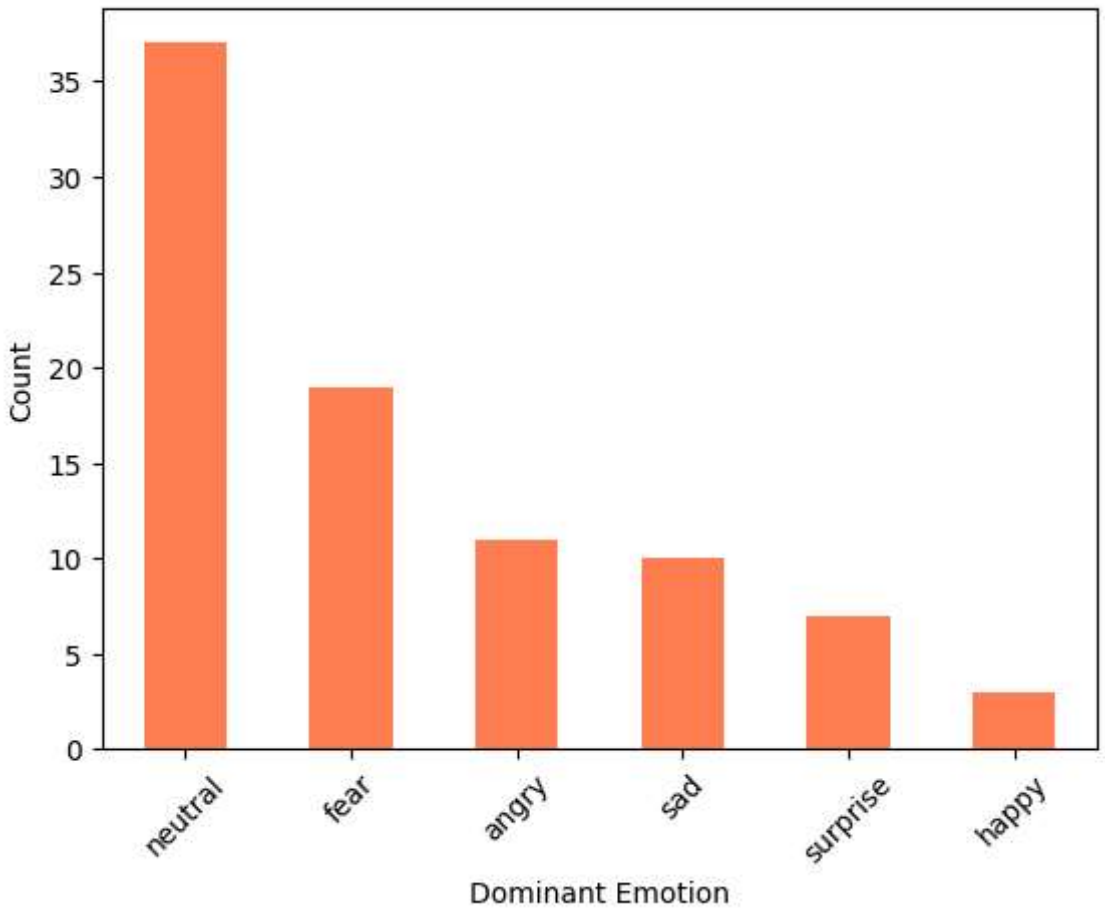

Dominant Emotion Distribution for 8
Count
0
10
20
30
40
50
60
70
neutral
fear
angry
happy
sad
surprise
Dominant Emotion

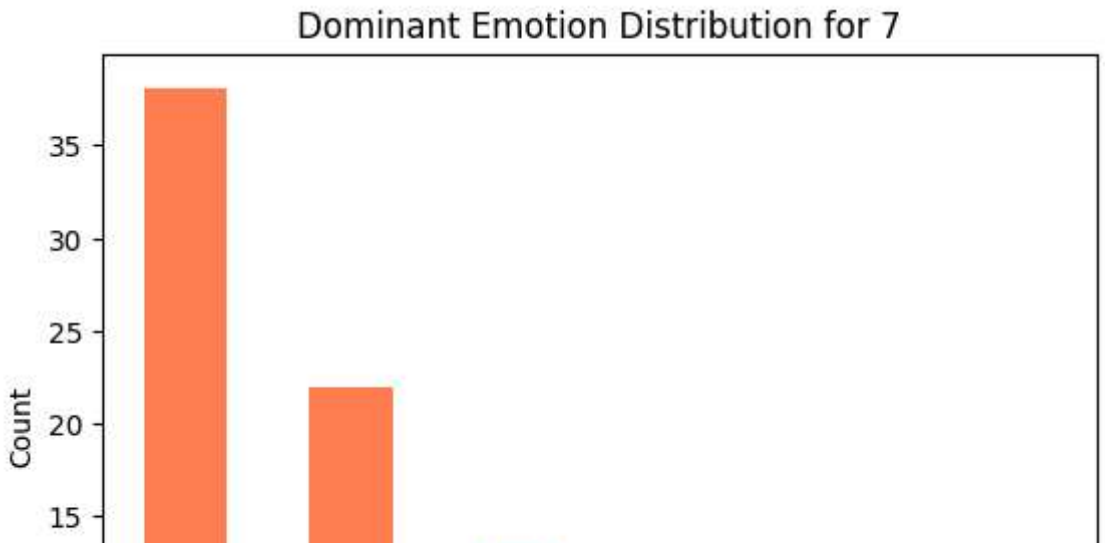

Dominant Emotion Distribution for 4
Count
40
60
80
100

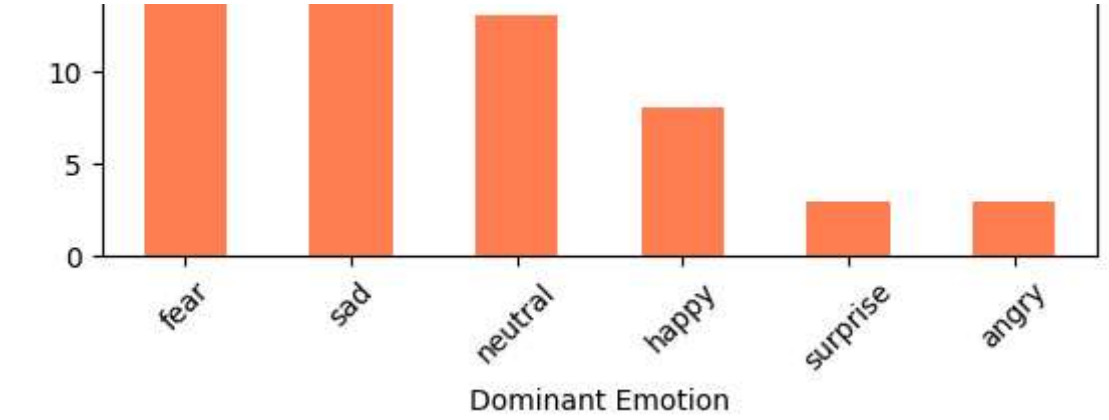


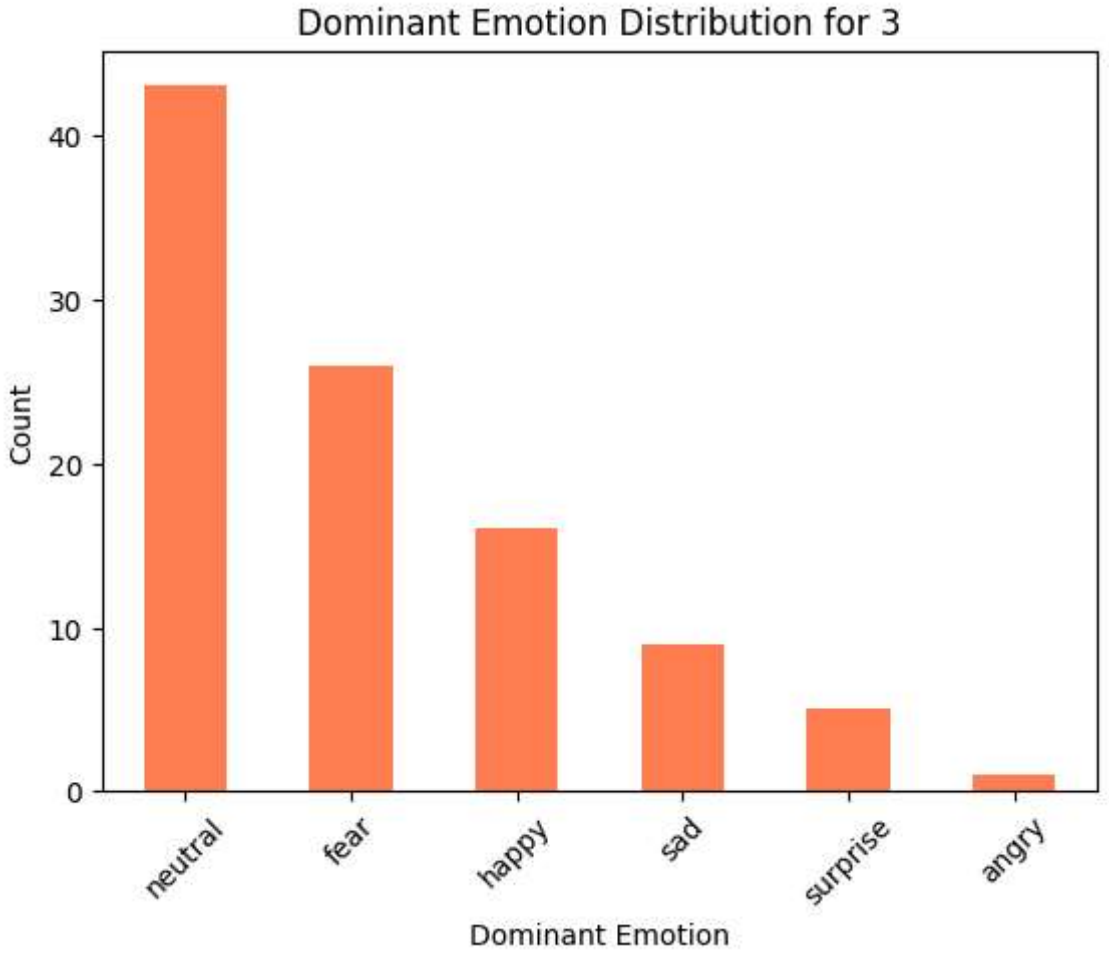


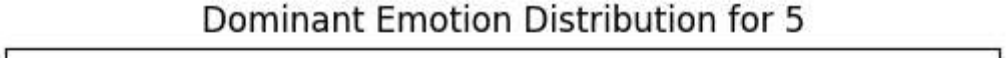
Dominant Emotion Distribution for 1

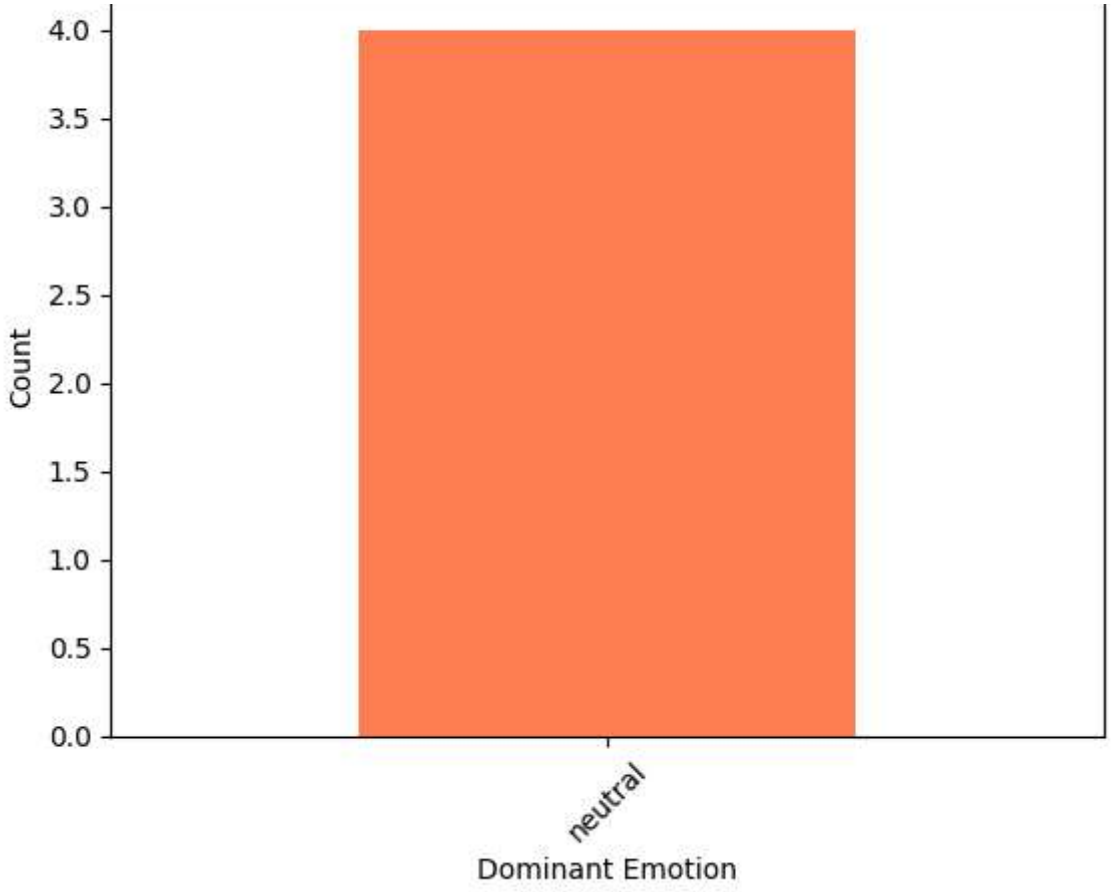

35
30
25
20
15
10
5
0
Count
neutral
fear
angry
sad
surprise
happy
Dominant Emotion


Dominant Emotion Distribution for 7
35
30
25
20
15
Count

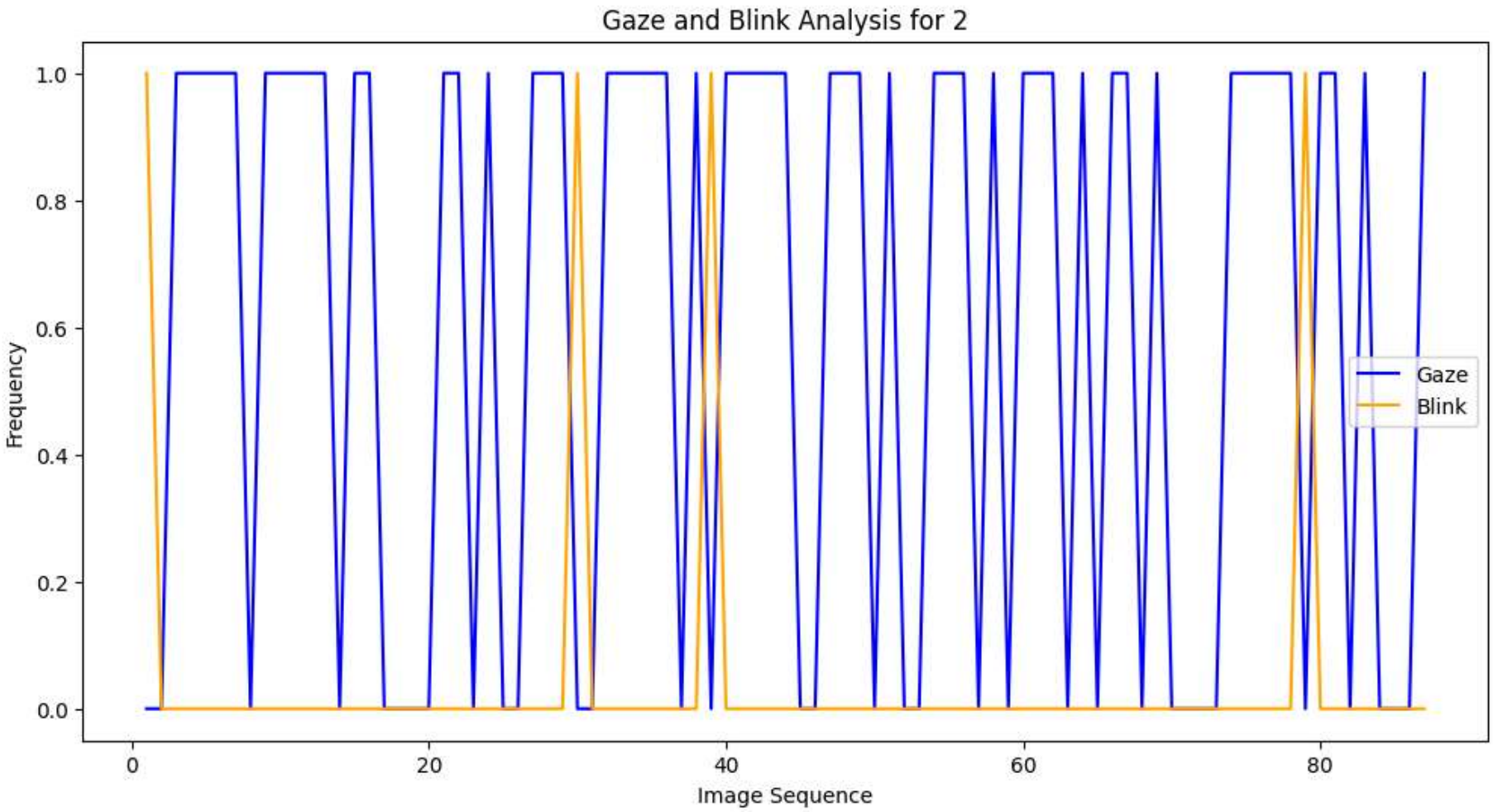

10
5
0
fear
sad
neutral
happy
surprise
angry
Dominant Emotion

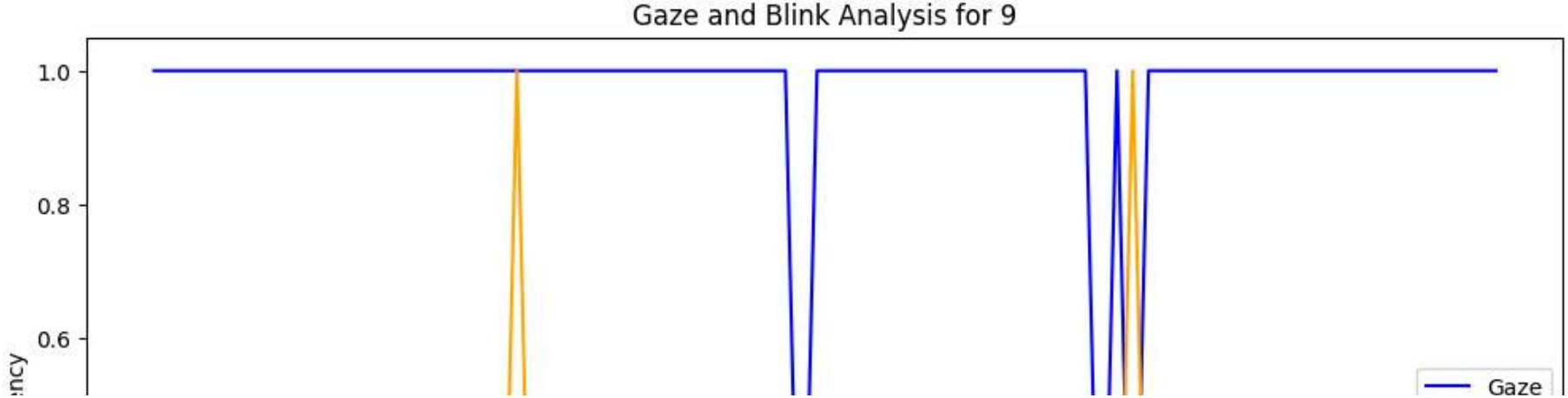

Dominant Emotion Distribution for 3
40
30
20
10
0
Count
neutral
fear
happy
sad
surprise
angry
Dominant Emotion

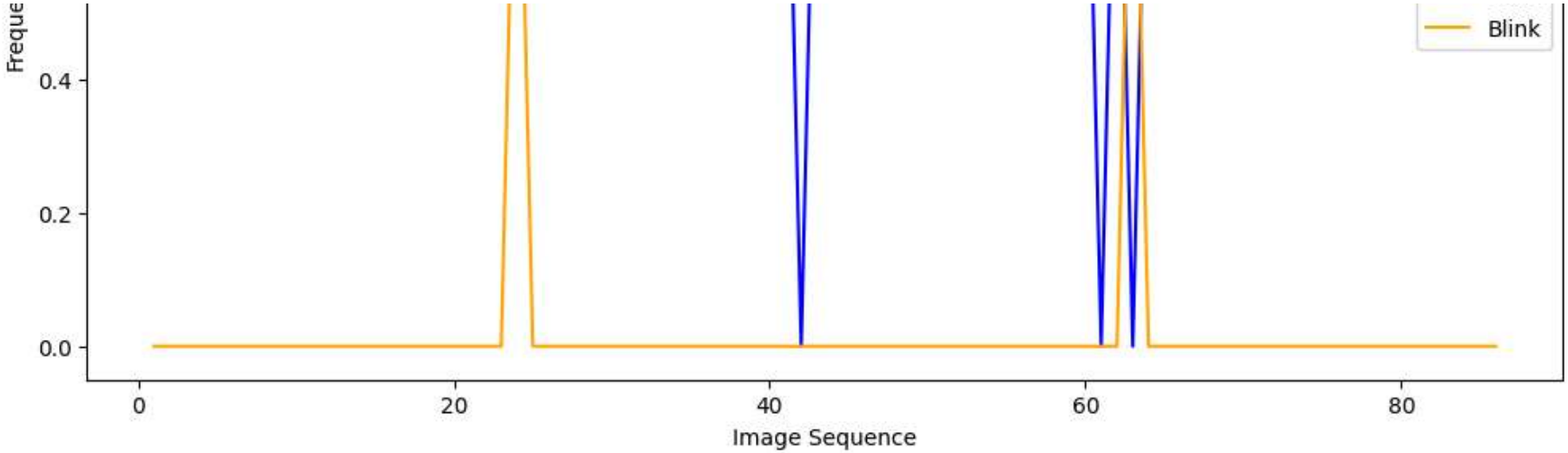

Dominant Emotion Distribution for 5

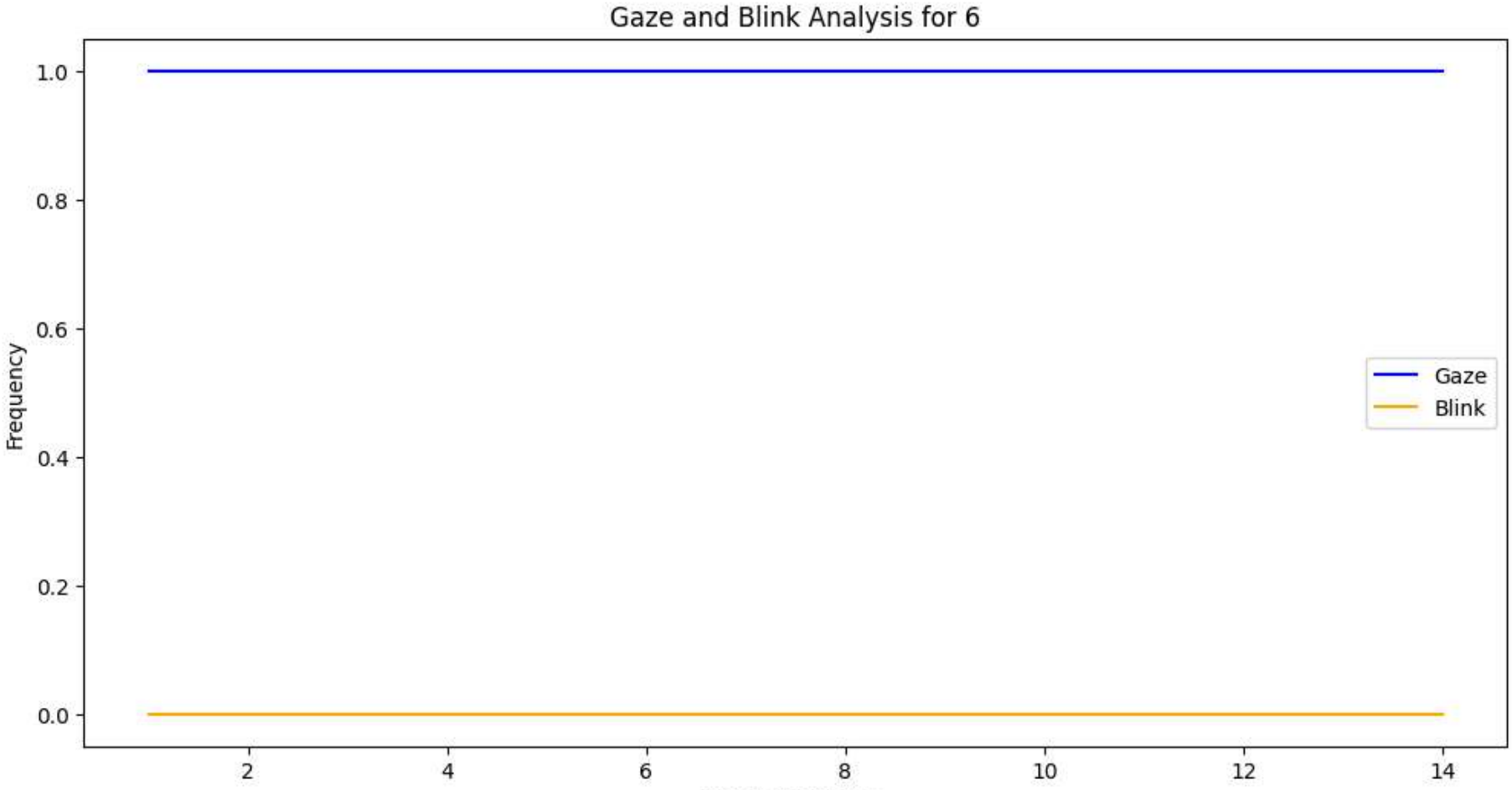

4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Count
neutral
Dominant Emotion

```
# Function to analyze gaze and blink frequency
def gaze_analysis(gaze_df, candidate_id):
    plt.figure(figsize=(12, 6))
    sns.lineplot(data=gaze_df, x='image_seq', y='gaze', label='Gaze', color='blue')
    sns.lineplot(data=gaze_df, x='image_seq', y='blink', label='Blink', color='orange')
    plt.title(f'Gaze and Blink Analysis for {candidate_id}')
    plt.xlabel('Image Sequence')
    plt.ylabel('Frequency')
    plt.legend()
    plt.show()

# Analyze for each candidate
for candidate_id, df in gaze_dfs.items():
    gaze_analysis(df, candidate_id)
```


Gaze and Blink Analysis for 2
Frequency
Image Sequence
Gaze
Blink
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80


Gaze and Blink Analysis for 9
Gaze
1.0
0.8
0.6


Freque
0.4
0.2
0.0
0
20
40
60
80
Image Sequence
Blink


Gaze and Blink Analysis for 6
Frequency
1.0
0.8
0.6
0.4
0.2
0.0
2
4
6
8
10
12
14
Gaze
Blink

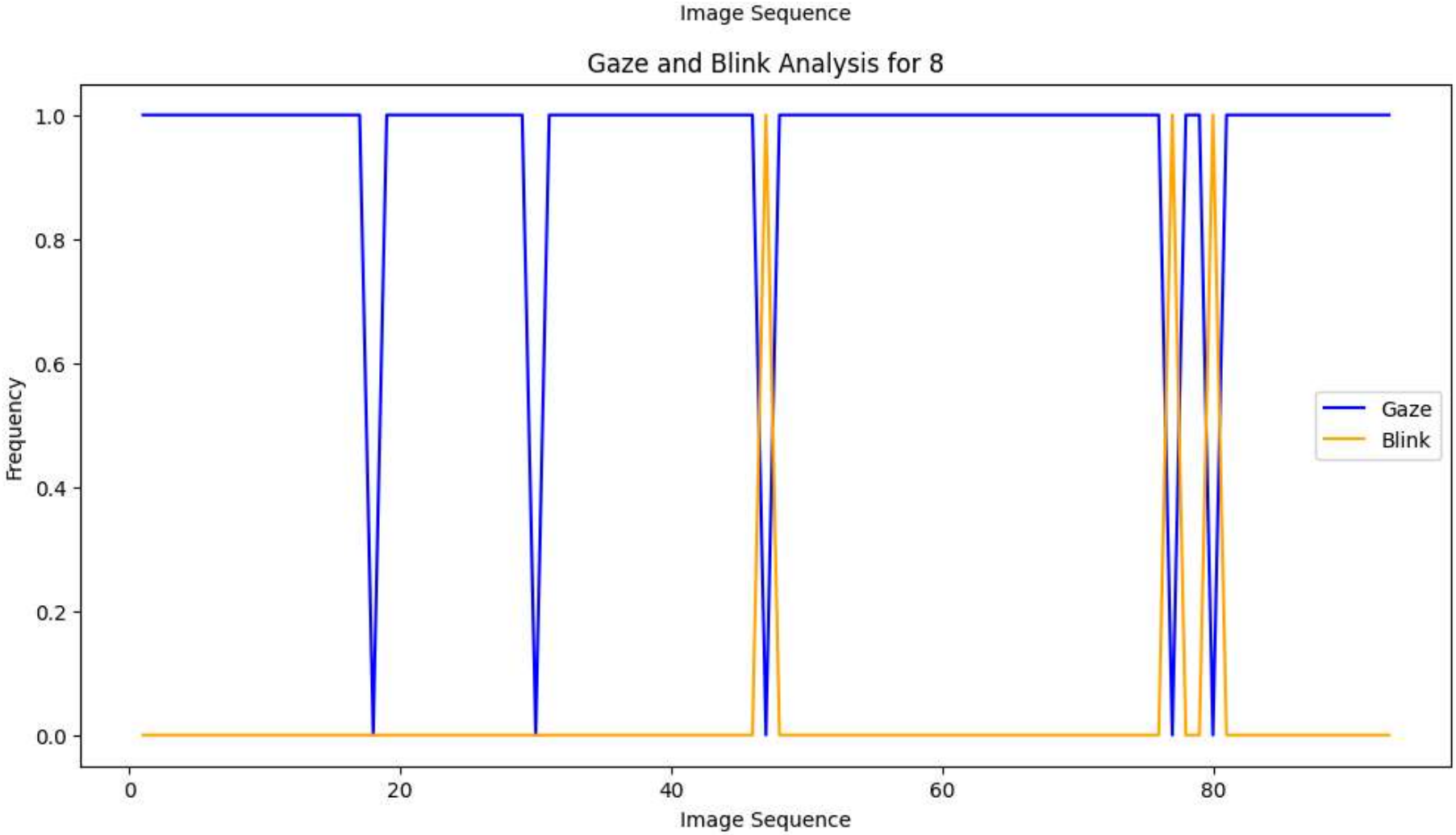
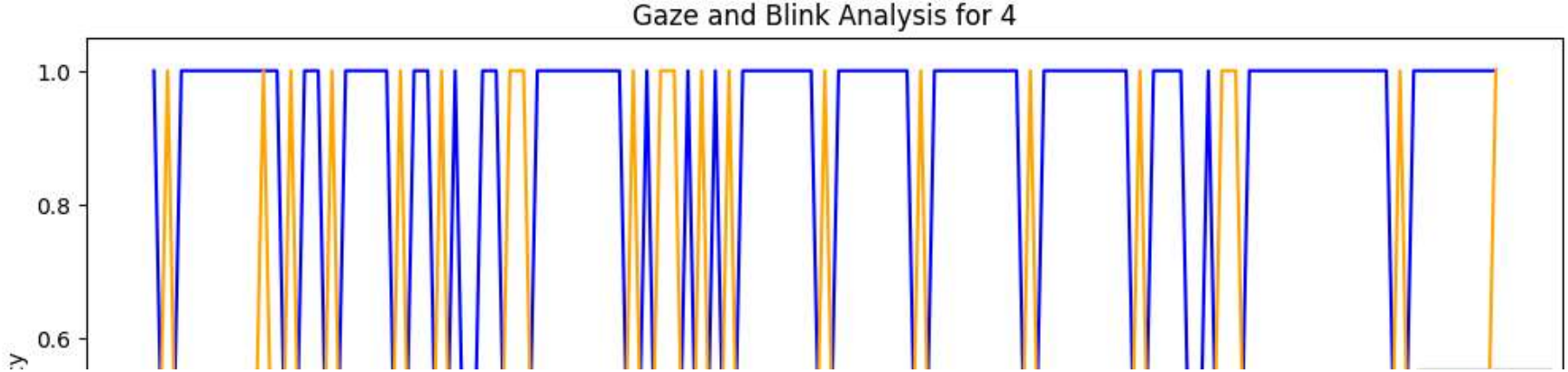
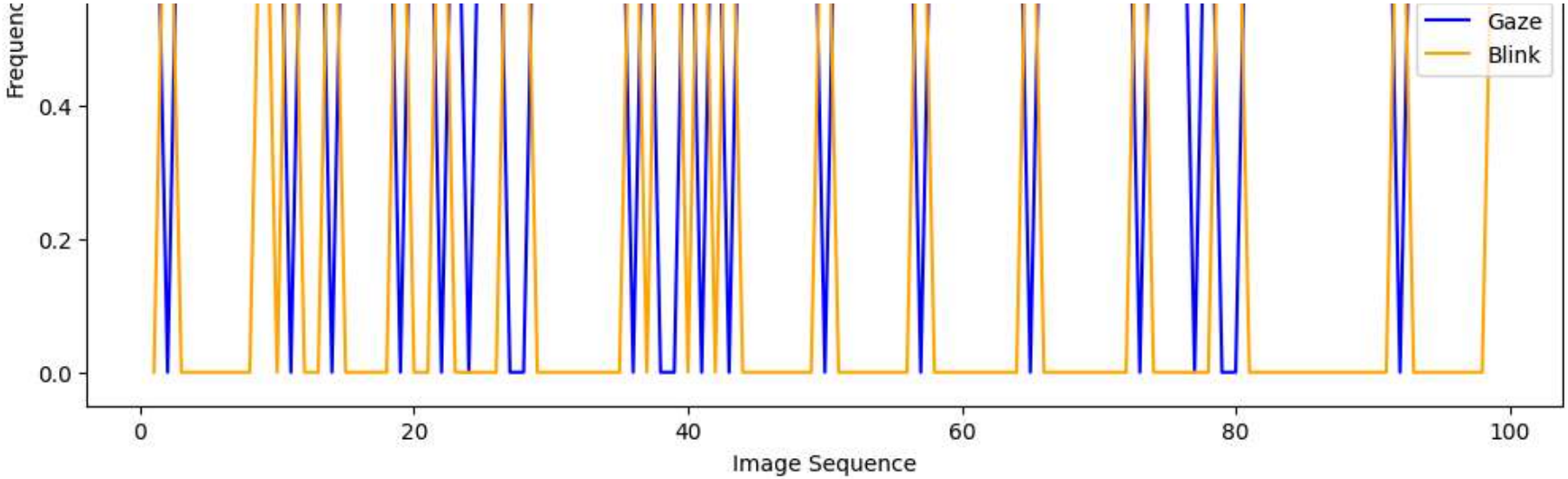
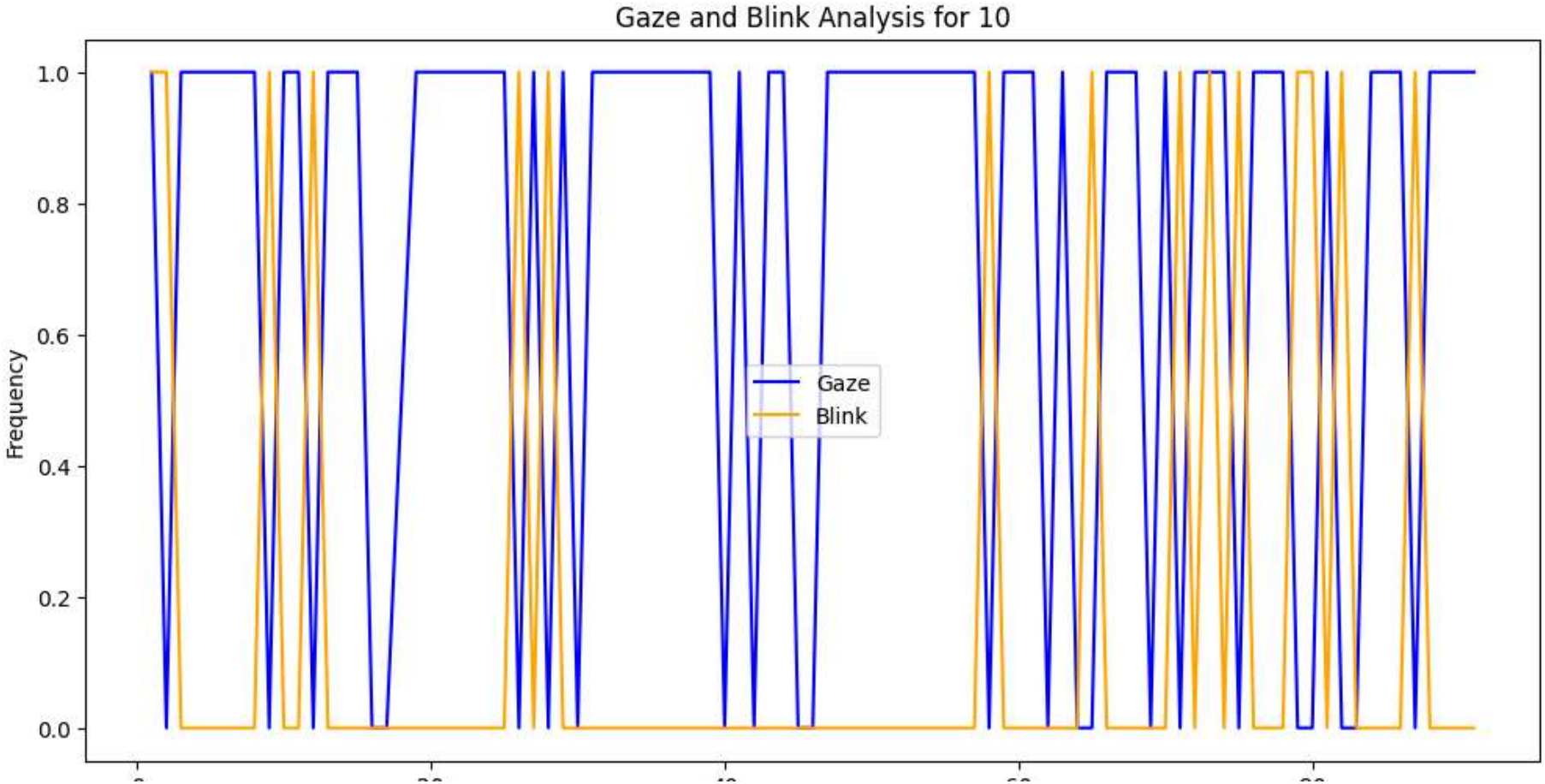
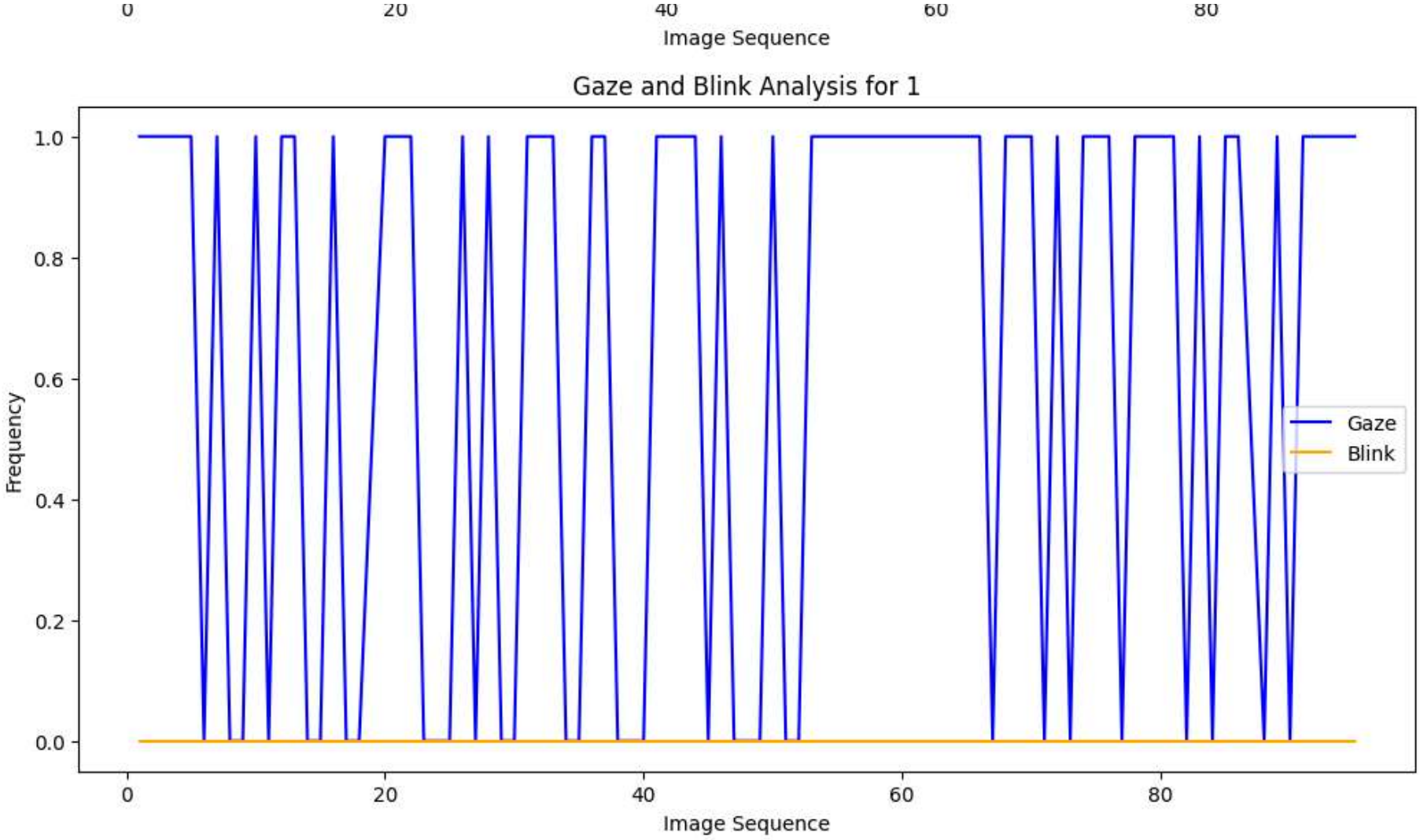


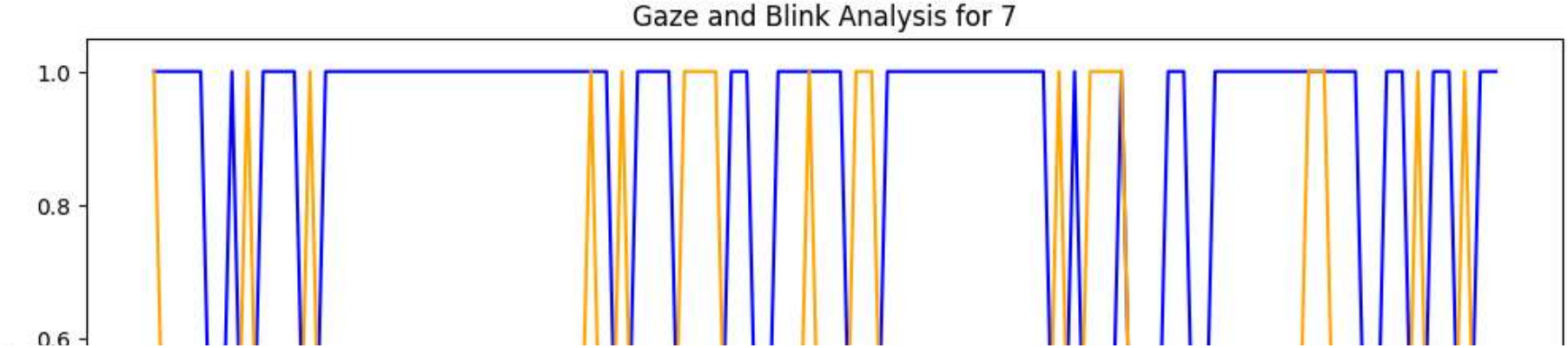
Image Sequence
Gaze and Blink Analysis for 8
Frequency
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
Image Sequence
Gaze
Blink

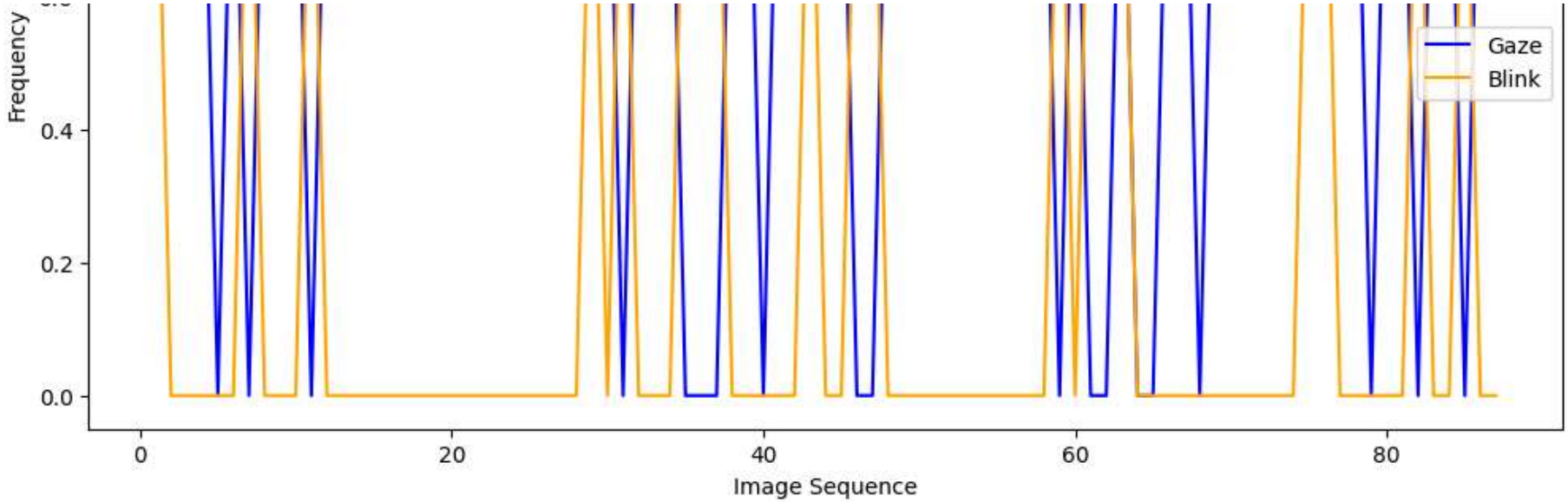

Gaze and Blink Analysis for 4
1.0
0.8
0.6

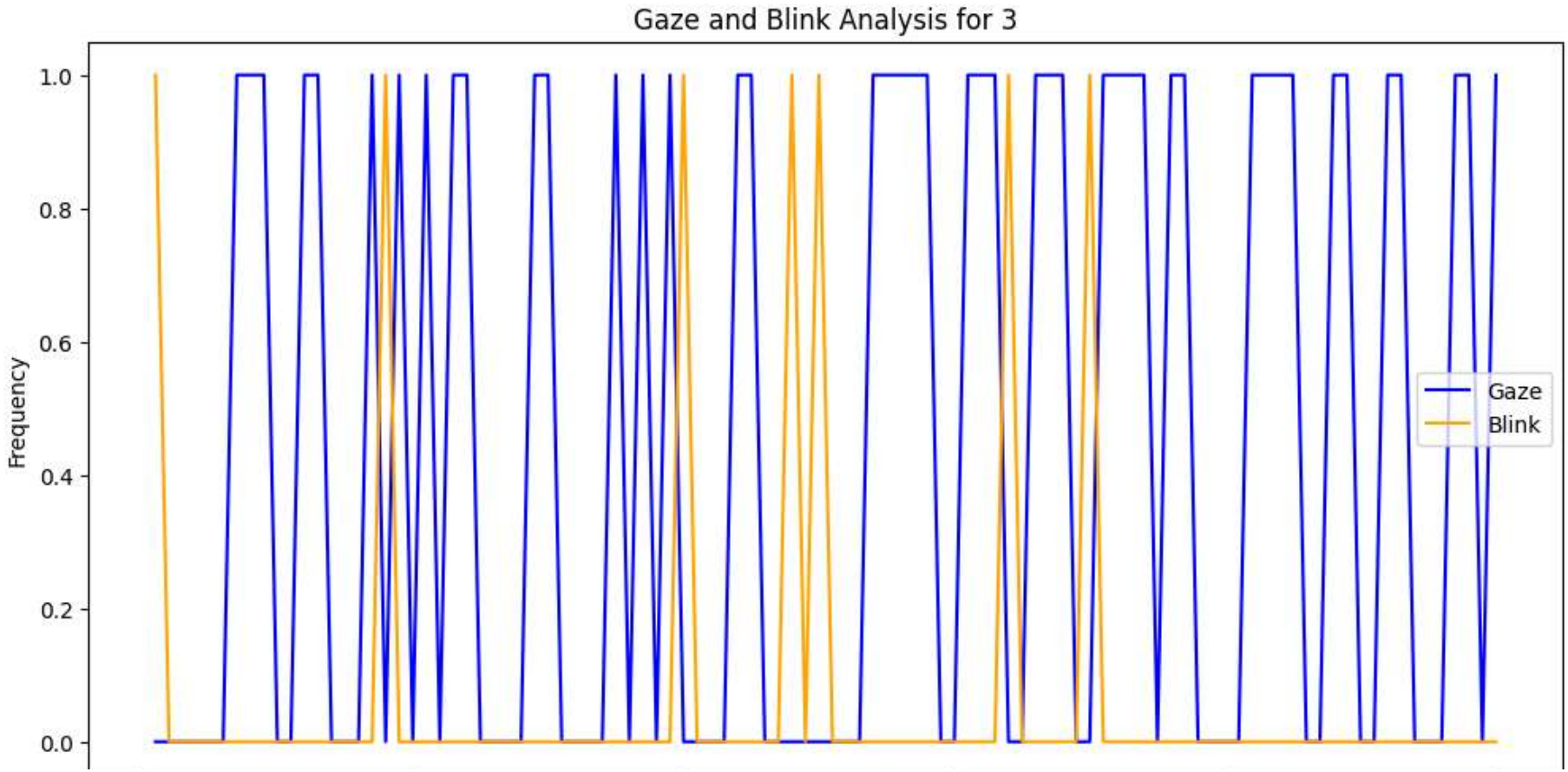

Frequenc
0.4
0.2
0.0
0
20
40
60
80
100
Image Sequence
Gaze
Blink

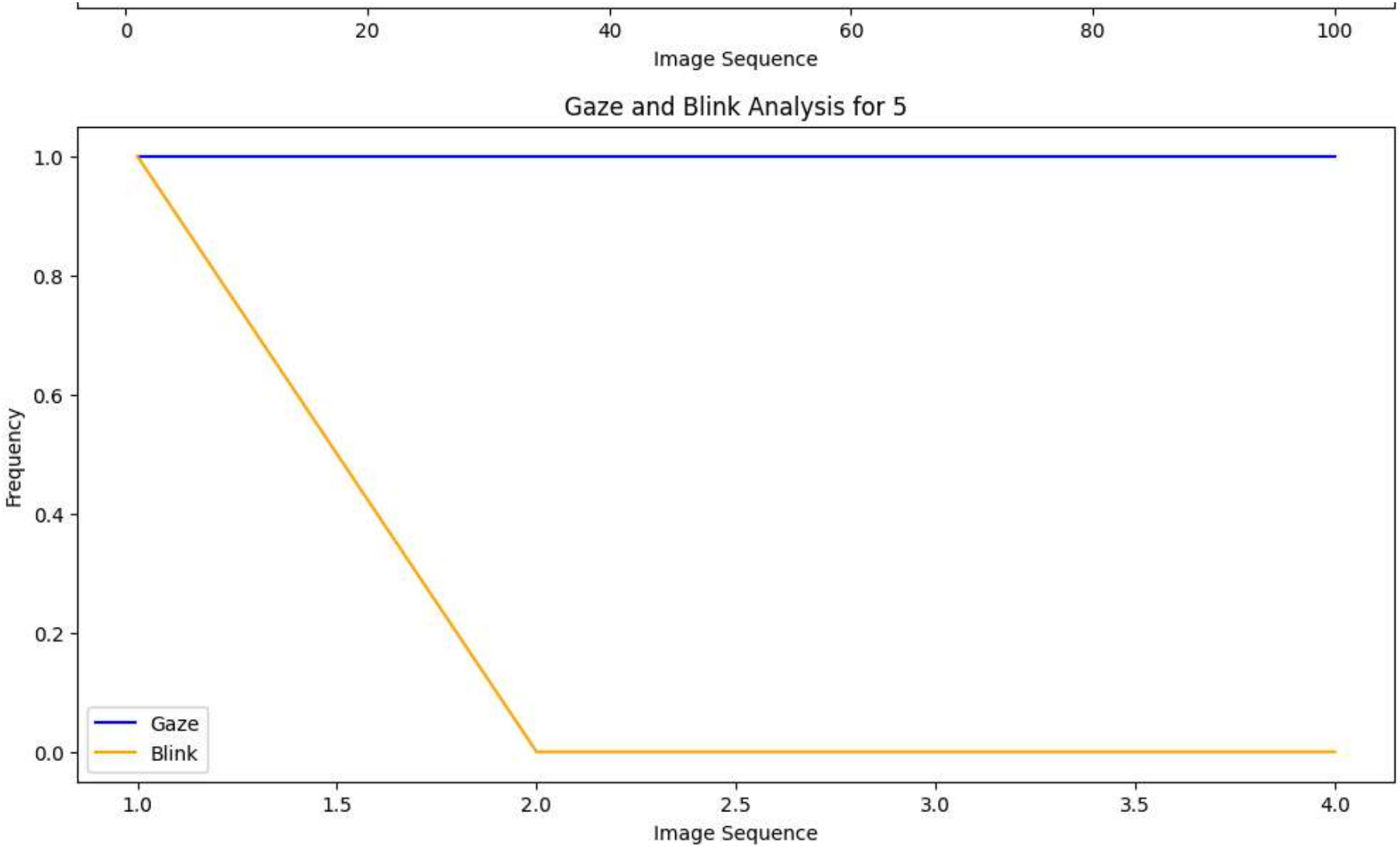

Gaze and Blink Analysis for 10
Frequency
1.0
0.8
0.6
0.4
0.2
0.0
Gaze
Blink


0
20
40
60
80
Image Sequence
Gaze and Blink Analysis for 1
1.0
0.8
0.6
0.4
0.2
0.0
Frequency
Gaze
Blink
0
20
40
60
80
Image Sequence


Gaze and Blink Analysis for 7
1.0
0.8
0.6


Frequency
0.4
0.2
0.0
0
20
40
60
80
Image Sequence
Gaze
Blink

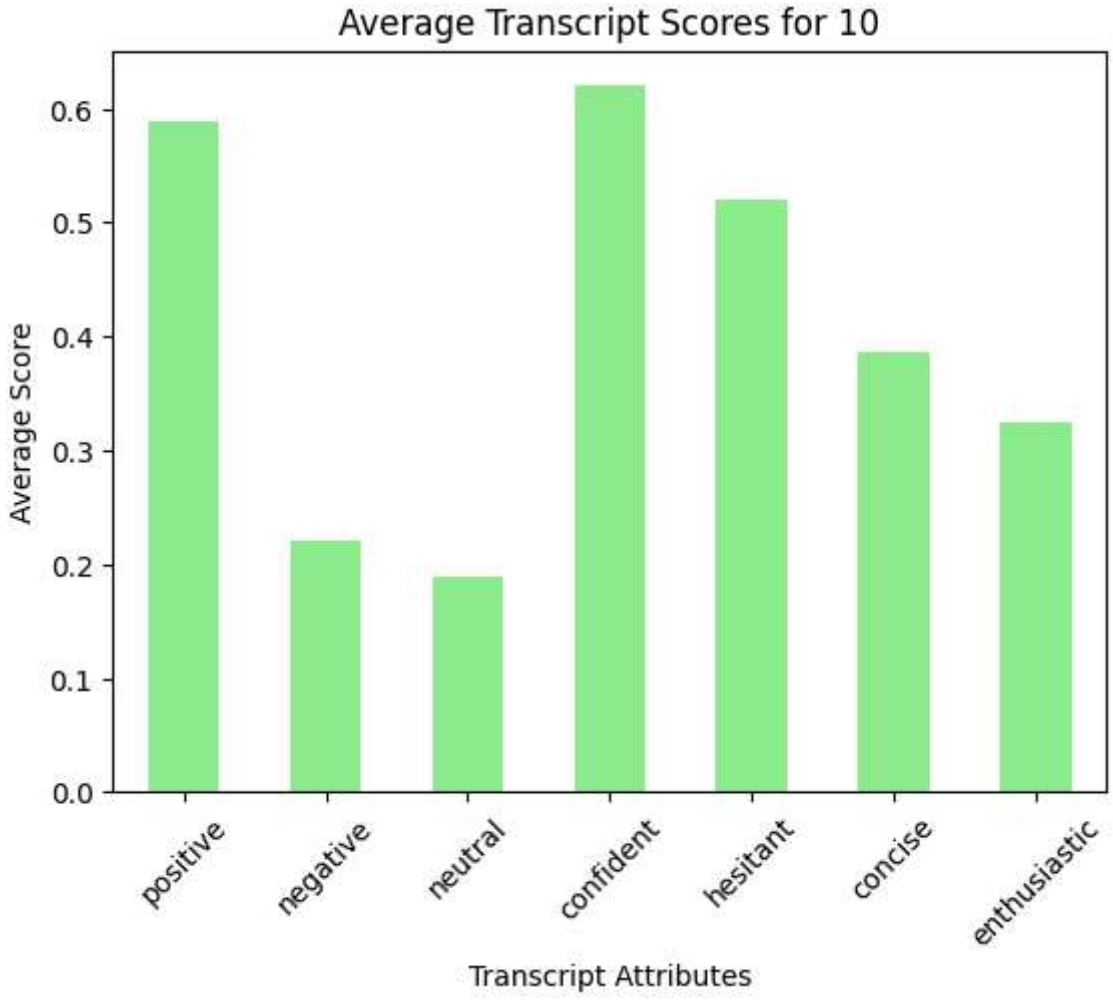

Gaze and Blink Analysis for 3
Frequency
1.0
0.8
0.6
0.4
0.2
0.0
Gaze
Blink

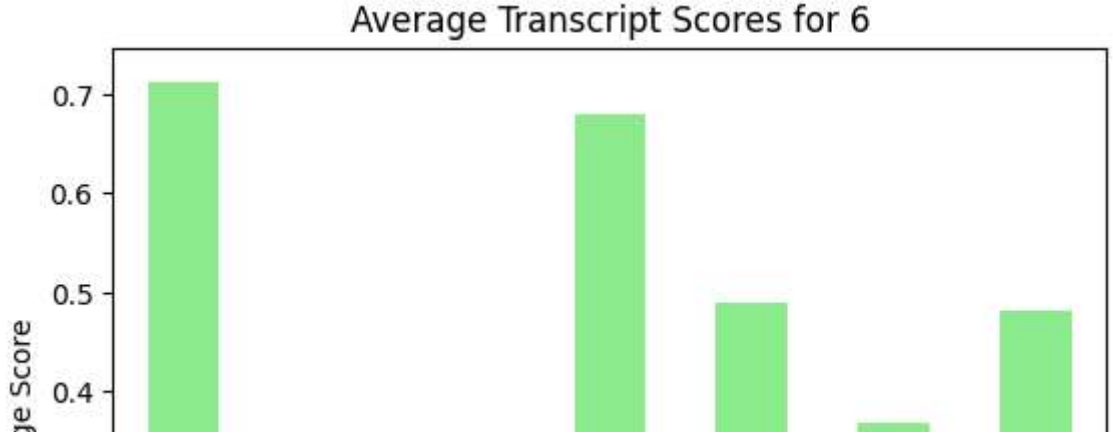

0
20
40
60
80
100
Image Sequence
Gaze and Blink Analysis for 5
1.0
0.8
0.6
0.4
0.2
0.0
Frequency
Gaze
Blink
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Image Sequence

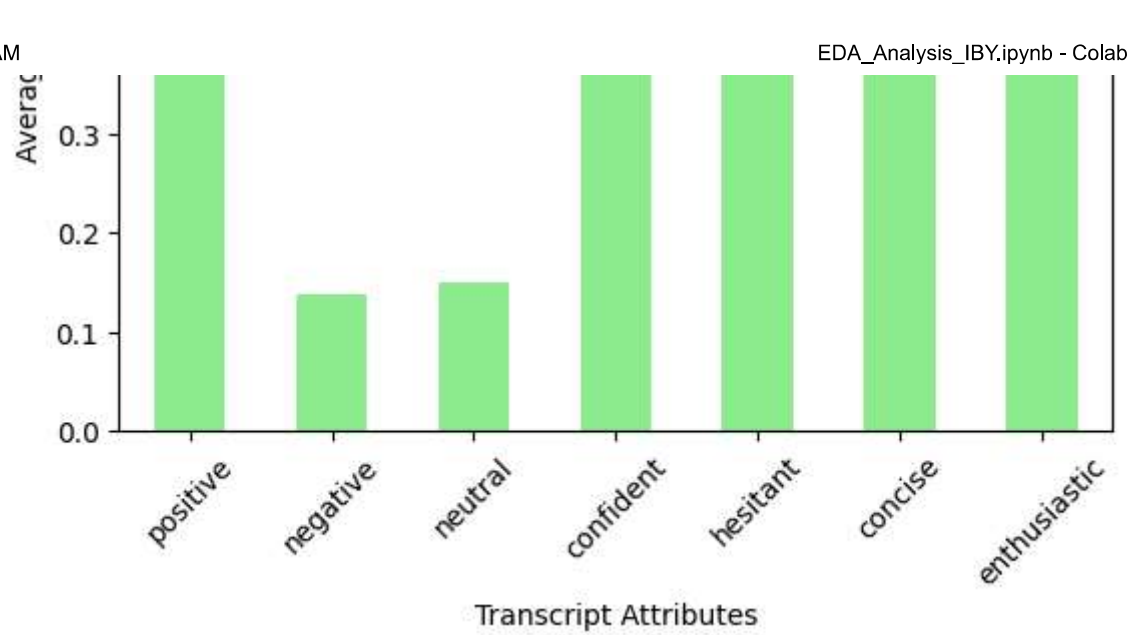
```
# Define the path to the Transcript Scores folder
transcript_scores_path = os.path.join(extraction_path, 'transcript_data')

# Load transcript scores for each candidate
transcript_dfs = {}

for transcript_file in os.listdir(transcript_scores_path):
    candidate_id = transcript_file.split('.')[0]  # Assuming filename is candidate_id.xlsx
    transcript_dfs[candidate_id] = pd.read_csv(os.path.join(transcript_scores_path, transcript_file))

# Display a sample of transcript scores for one candidate
print(transcript_dfs[list(transcript_dfs.keys())[0]].head())
```

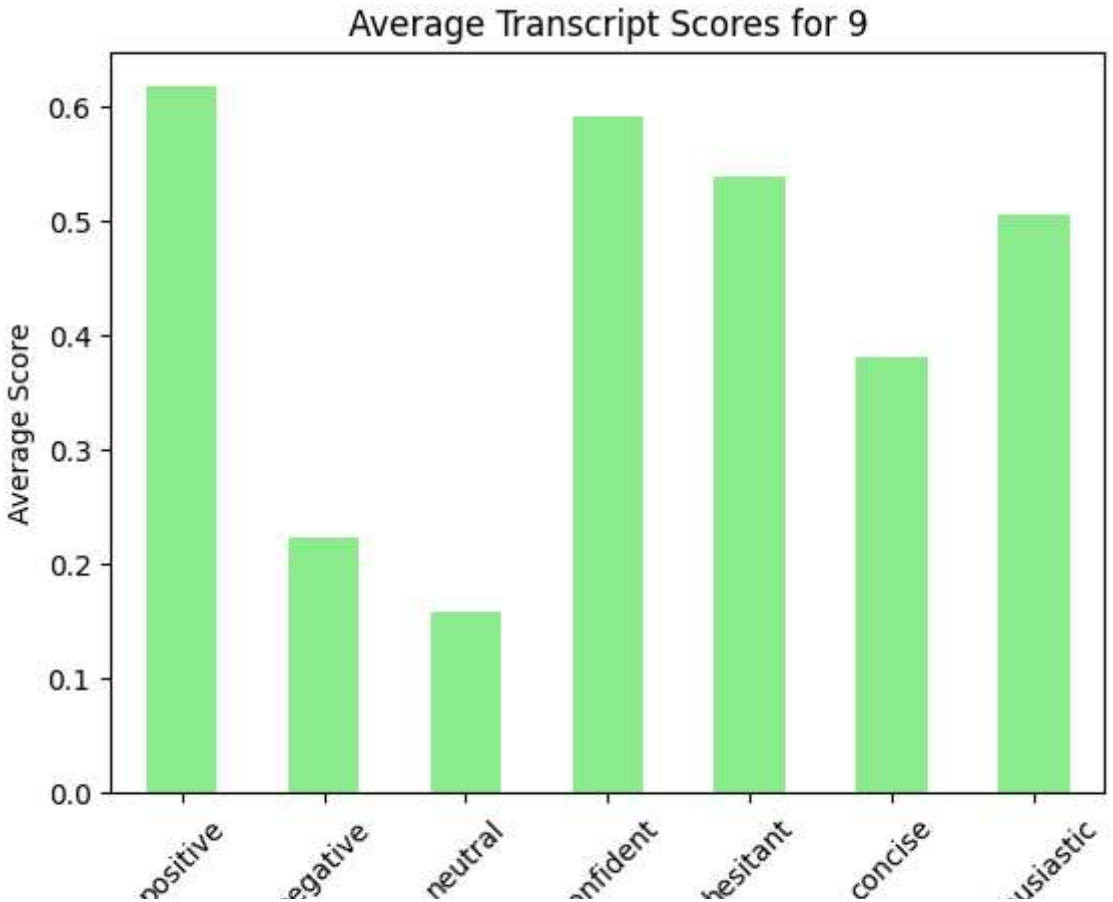
```
   id  seek  start    end                                               text  \
0   0     0   0.00   6.04   My name is Michael Ramos, I am from Patna, Bi...
1   1     0   6.04  12.08   I went up to do my graduation in B.Com Honour...
2   2     0  12.08  16.68   as well as a Tax Associate wherein I got the ...
3   3     0  16.68  22.56   learned in my B.Com to apply them in real lif...
4   4     0  22.56  26.76   been involved in a lot of extracurricular act...

                                              tokens  temperature  \
0  [50364, 1222, 1315, 307, 49328, 2786, 11, 286,...          0.0
1  [50666, 286, 1437, 493, 281, 360, 452, 15652, ...          0.0
2  [50968, 382, 731, 382, 257, 23263, 28520, 4353...          0.0
3  [51198, 3264, 294, 452, 363, 13, 14627, 281, 3...          0.0
4  [51492, 668, 3288, 294, 257, 688, 295, 1279, 1...          0.0

   avg_logprob  compression_ratio  no_speech_prob  positive  negative  \
0    -0.331398           1.622711        0.543254  0.457349  0.206400
1    -0.331398           1.622711        0.543254  0.532671  0.222764
2    -0.331398           1.622711        0.543254  0.714758  0.117540
3    -0.331398           1.622711        0.543254  0.606619  0.132189
4    -0.331398           1.622711        0.543254  0.629161  0.139293

    neutral  confident  hesitant   concise  enthusiastic  speech_speed
0  0.336251   0.888680  0.881647  0.705780      0.623946      2.980132
1  0.244564   0.732424  0.750270  0.620743      0.367078      2.814570
2  0.167701   0.764776  0.384795  0.388321      0.316069      4.130435
3  0.261192   0.829503  0.472637  0.776919      0.383717      2.721088
4  0.231547   0.732535  0.387367  0.098993      0.349005      3.571429
```

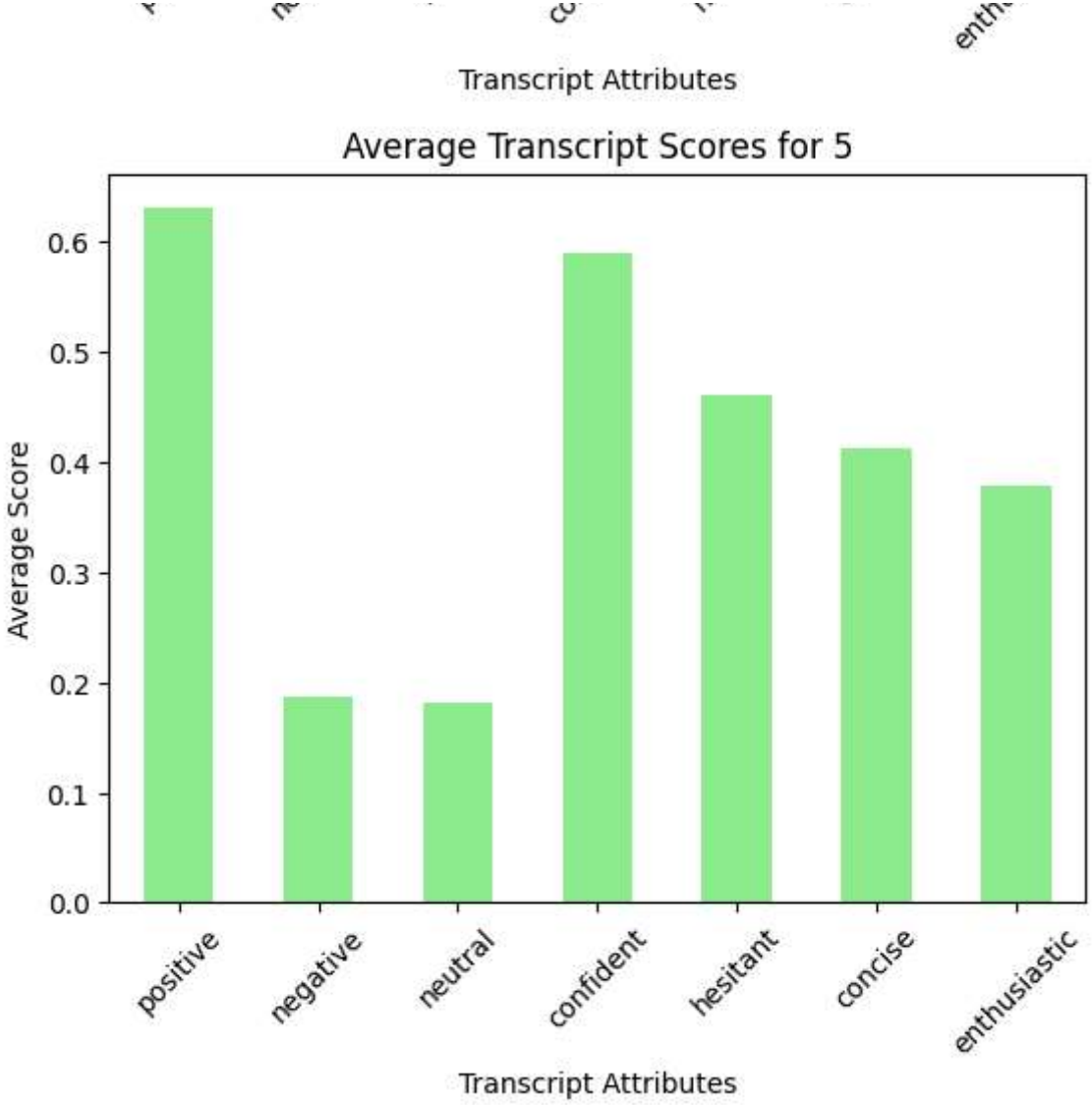
```
# Function to plot transcript score distributions
def transcript_score_distribution(transcript_df, candidate_id):
    score_cols = ['positive', 'negative', 'neutral', 'confident', 'hesitant', 'concise', 'enthusiastic']
    transcript_df[score_cols].mean().plot(kind='bar', color='lightgreen')
    plt.title(f'Average Transcript Scores for {candidate_id}')
    plt.ylabel('Average Score')
    plt.xlabel('Transcript Attributes')
    plt.xticks(rotation=45)
    plt.show()

# Analyze for each candidate
for candidate_id, df in transcript_dfs.items():
    transcript_score_distribution(df, candidate_id)
```

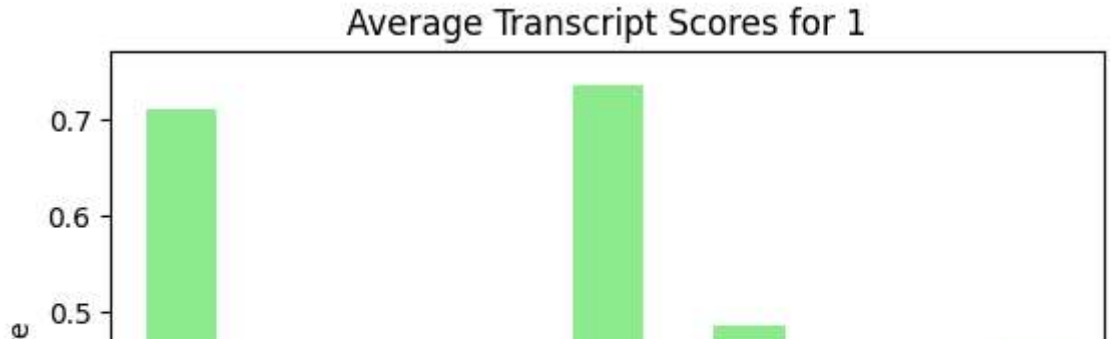

Average Transcript Scores for 10
Average Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
positive
negative
neutral
confident
hesitant
concise
enthusiastic
Transcript Attributes

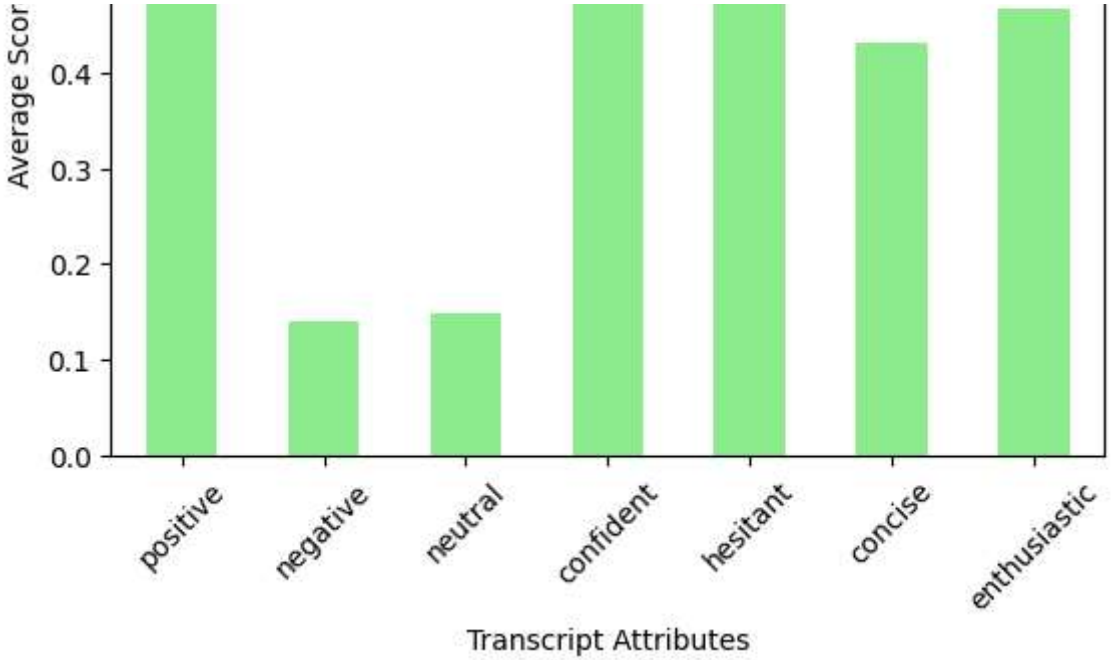

Average Transcript Scores for 6
0.4
0.5
0.6
0.7

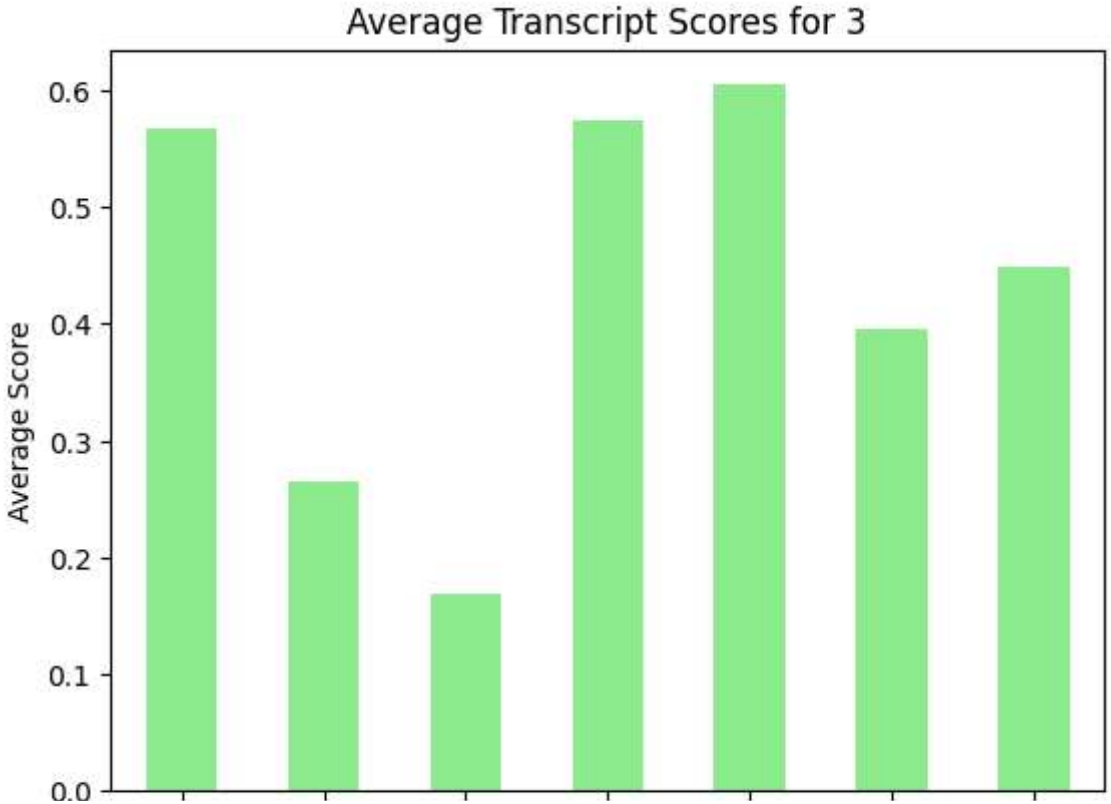

0.3
0.2
0.1
0.0
positive
negative
neutral
confident
hesitant
concise
enthusiastic
Transcript Attributes


Average Transcript Scores for 9
Average Score
0.6
0.5
0.4
0.3
0.2
0.1
0.0
concise

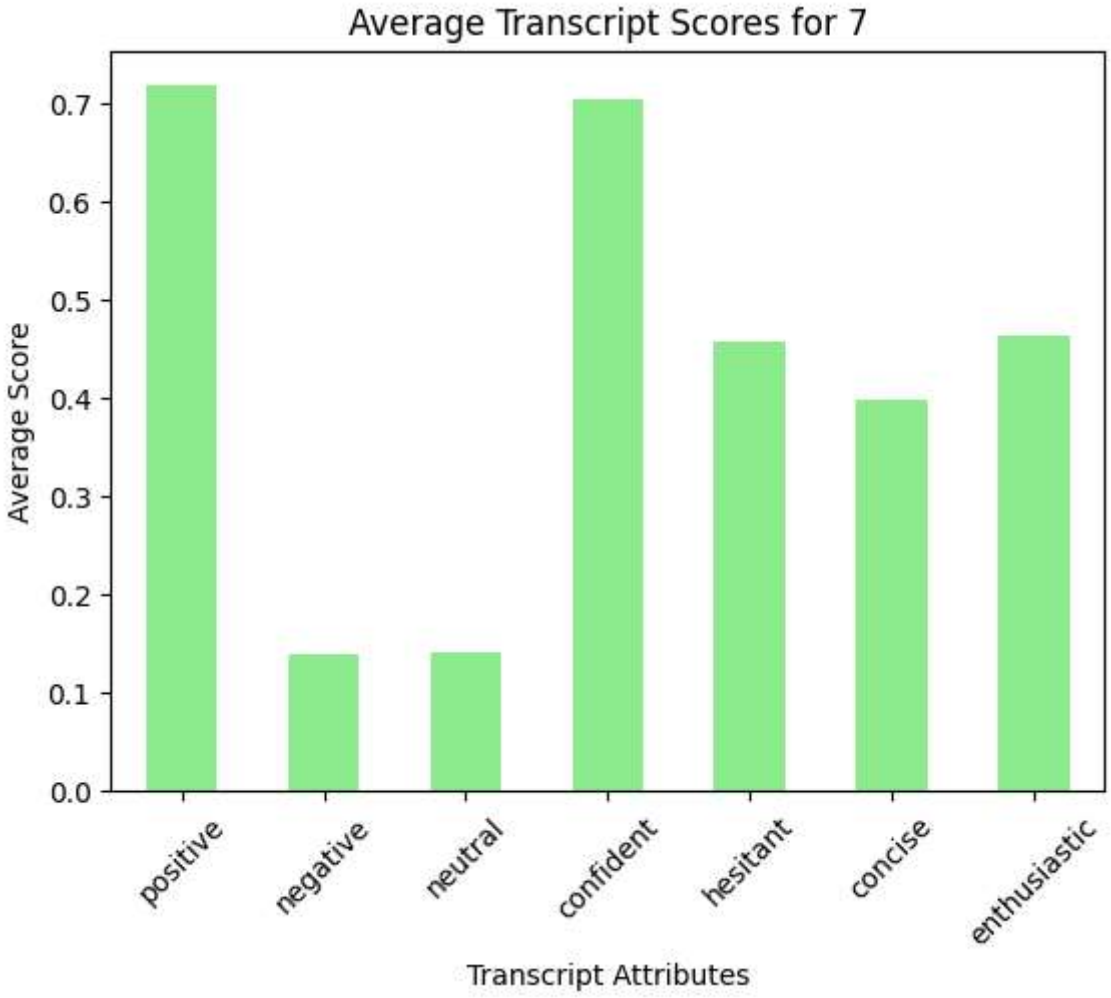

Transcript Attributes
Average Transcript Scores for 5
Average Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
positive
negative
neutral
confident
hesitant
concise
enthusiastic
Transcript Attributes


Average Transcript Scores for 1
0.5
0.6
0.7

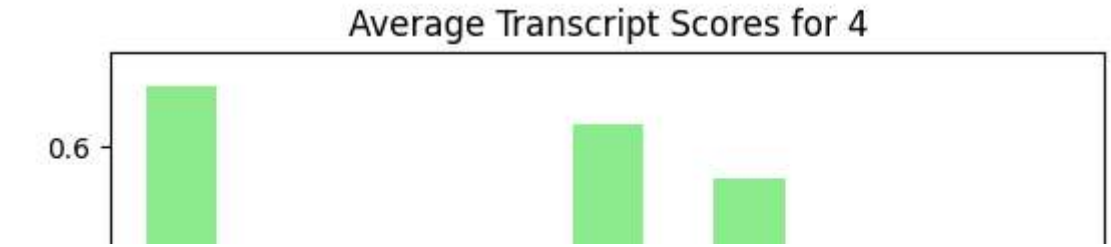

Average Scor
0.0
0.1
0.2
0.3
0.4
positive
negative
neutral
confident
hesitant
concise
enthusiastic
Transcript Attributes

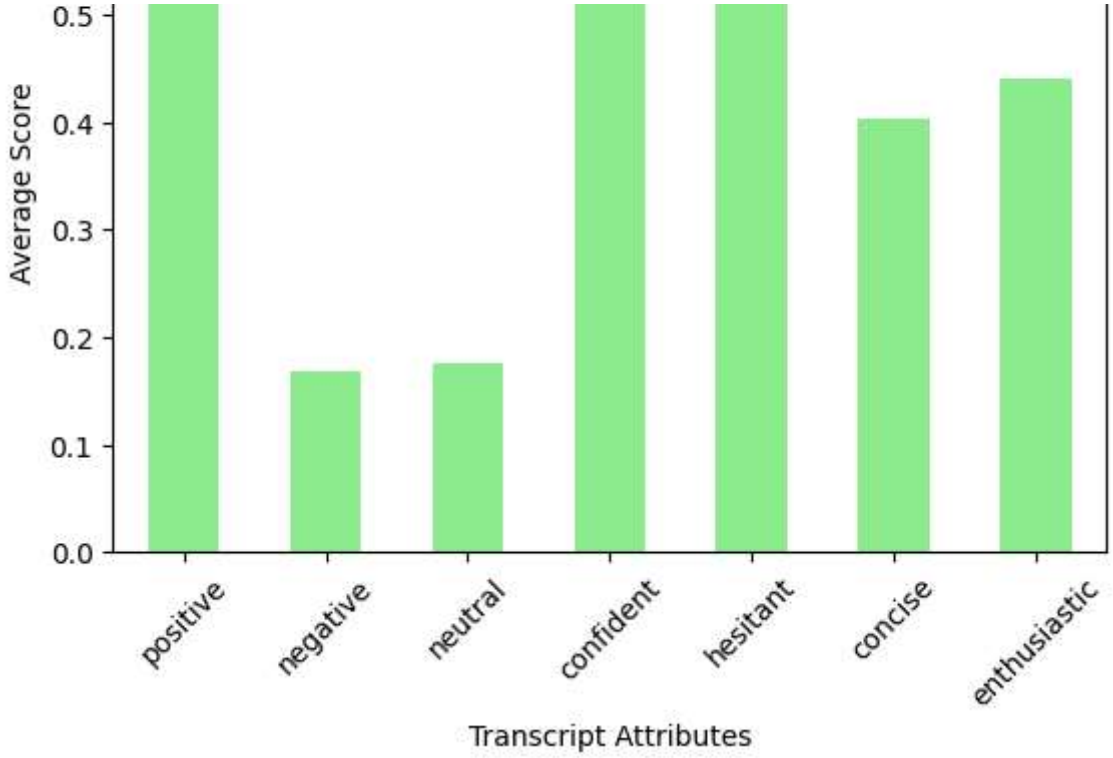

Average Transcript Scores for 3
Average Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6

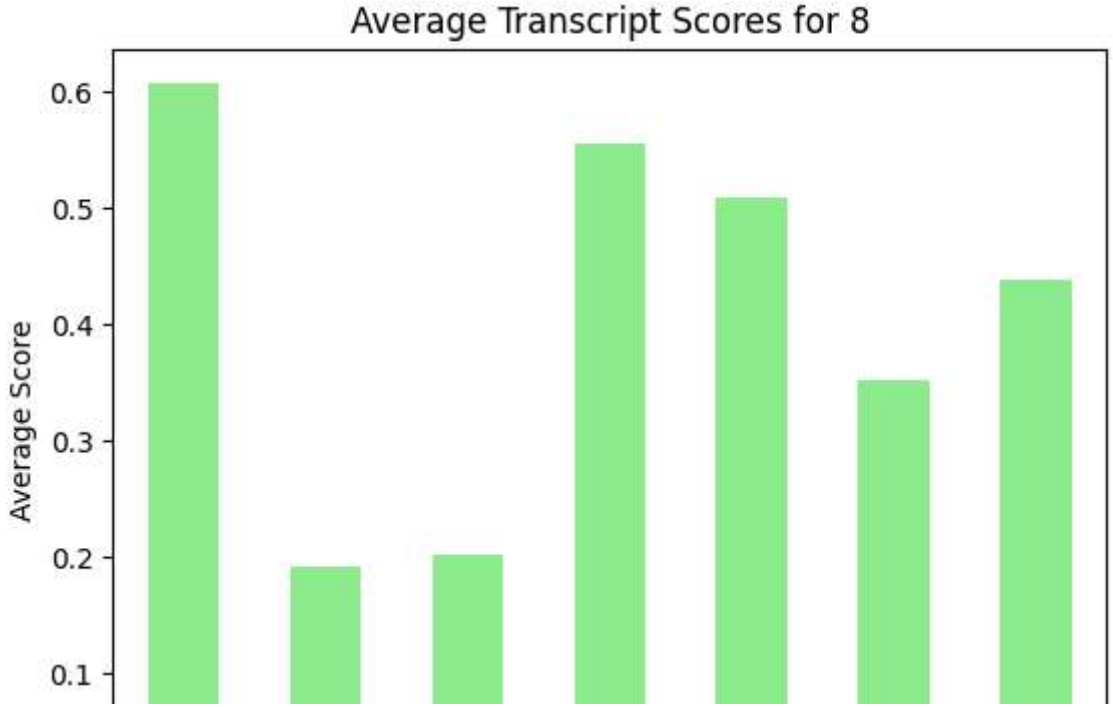
Transcript Attributes

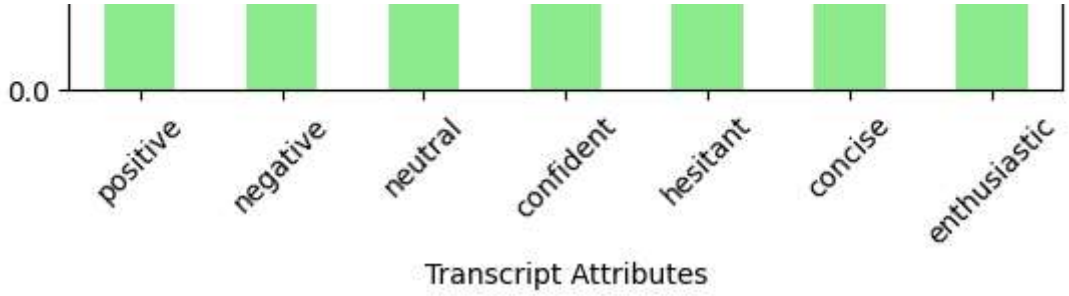


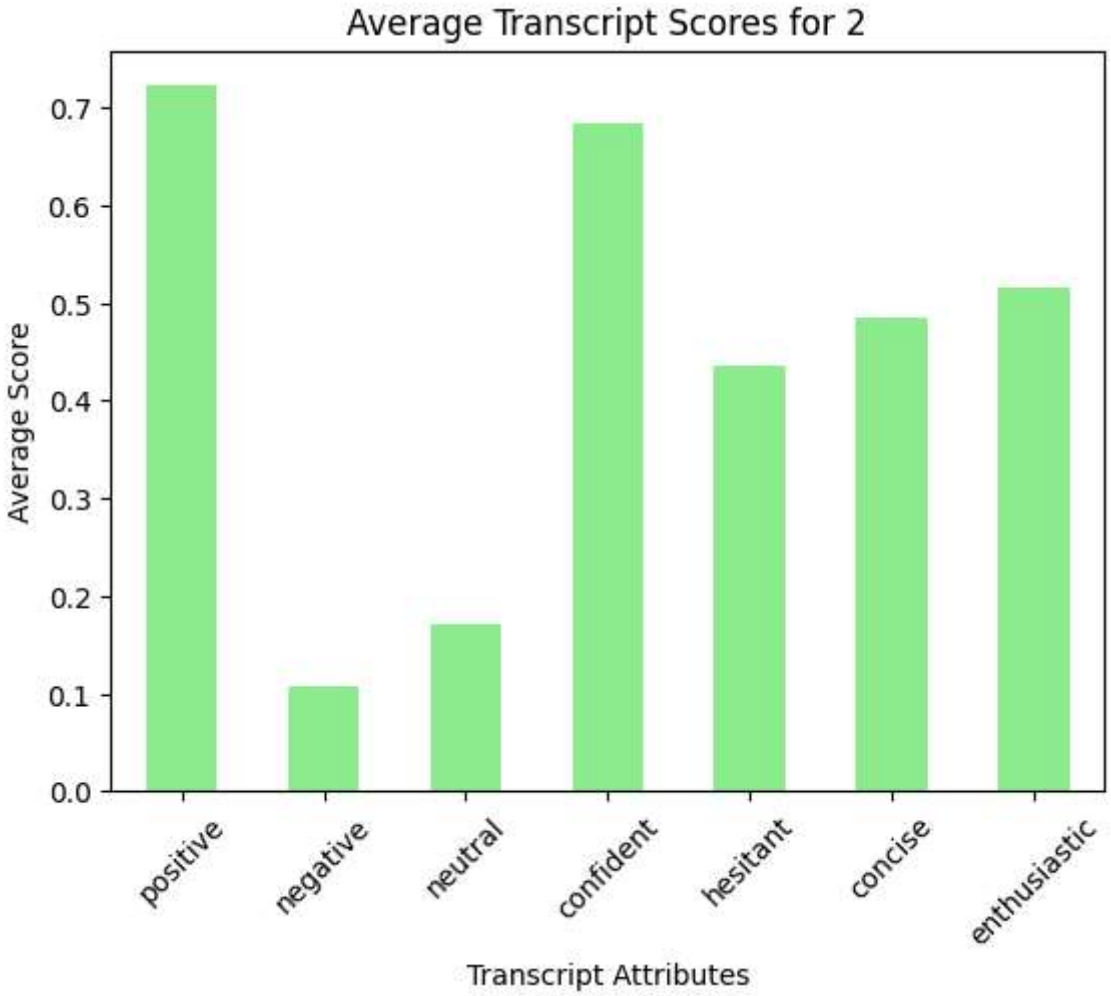
Transcript Attributes



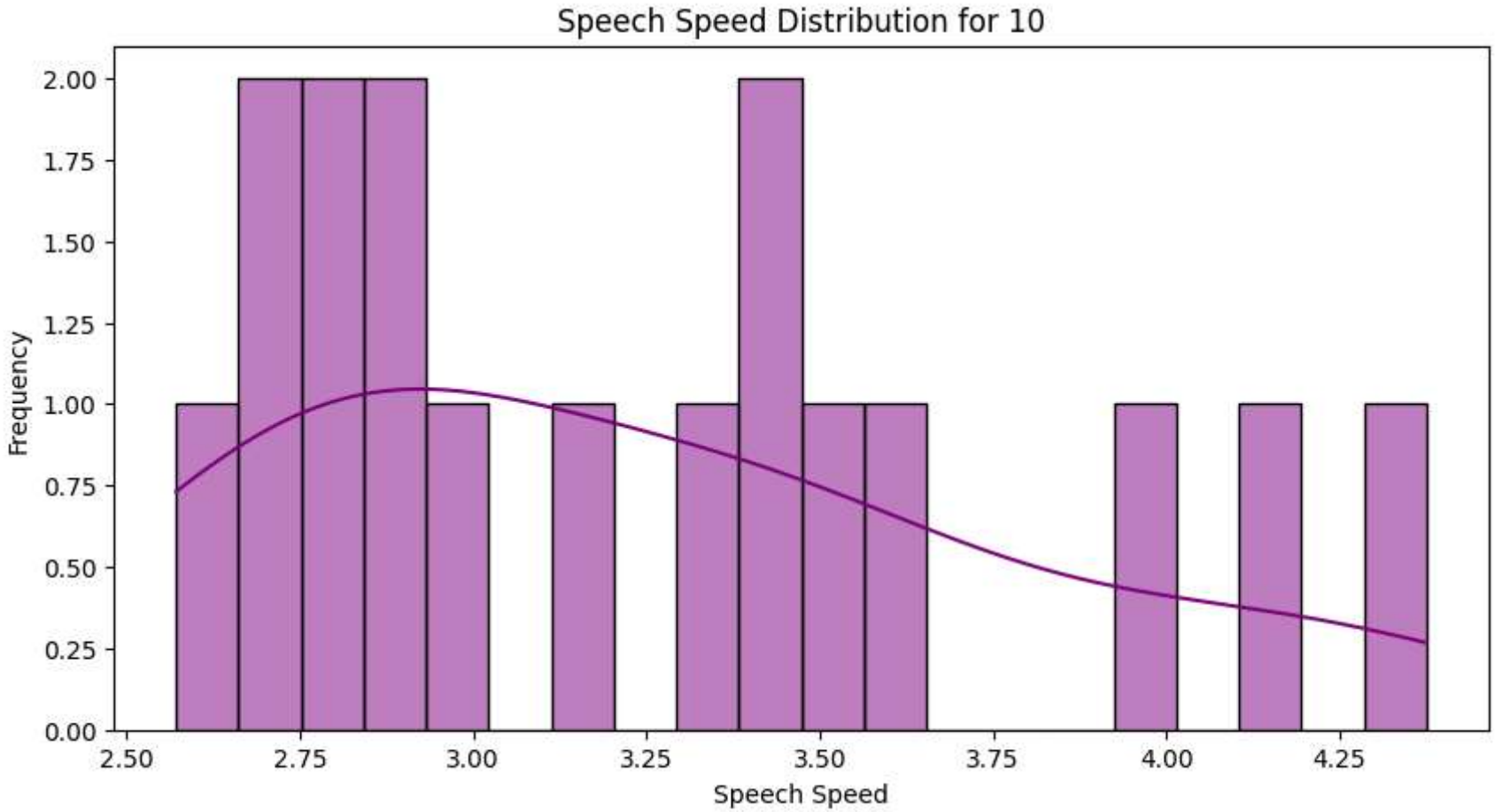

Average Score
0.0
0.1
0.2
0.3
0.4
0.5
positive
negative
neutral
confident
hesitant
concise
enthusiastic
Transcript Attributes

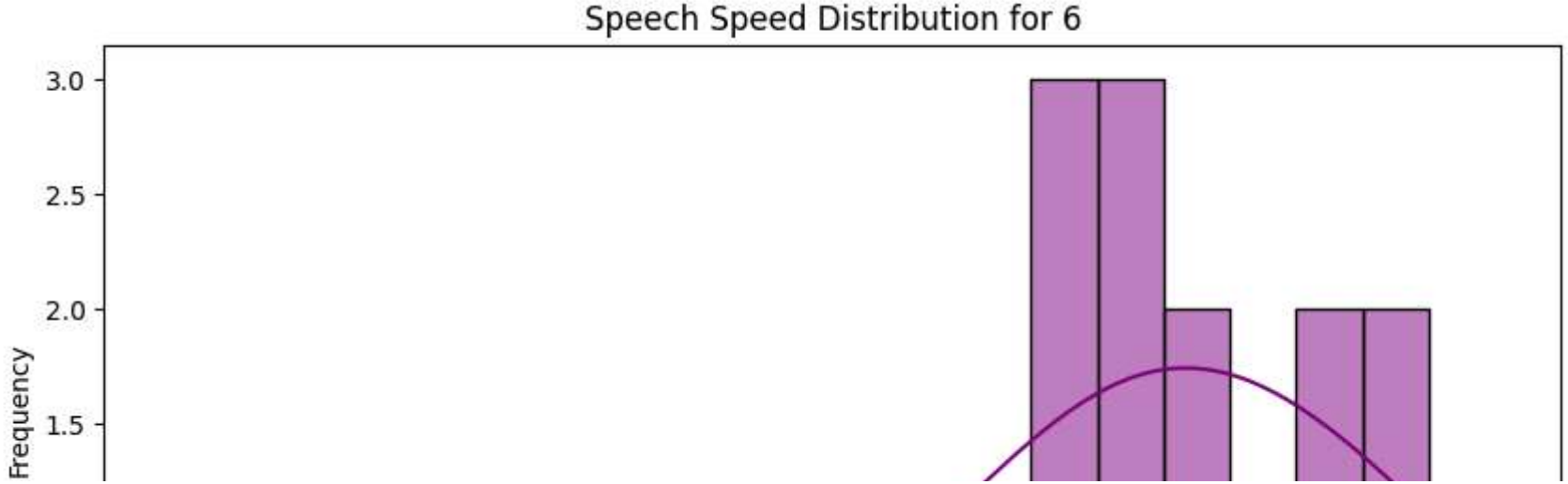

Average Transcript Scores for 8
Average Score
0.1
0.2
0.3
0.4
0.5
0.6

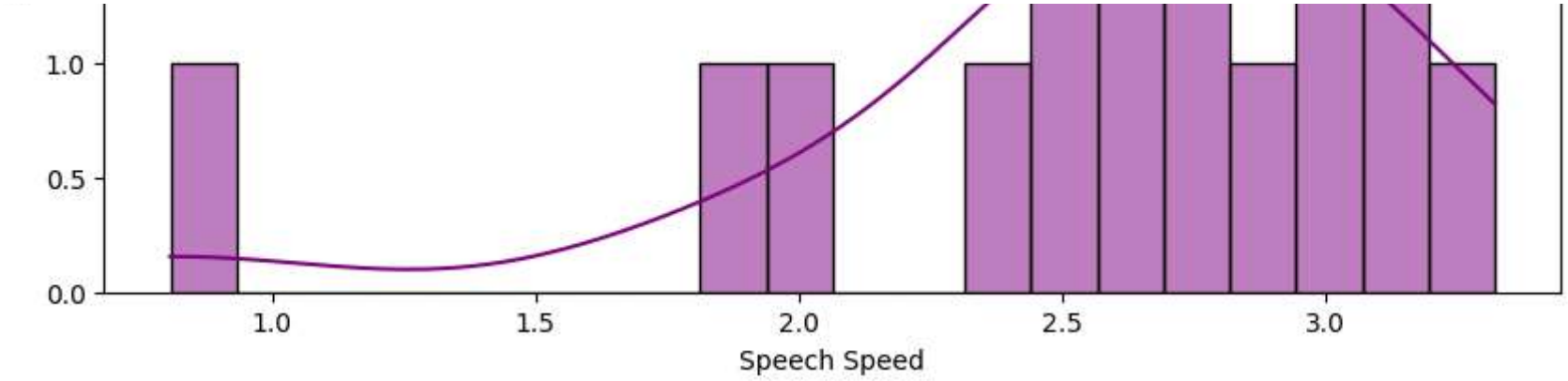

0.0
positive
negative
neutral
confident
hesitant
concise
enthusiastic
Transcript Attributes

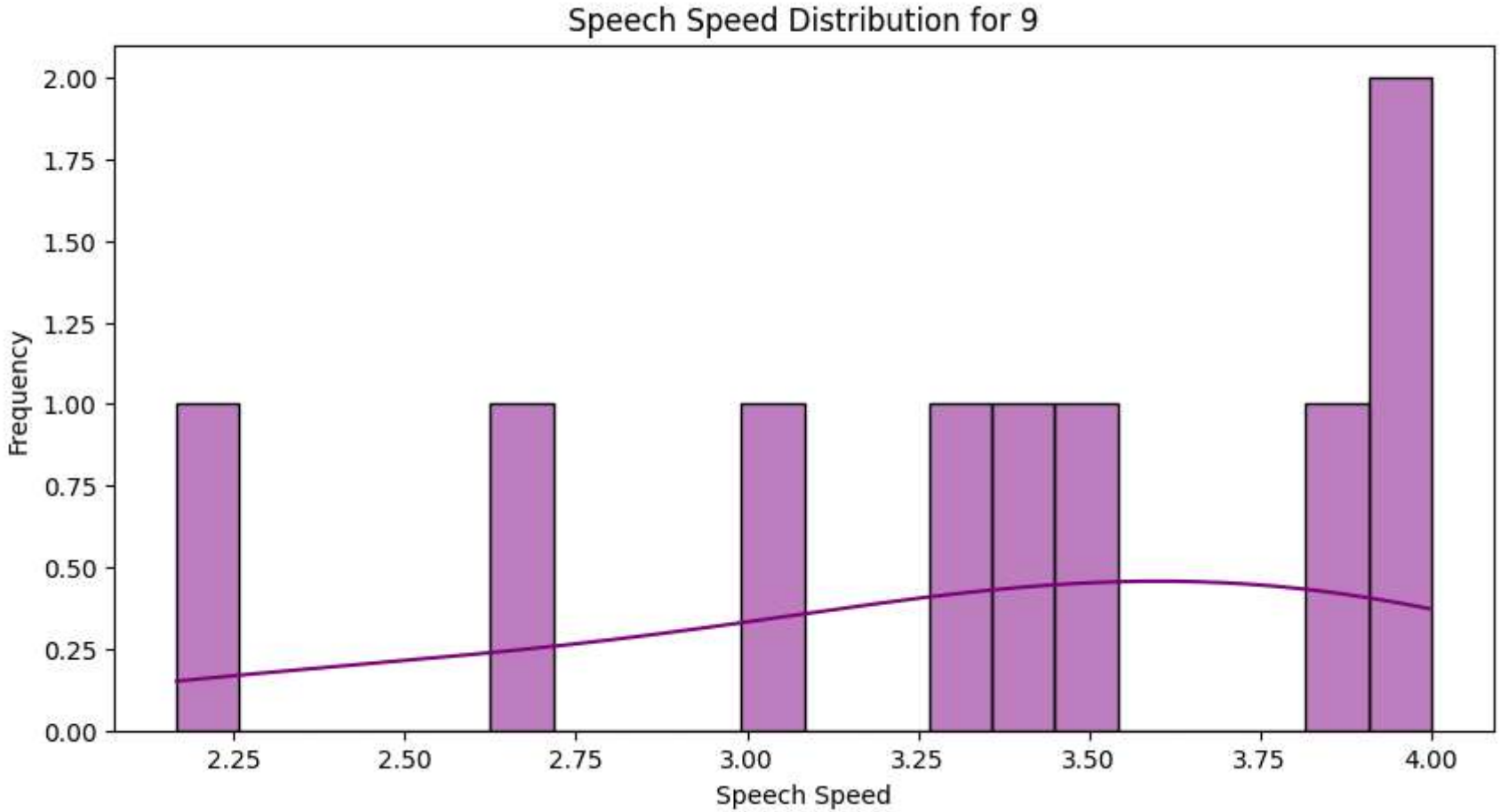

Average Transcript Scores for 2
Average Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
positive
negative
neutral
confident
hesitant
concise
enthusiastic
Transcript Attributes

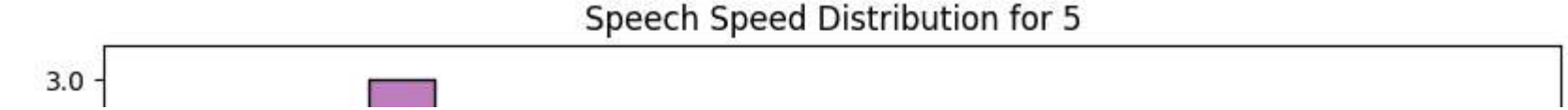
```
# Function to analyze speech speed
def speech_speed_analysis(transcript_df, candidate_id):
    plt.figure(figsize=(10, 5))
    sns.histplot(transcript_df['speech_speed'], bins=20, kde=True, color='purple')
    plt.title(f'Speech Speed Distribution for {candidate_id}')
    plt.xlabel('Speech Speed')
    plt.ylabel('Frequency')
    plt.show()

# Analyze for each candidate
for candidate_id, df in transcript_dfs.items():
    speech_speed_analysis(df, candidate_id)
```

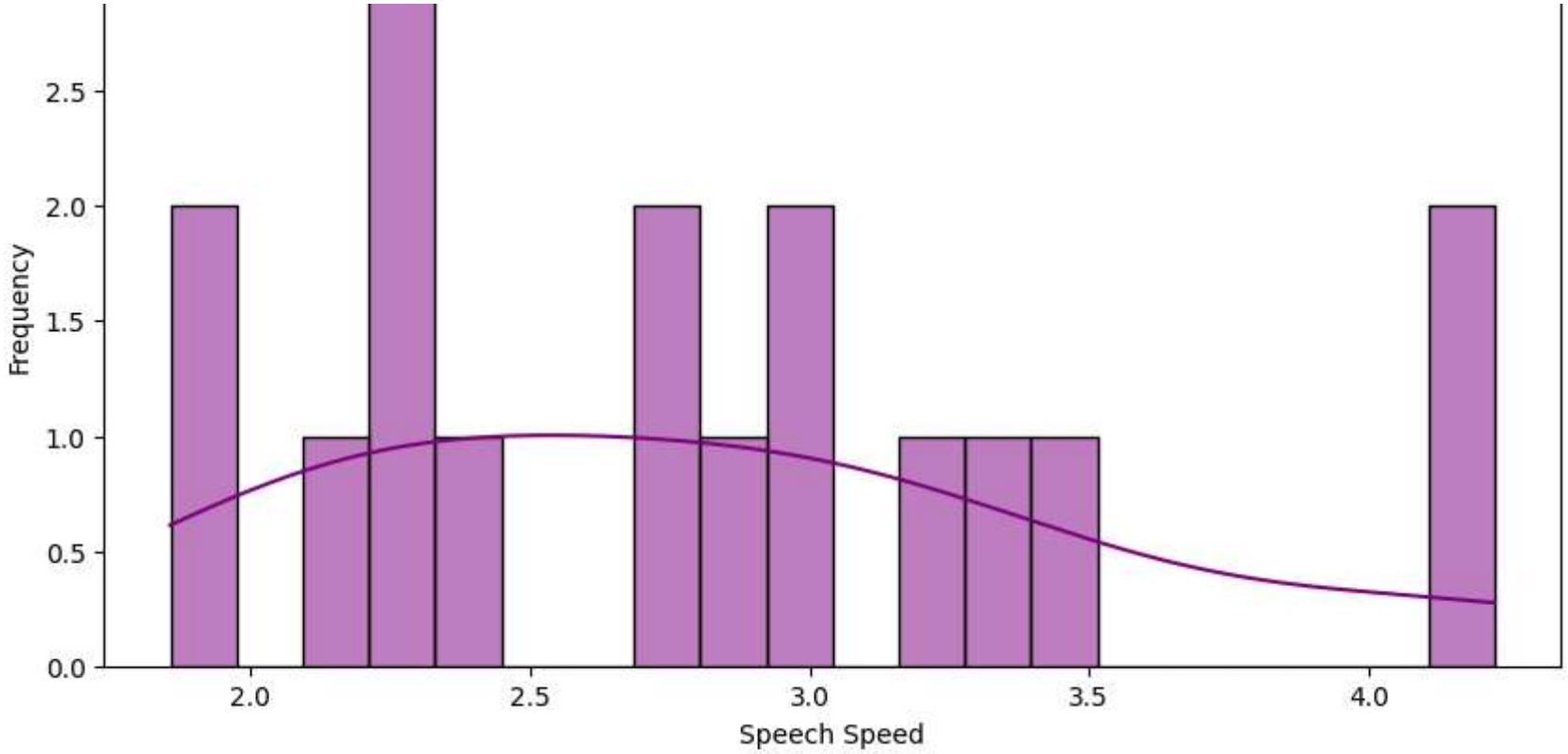

Speech Speed Distribution for 10
Frequency
Speech Speed
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2.50
2.75
3.00
3.25
3.50
3.75
4.00
4.25

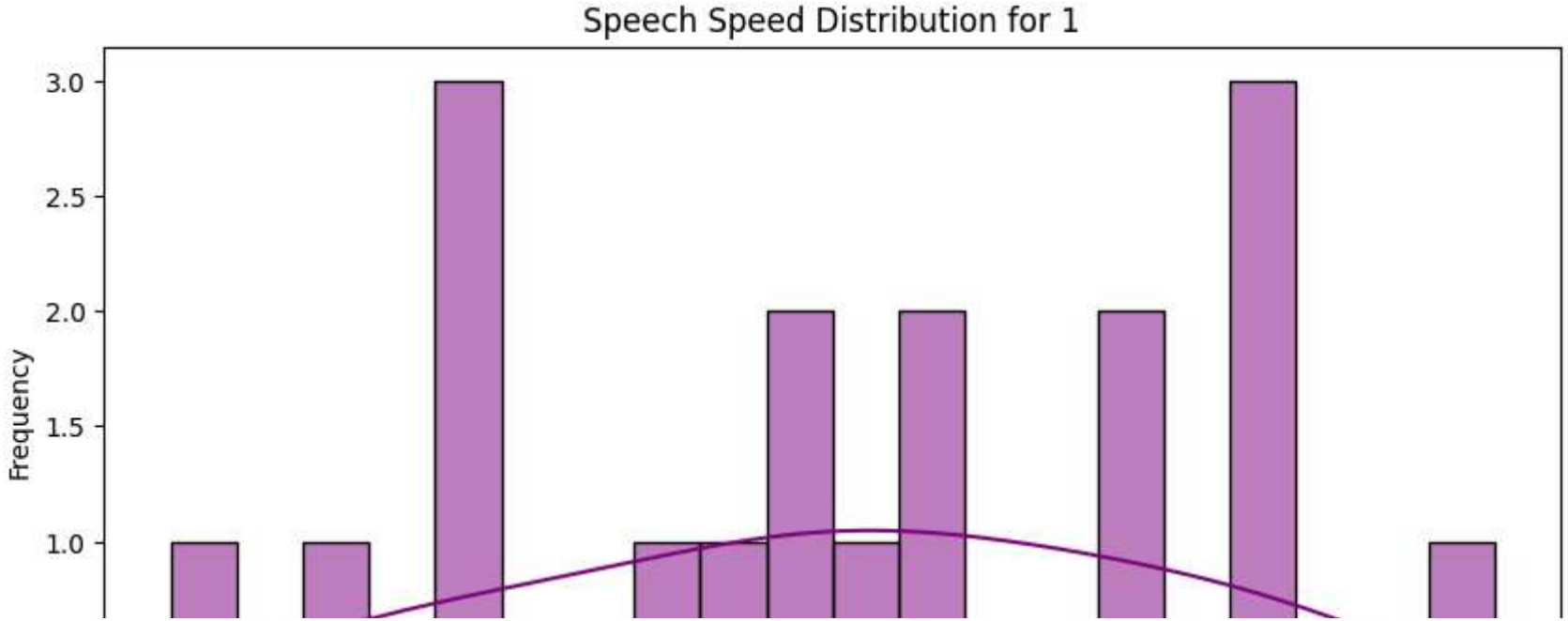

Speech Speed Distribution for 6
Frequency
1.5
2.0
2.5
3.0

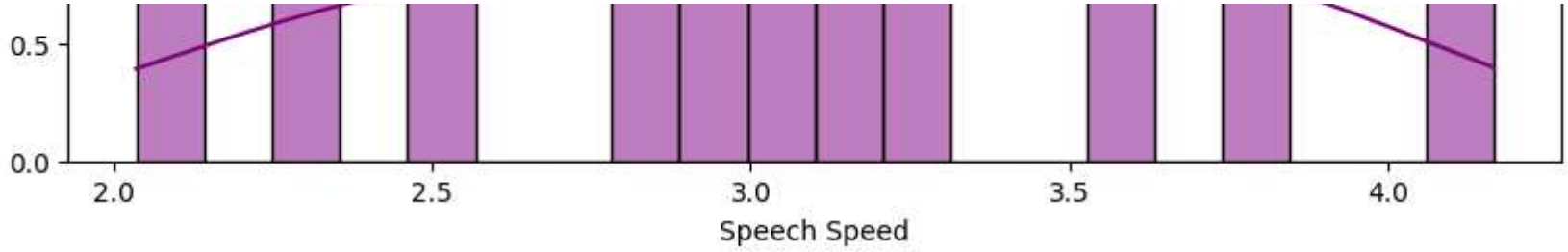

1.0
0.5
0.0
1.0
1.5
2.0
2.5
3.0
Speech Speed

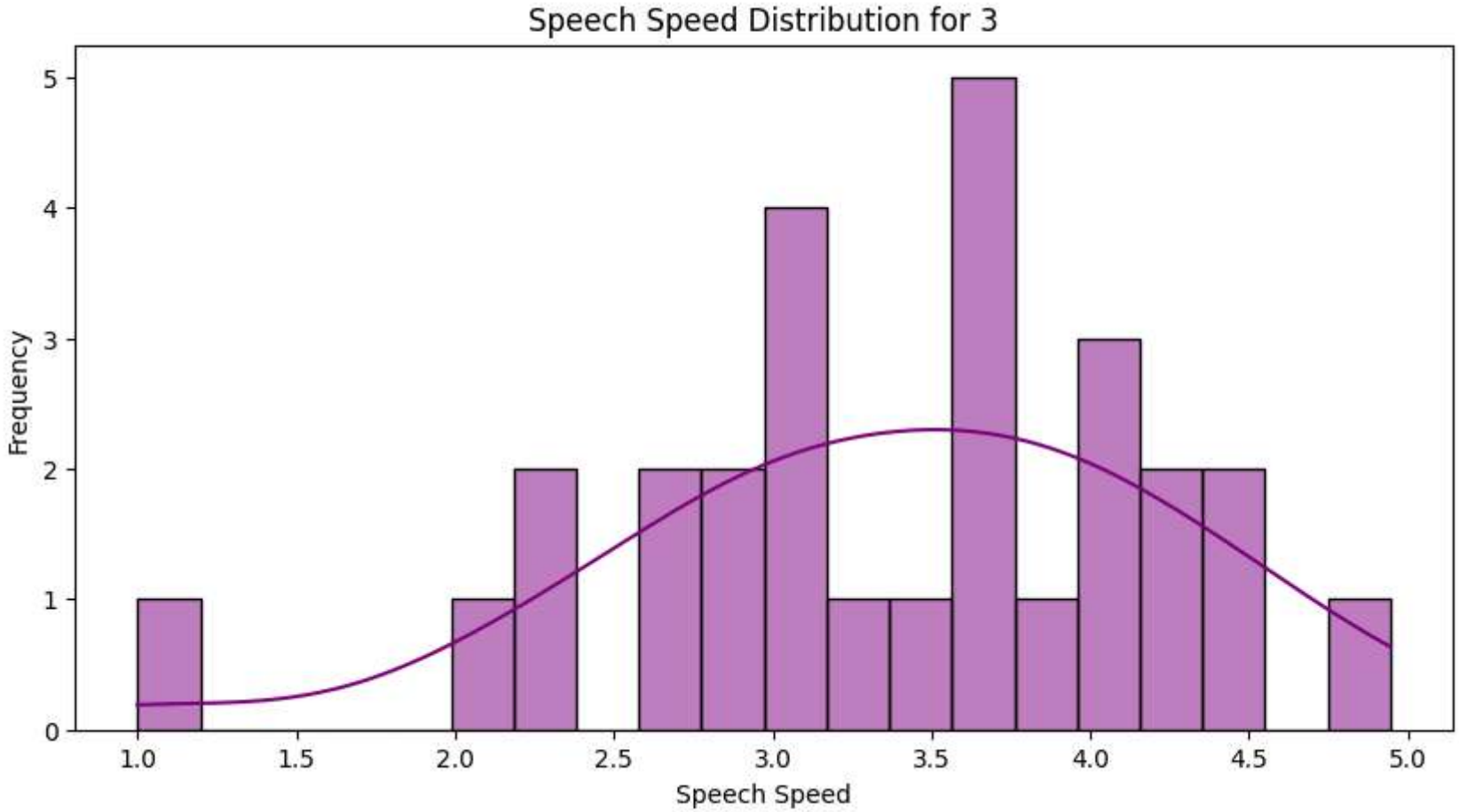

Speech Speed Distribution for 9
Frequency
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2.25
2.50
2.75
3.00
3.25
3.50
3.75
4.00
Speech Speed

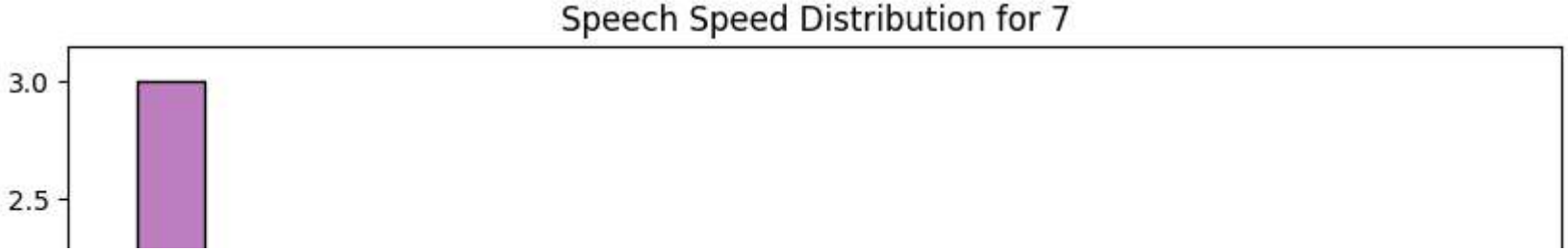

Speech Speed Distribution for 5
3.0

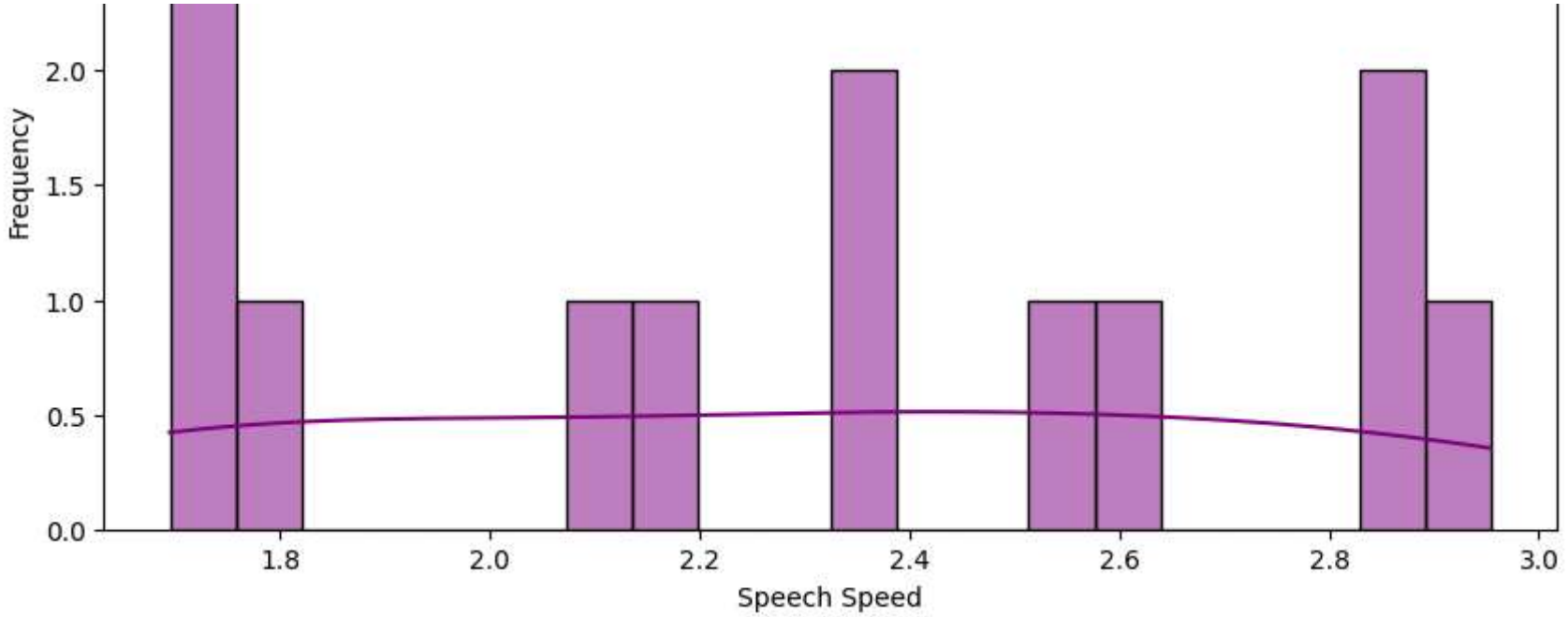

Frequency
Speech Speed
0.0
0.5
1.0
1.5
2.0
2.5
2.0
2.5
3.0
3.5
4.0

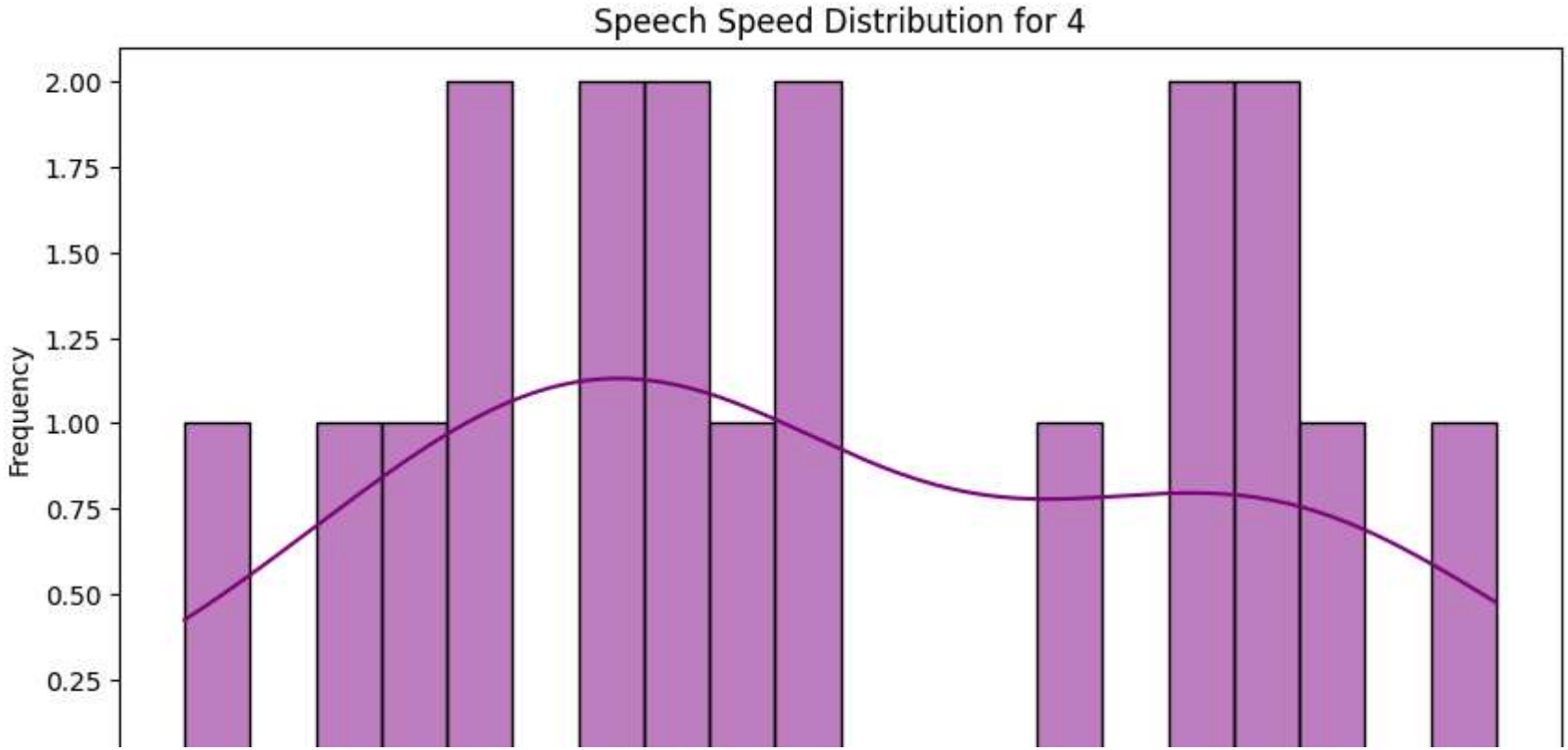

Speech Speed Distribution for 1
Frequency
1.0
1.5
2.0
2.5
3.0


0.5
0.0
2.0
2.5
3.0
3.5
4.0
Speech Speed

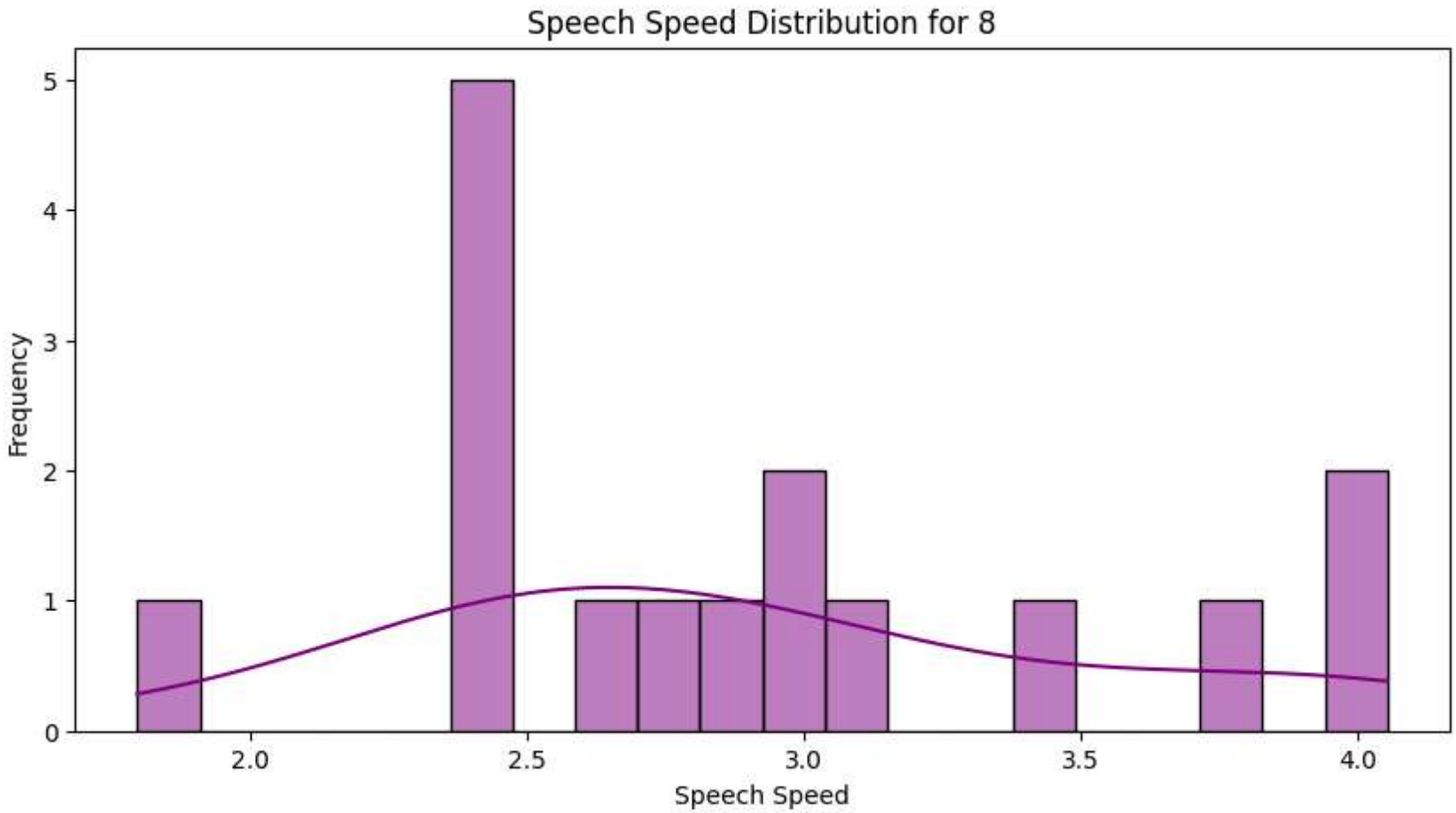

Speech Speed Distribution for 3
Frequency
0
1
2
3
4
5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Speech Speed

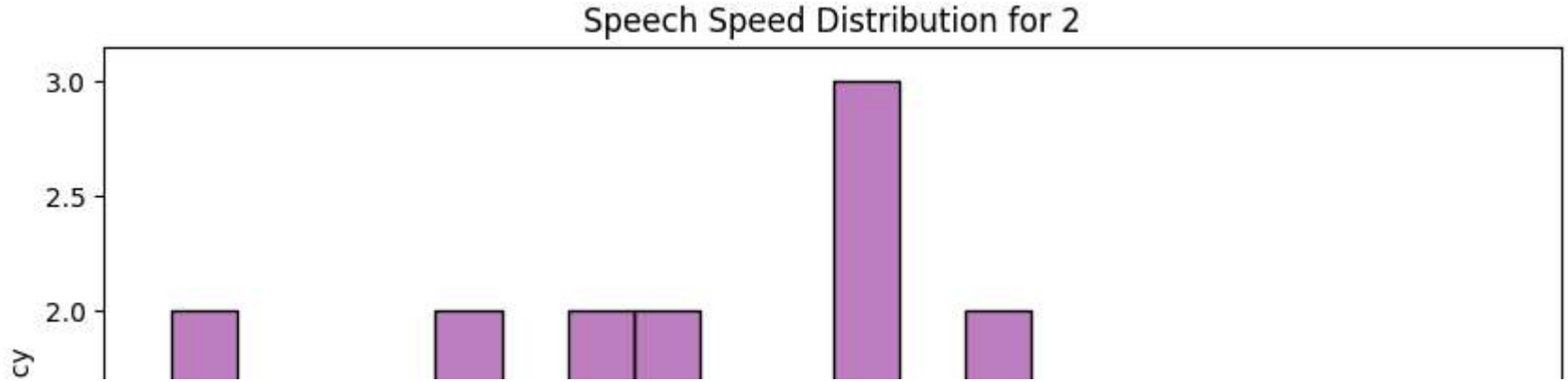

Speech Speed Distribution for 7
3.0
2.5

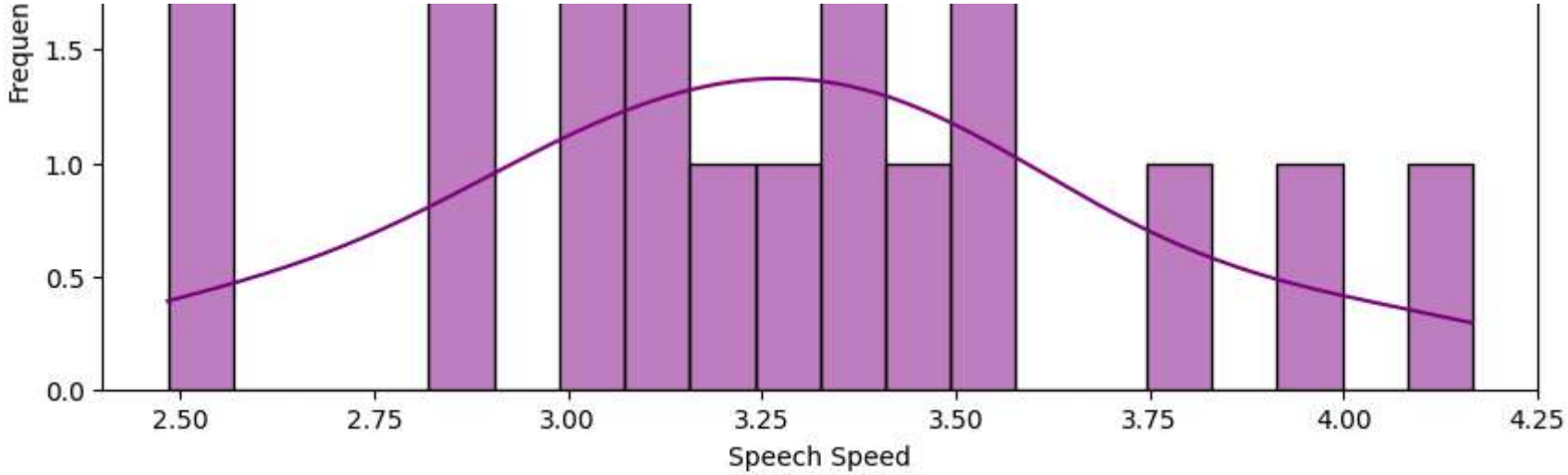

Frequency
0.0
0.5
1.0
1.5
2.0
1.8
2.0
2.2
2.4
2.6
2.8
3.0
Speech Speed


Speech Speed Distribution for 4
Frequency
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


0.00
2.2
2.4
2.6
2.8
3.0
3.2
3.4
Speech Speed


Speech Speed Distribution for 8
Frequency
0
1
2
3
4
5
2.0
2.5
3.0
3.5
4.0
Speech Speed


Speech Speed Distribution for 2
2.0
2.5
3.0


Frequen
1.5
1.0
0.5
0.0
2.50
2.75
3.00
3.25
3.50
3.75
4.00
4.25
Speech Speed

```
import os
import nltk
from nltk import sent_tokenize
from nltk.sentiment import SentimentIntensityAnalyzer

# Download the necessary NLTK data
nltk.download('vader_lexicon')
nltk.download('punkt')

# Initialize the sentiment analyzer
sia = SentimentIntensityAnalyzer()

# Function to analyze transcripts
def analyze_transcript(transcript_path, candidate_id):
    with open(transcript_path, 'r') as f:
        original_transcript = f.read()

    sentences = sent_tokenize(original_transcript)
    print(f"Key Sentences from {candidate_id} Transcript:")

    # Display first 5 sentences and perform sentiment analysis
    for sentence in sentences[:5]:
        sentiment_scores = sia.polarity_scores(sentence)
        print(f"- {sentence} (Sentiment: {sentiment_scores})")

# Path to the Transcripts folder
transcripts_path = os.path.join(extraction_path, 'transcripts')

# Analyze for each candidate
for transcript_file in os.listdir(transcripts_path):
    candidate_id = transcript_file.split('.')[0]  # Assuming filename is candidate_id.txt
    transcript_path = os.path.join(transcripts_path, transcript_file)
    analyze_transcript(transcript_path, candidate_id)
```

```
I was the captain of the students' committee in my final school year and was also awarded the Best Student Award a
Key Sentences from 5 Transcript:
-  Hello, I'm Sakshi. (Sentiment: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0})
- I come from Mumbai. (Sentiment: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0})
- I did my undergraduation in mass media  with specialization in advertising. (Sentiment: {'neg': 0.0, 'neu': 1.0, '
- I have completed two  certification courses. (Sentiment: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0})
- One was entrepreneurship course by Turnip and  second one was foundations of management by Google. (Sentiment: {'n
Key Sentences from 9 Transcript:
-  Hello, myself is Alexander Smith. (Sentiment: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0})
- I am a first year MBA student here at IIM Lucknow. (Sentiment: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.
- I come from a suburban part of India. (Sentiment: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0})
- I did my B.Tech in Agriculture Engineering, my M.Tech in Food Process Engineering. (Sentiment: {'neg': 0.0, 'neu':
- During my M.Tech I co-founded an Agritech startup. (Sentiment: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.
Key Sentences from 1 Transcript:
-  Hello, I am Jeffrey Shepherd and I am currently pursuing postgraduate and management from  IIM Coikode. (Sentimen
- I have completed my B.Tech in Biotechnology from Heritage Institute of  Technology Kolkata, followed by my M.Tech
- I come with an experience  of three years in the regulatory affairs domain of the pharmaceutical industry and I wo
- What sets me apart is the expertise I bring in with my three years of experience  and an added two years of postgr
- Along with this, I add another dimension to  the discussion with my background in biotechnology. (Sentiment: {'neg
Key Sentences from 7 Transcript:
-  Hello, I am Joseph Nichols. (Sentiment: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0})
- I belong to the heritage city of Varanasi. (Sentiment: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0})
- I have done my undergraduation  in earth science from Banaras Hindu University. (Sentiment: {'neg': 0.0, 'neu': 1.
- And post that I have worked with the single  largest government owned reinsurer in the country called General Insu
- My thorough understanding  of the Indian markets and analytical skills acquired due to working in such a diverse
Key Sentences from 8 Transcript:
-  Hi, hope you're doing well. (Sentiment: {'neg': 0.0, 'neu': 0.375, 'pos': 0.625, 'compound': 0.6124})
- I'm Srivats Biyani. (Sentiment: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0})
- I am a PGP finance student at IIM Co-Ecode. (Sentiment: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0})
- Now before joining IIM Co-Ecode, what all have I done? (Sentiment: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound'
- I am a chartered accountant and  I've also cleared CFA level 1. (Sentiment: {'neg': 0.0, 'neu': 0.865, 'pos': 0.13
Key Sentences from 2 Transcript:
-  Hello, I am Beside You. (Sentiment: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0})
- I am Cameron Barajas and I am thrilled to apply to this opportunity  today. (Sentiment: {'neg': 0.0, 'neu': 0.619,
- I recently completed my BBA in 2022. (Sentiment: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0})
```

```
Also, my ongoing  post graduation in MBA analytics has taken me closer to the field of analytics  in business. (Se
Key Sentences from 4 Transcript:
-  Hello, my name is Monique Mccormick. (Sentiment: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0})
- I'm from Guntur, Andhra Pradesh. (Sentiment: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0})
- I'm an engineering  graduate in electronics and communication field. (Sentiment: {'neg': 0.0, 'neu': 1.0, 'pos': 0
- My final year project is on performance  analysis of shift resistors. (Sentiment: {'neg': 0.0, 'neu': 1.0, 'pos':
- I qualified GATE and secured a rank of 5300. (Sentiment: {'neg': 0.0, 'neu': 0.69, 'pos': 0.31, 'compound': 0.4019
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
```

```
# Summary of Findings
summary = {
    "Candidates": {},
}

for candidate_id in emotion_dfs.keys():
    # Store insights for each candidate
    summary["Candidates"][candidate_id] = {
        "Average Emotions": emotion_dfs[candidate_id][['happy', 'sad', 'angry', 'fear', 'surprise']].mean().to_dict(),
        "Dominant Emotion": emotion_dfs[candidate_id]['dominant_emotion'].mode()[0],
        "Average Positive Score": transcript_dfs[candidate_id]['positive'].mean(),
        "Average Speech Speed": transcript_dfs[candidate_id]['speech_speed'].mean(),
    }

# Display the summary
print(summary)
```

```
{'Candidates': {'2': {'Average Emotions': {'happy': 35.063288345977014, 'sad': 18.55839076, 'angry': 10.395040800995405
```

```
import pandas as pd

# Summary dictionary
summary = {
    "Candidates": {},
}
```

```
# Collect key findings for each candidate
for candidate_id in emotion_dfs.keys():
    summary["Candidates"][candidate_id] = {
        "Average Happy": emotion_dfs[candidate_id]['happy'].mean(),
        "Average Sad": emotion_dfs[candidate_id]['sad'].mean(),
        "Average Angry": emotion_dfs[candidate_id]['angry'].mean(),
        "Average Fear": emotion_dfs[candidate_id]['fear'].mean(),
        "Average Surprise": emotion_dfs[candidate_id]['surprise'].mean(),
        "Dominant Emotion": emotion_dfs[candidate_id]['dominant_emotion'].mode()[0],
        "Average Positive Score": transcript_dfs[candidate_id]['positive'].mean(),
        "Average Speech Speed": transcript_dfs[candidate_id]['speech_speed'].mean(),
    }

# Convert the summary into a DataFrame for better readability
summary_df = pd.DataFrame.from_dict(summary["Candidates"], orient='index')

# Display the DataFrame
print(summary_df)
```

```
    Average Happy  Average Sad  Average Angry  Average Fear  Average Surprise  \
2       35.063288    18.558391      10.395041      6.747536          2.267330
9       16.734190     4.293063       6.337654     18.602680         15.761101
6       22.363658     0.436512       0.004150      0.035453          0.004707
8        2.120367     1.955231       8.115339     11.939264          1.901795
4        0.572650     1.057942       1.734140      2.602291          1.403702
10       4.215283    32.656818       3.856539     36.143804          4.330881
1        5.865318    13.575324      14.451059     18.382797          8.744969
7        8.994113    23.106425       5.641183     41.652400          4.081041
3       21.428420    10.295898       1.531082     21.633298          7.268728
5        0.101414     0.654111       6.261661      0.202875          0.008069

   Dominant Emotion  Average Positive Score  Average Speech Speed
2             happy                0.722006              3.269092
9           neutral                0.617353              3.329938
6           neutral                0.711182              2.583163
8           neutral                0.605402              2.902953
4           neutral                0.655748              2.775454
10             fear                0.589267              3.248518
1           neutral                0.709199              3.113771
7              fear                0.717354              2.284897
```

```
3           neutral               0.567257              3.385636
5           neutral               0.630573              2.817341
```

```
# Convert the summary into a DataFrame
summary_df = pd.DataFrame.from_dict(summary["Candidates"], orient='index')

# Save the DataFrame to a CSV file
csv_path = os.path.join(extraction_path, 'candidate_summary.csv')
summary_df.to_csv(csv_path, index=True)

# Display a confirmation message
print(f"Summary saved as CSV at: {csv_path}")
```

```
Summary saved as CSV at: /content/data/candidate_summary.csv
```

```
# Sample data structure for candidates
candidate_summary = [
    {"Candidate ID": "C1", "Average Happy": 0.6, "Average Sad": 0.1, "Average Angry": 0.1,
     "Average Fear": 0.1, "Average Surprise": 0.4, "Dominant Emotion": "Happy",
     "Average Positive Score": 0.7, "Average Speech Speed": 150},
    # Add other candidates here...
]

# Function to evaluate candidates
def evaluate_candidates(candidate_summary):
    results = []

    for candidate in candidate_summary:
        decision = "Not Recommended"
        reasons = []

        # Evaluate emotional scores
        if (candidate["Average Happy"] > 0.5 and
            candidate["Average Sad"] < 0.2 and
            candidate["Average Angry"] < 0.2 and
            candidate["Average Fear"] < 0.2 and
            candidate["Average Surprise"] > 0.3):
```

```
                reasons.append("Positive emotional scores.")

            # Evaluate dominant emotion
            if candidate["Dominant Emotion"] == "Happy":
                reasons.append("Dominant emotion is happy.")

            # Evaluate positive score
            if candidate["Average Positive Score"] > 0.5:
                reasons.append("High average positive score.")

            # Evaluate speech speed
            if 120 <= candidate["Average Speech Speed"] <= 160:
                reasons.append("Speech speed is within the ideal range.")

            # Make recruitment decision based on reasons
            if len(reasons) >= 3:  # Threshold for enough positive reasons
                decision = "Recommended"

            results.append({"Candidate ID": candidate["Candidate ID"], "Decision": decision, "Reasons": reasons})

        return results

    # Evaluate candidates
    evaluation_results = evaluate_candidates(candidate_summary)

    # Display results
    for result in evaluation_results:
        print(f"Candidate ID: {result['Candidate ID']}, Decision: {result['Decision']}, Reasons: {', '.join(result['Reasons'])}
```

```
Candidate ID: C1, Decision: Recommended, Reasons: Positive emotional scores., Dominant emotion is happy., High average
```

```
    import pandas as pd
    import os
    from nltk import sent_tokenize
    from nltk.sentiment import SentimentIntensityAnalyzer

    # Initialize the sentiment analyzer
```

```
    sia = SentimentIntensityAnalyzer()

    # Function to analyze transcripts and return sentiment scores
    def analyze_transcript(transcript_path):
        with open(transcript_path, 'r') as f:
            original_transcript = f.read()

        sentences = sent_tokenize(original_transcript)
        sentiments = [sia.polarity_scores(sentence) for sentence in sentences]

        # Aggregate sentiment scores
        avg_sentiment = {
            'positive': sum(score['pos'] for score in sentiments) / len(sentiments),
            'negative': sum(score['neg'] for score in sentiments) / len(sentiments),
            'neutral': sum(score['neu'] for score in sentiments) / len(sentiments)
        }

        return avg_sentiment

    # Function to evaluate candidates based on sentiment scores
    def evaluate_candidates(transcripts_path, candidate_summary):
        results = []

        for candidate in candidate_summary:
            candidate_id = candidate["Candidate ID"]
            transcript_path = os.path.join(transcripts_path, f"{candidate_id}.txt")

            # Analyze transcript for the candidate
            avg_sentiment = analyze_transcript(transcript_path)

            # Evaluate based on thresholds
            decision = "Not Recommended"
            reasons = []

            # Sentiment thresholds
            if avg_sentiment['positive'] > 0.5:
                reasons.append("High positive sentiment.")
            if avg_sentiment['negative'] < 0.2:
                reasons.append("Low negative sentiment.")
```

```
            if avg_sentiment['neutral'] < 0.5:
                reasons.append("Balanced emotional expression.")

            # Make recruitment decision based on reasons
            if len(reasons) >= 2:  # Threshold for enough positive reasons
                decision = "Recommended"

            results.append({
                "Candidate ID": candidate_id,
                "Decision": decision,
                "Average Positive Score": avg_sentiment['positive'],
                "Average Negative Score": avg_sentiment['negative'],
                "Average Neutral Score": avg_sentiment['neutral'],
                "Reasons": reasons
            })

        return results

    # Example candidate summary
    candidate_summary = [
        {"Candidate ID": "1"},
        {"Candidate ID": "2"},
        {"Candidate ID": "3"},
        {"Candidate ID": "4"},
        {"Candidate ID": "5"},
        {"Candidate ID": "6"},
        {"Candidate ID": "7"},
        {"Candidate ID": "8"},
        {"Candidate ID": "9"},
        {"Candidate ID": "10"},
        # Add other candidates here...
    ]

    # Path to the Transcripts folder
    transcripts_path = os.path.join('/content/data/transcripts')

    # Evaluate candidates
    evaluation_results = evaluate_candidates(transcripts_path, candidate_summary)
```

```
# Display results
for result in evaluation_results:
    print(f"Candidate ID: {result['Candidate ID']}, Decision: {result['Decision']}, "
          f"Average Positive Score: {result['Average Positive Score']:.2f}, "
          f"Average Negative Score: {result['Average Negative Score']:.2f}, "
          f"Average Neutral Score: {result['Average Neutral Score']:.2f}, "
          f"Reasons: {', '.join(result['Reasons'])}")
```

```
Candidate ID: 1, Decision: Not Recommended, Average Positive Score: 0.18, Average Negative Score: 0.01, Average Neutral
Candidate ID: 2, Decision: Not Recommended, Average Positive Score: 0.17, Average Negative Score: 0.00, Average Neutral
Candidate ID: 3, Decision: Not Recommended, Average Positive Score: 0.17, Average Negative Score: 0.02, Average Neutral
Candidate ID: 4, Decision: Not Recommended, Average Positive Score: 0.14, Average Negative Score: 0.01, Average Neutral
Candidate ID: 5, Decision: Not Recommended, Average Positive Score: 0.14, Average Negative Score: 0.00, Average Neutral
Candidate ID: 6, Decision: Not Recommended, Average Positive Score: 0.14, Average Negative Score: 0.00, Average Neutral
Candidate ID: 7, Decision: Not Recommended, Average Positive Score: 0.09, Average Negative Score: 0.01, Average Neutral
Candidate ID: 8, Decision: Not Recommended, Average Positive Score: 0.16, Average Negative Score: 0.02, Average Neutral
Candidate ID: 9, Decision: Not Recommended, Average Positive Score: 0.14, Average Negative Score: 0.00, Average Neutral
Candidate ID: 10, Decision: Not Recommended, Average Positive Score: 0.13, Average Negative Score: 0.01, Average Neutra
```

Weighted Scoriing

```
import pandas as pd
import os
from nltk import sent_tokenize
from nltk.sentiment import SentimentIntensityAnalyzer

# Initialize the sentiment analyzer
sia = SentimentIntensityAnalyzer()

# Function to analyze transcripts and return sentiment scores
def analyze_transcript(transcript_path):
    with open(transcript_path, 'r') as f:
        original_transcript = f.read()

    sentences = sent_tokenize(original_transcript)
    sentiments = [sia.polarity_scores(sentence) for sentence in sentences]
```

```
        # Aggregate sentiment scores
        avg_sentiment = {
            'positive': sum(score['pos'] for score in sentiments) / len(sentiments),
            'negative': sum(score['neg'] for score in sentiments) / len(sentiments),
            'neutral': sum(score['neu'] for score in sentiments) / len(sentiments)
        }

        return avg_sentiment

    # Function to evaluate candidates based on sentiment scores
    def evaluate_candidates(transcripts_path, candidate_summary):
        results = []

        for candidate in candidate_summary:
            candidate_id = candidate["Candidate ID"]
            transcript_path = os.path.join(transcripts_path, f"{candidate_id}.txt")

            # Analyze transcript for the candidate
            avg_sentiment = analyze_transcript(transcript_path)

            # Store results for comparison later
            results.append({
                "Candidate ID": candidate_id,
                "Average Positive Score": avg_sentiment['positive'],
                "Average Negative Score": avg_sentiment['negative'],
                "Average Neutral Score": avg_sentiment['neutral']
            })

        # Determine the best candidate based on scores
        best_candidate = max(results, key=lambda x: x['Average Positive Score'])

        # Set dynamic thresholds
        positive_threshold = best_candidate['Average Positive Score'] * 0.8
        negative_threshold = best_candidate['Average Negative Score'] * 1.5  # Loosen negative threshol

        final_results = []

        for result in results:
            decision = "Not Recommended"
```

```
            reasons = []

            # Evaluate based on dynamic thresholds
            if result['Average Positive Score'] >= positive_threshold:
                reasons.append("High positive sentiment.")
                decision = "Recommended"
            if result['Average Negative Score'] <= negative_threshold:
                reasons.append("Low negative sentiment.")

            # Append to final results
            final_results.append({
                "Candidate ID": result['Candidate ID'],
                "Decision": decision,
                "Average Positive Score": result['Average Positive Score'],
                "Average Negative Score": result['Average Negative Score'],
                "Average Neutral Score": result['Average Neutral Score'],
                "Reasons": reasons
            })

        return final_results

    # Example candidate summary
    candidate_summary = [
        {"Candidate ID": "1"},
        {"Candidate ID": "2"},
        {"Candidate ID": "3"},
        {"Candidate ID": "4"},
        {"Candidate ID": "5"},
        {"Candidate ID": "6"},
        {"Candidate ID": "7"},
        {"Candidate ID": "8"},
        {"Candidate ID": "9"},
        {"Candidate ID": "10"},
    ]

    # Path to the Transcripts folder
    transcripts_path = os.path.join('/content/data/transcripts')

    # Evaluate candidates
```

```
evaluation_results = evaluate_candidates(transcripts_path, candidate_summary)

# Display results
for result in evaluation_results:
    print(f"Candidate ID: {result['Candidate ID']}, Decision: {result['Decision']}, "
          f"Average Positive Score: {result['Average Positive Score']:.2f}, "
          f"Average Negative Score: {result['Average Negative Score']:.2f}, "
          f"Average Neutral Score: {result['Average Neutral Score']:.2f}, "
          f"Reasons: {', '.join(result['Reasons'])}")
```

```
Candidate ID: 1, Decision: Recommended, Average Positive Score: 0.18, Average Negative Score: 0.01, Average Neutral Sco
Candidate ID: 2, Decision: Recommended, Average Positive Score: 0.17, Average Negative Score: 0.00, Average Neutral Sco
Candidate ID: 3, Decision: Recommended, Average Positive Score: 0.17, Average Negative Score: 0.02, Average Neutral Sco
Candidate ID: 4, Decision: Not Recommended, Average Positive Score: 0.14, Average Negative Score: 0.01, Average Neutral
Candidate ID: 5, Decision: Not Recommended, Average Positive Score: 0.14, Average Negative Score: 0.00, Average Neutral
Candidate ID: 6, Decision: Not Recommended, Average Positive Score: 0.14, Average Negative Score: 0.00, Average Neutral
Candidate ID: 7, Decision: Not Recommended, Average Positive Score: 0.09, Average Negative Score: 0.01, Average Neutral
Candidate ID: 8, Decision: Recommended, Average Positive Score: 0.16, Average Negative Score: 0.02, Average Neutral Sco
Candidate ID: 9, Decision: Not Recommended, Average Positive Score: 0.14, Average Negative Score: 0.00, Average Neutral
Candidate ID: 10, Decision: Not Recommended, Average Positive Score: 0.13, Average Negative Score: 0.01, Average Neutra
```

```
import pandas as pd
import os
```