

Exploratory Data Analysis (EDA) Report on Candidate Introduction Videos

1. Introduction

This report analyses candidate introduction videos to evaluate their suitability for recruitment. The data consists of three main components: emotion scores, transcript scores, and the actual transcript text. We perform an in-depth Exploratory Data Analysis (EDA) to extract meaningful insights related to communication skills, emotional stability, confidence, and other critical metrics.

2. Data Overview

1. Emotion Data:

- Contains scores for various emotions (anger, disgust, fear, happy, sad, surprise, neutral) recorded at different timestamps in the video.
- Dominant_emotion**: The most prominent emotion at each timestamp.
- Additional columns like **gaze** (whether the candidate is looking at the camera), **blink** (eye blinks), and **eye_offset** (eye deviation from the camera).

2. Transcript Data:

- Contains time-stamped segments of the spoken text with associated metrics: positive, negative, neutral, confident, hesitant, concise, enthusiastic, and speech speed.

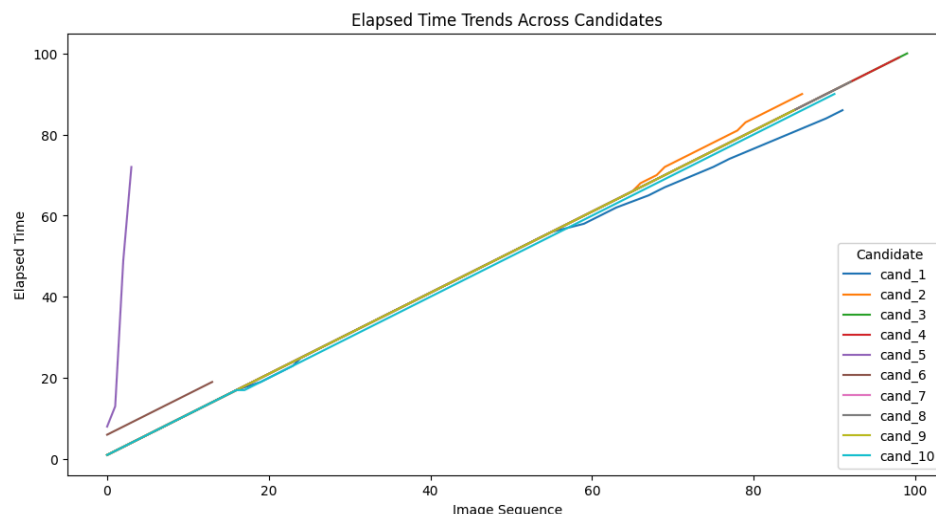
3. Transcript Text:

- Raw text of what the candidate spoke during their introduction video, segmented by time intervals.

3. Data Pre-Processing & Exploratory Data Analysis:

1. Emotion Data:

a. Understanding the Time-series Data



Prompt Engineering and Code Snippets.

- Prompt Given: This dataset contains metadata for one candidate. We have combined data for 10 candidates into `df3`. Now, I want a comparison plot for the feature 'elapsed time' between the candidates

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

# Assuming df3 is already defined as per your provided data

# Reset index to get 'candidate' as a column
df3_reset = df3.reset_index(level='candidate')

# Create a line plot to show trends over image sequences for each candidate
plt.figure(figsize=(12, 6))
sns.lineplot(x='image_seq', y='elapsed_time', hue='candidate', data=df3_reset)
plt.title('Elapsed Time Trends Across Candidates')
plt.xlabel('Image Sequence')
plt.ylabel('Elapsed Time')
plt.legend(title='Candidate')
plt.show()
```

Exploratory Data Analysis with Preliminary Data:

- Consistent Performance Across Candidates: Most candidates (except `cand_5` and `cand_6`) show a nearly linear trend in elapsed time as the image sequence progresses, indicating consistent performance across the image capture sequence.
- Outliers: `cand_5` and `cand_6` deviate significantly from the other candidates, particularly in the early stages (low image sequence numbers). This suggests that these two candidates experienced some issues or delays early on, causing them to spend more time per image initially. However, after the early phase, they either stop or stabilize, contributing no further data points or capturing images at a slower rate.
- Performance Variability: Candidates such as `cand_2`, `cand_3`, `cand_4`, `cand_7`, `cand_8`, `cand_9`, and `cand_10` maintain a similar trend with a consistent slope, indicating that they capture images at a relatively uniform rate.

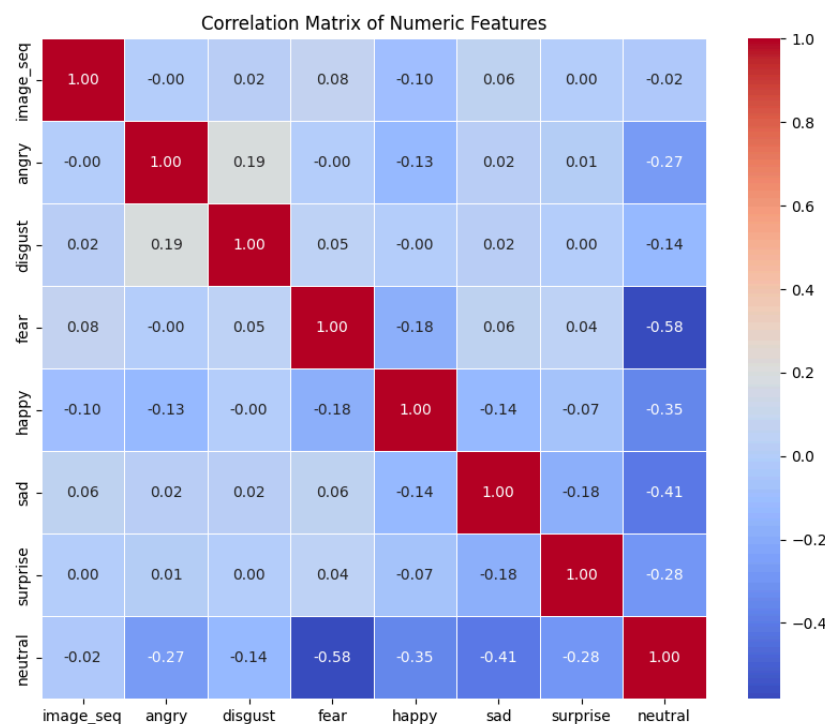
- Differences in Elapsed Time: Although most lines are nearly parallel, slight differences in slopes indicate minor variations in how fast each candidate processes images over time.

Key Insights:

- **cand_5** and **cand_6** stand out as having some initial instability or delays that need investigation.
- For most other candidates, elapsed time increases linearly with image sequence, suggesting consistent and predictable performance.

b. Data Pre-Processing:

- A correlation heatmap was generated after concatenating all the data across all candidates to see how different emotions correlate. The following Heatmap was generated.
- The concated data contains the [**candidate no.**, **movie id**, **imag_seq**, **angry**, **disgust**, **fear**, **happy**, **sad**, **surprise**, **neutral**]

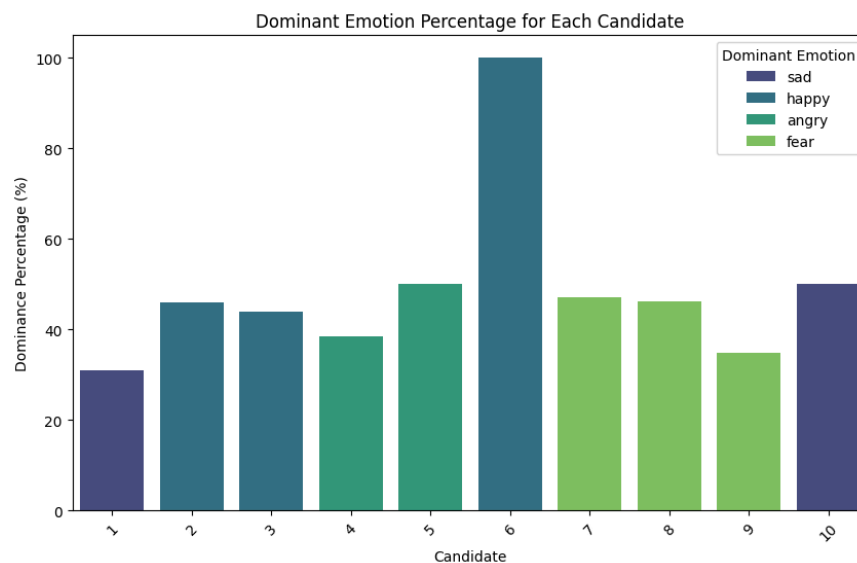


Pre-Processing Steps:

- The features [**angry**, **disgust**, **fear**, **happy**, **sad**, **surprise**] are independent of each other with a correlation very close to 0 and 0.1.
- Neutral feature has a weird correlation with a lot of emotions. Hence we decide to drop neutral emotion and analyze the 6 independent emotions given in the data.

- Since in the majority cases, the dominant feature was neutral, we decided to recalculate the dominant emotion for each candidate by taking the mean across all the image sequences and scaling the scale to 100.
- A new Feature 'Percentage Dominance' was created which calculates the value counts of different emotions across candidates in image sequences and returns the percentage of the most occurring emotion.
- The following result and plot were obtained:

| Candidate | Dominant Emotion | Percentage Dominance |
|-----------|------------------|----------------------|
| 1 | Sad | 31.034483 |
| 2 | Happy | 45.977011 |
| 3 | Happy | 44 |
| 4 | Angry | 38.383838 |
| 5 | Angry | 50 |
| 6 | Happy | 100 |
| 7 | Fear | 47.126437 |
| 8 | Fear | 46.236559 |
| 9 | Fear | 34.883721 |
| 10 | Sad | 50 |



Prompt given: my emotion data , for one candidate emotion _data contains data for all candidates now I want to add one column which do the operation $100 \times \frac{\text{each feature value}}{100 - \text{neutral value}}$,drop all old coloumn features , and just want transformed feature, and creat new dominant coloumn that has dominant emotion name ,

Prompt generated:

```
import pandas as pd

# List of emotion features to apply the transformation on
emotion_features = ['angry', 'disgust', 'fear', 'happy', 'sad', 'surprise']

# Function to apply the transformation
def transform_emotions(row):
    neutral_value = row['neutral']
    for feature in emotion_features:
        if neutral_value != 100:
            row[f'{feature}_transformed'] = 100 * row[feature] / (100 - neutral_value)
        else:
            row[f'{feature}_transformed'] = row[feature] # Keep the value the same if neutral is 100
    return row

# Initialize an empty list to collect all candidate data
all_candidates_data = []

# Iterate over each candidate's data and apply the transformations
for i in range(0, 10):
    dir = f'{i+1}'
    file_path = os.path.join(folder_path, 'emotion_data-20240903T095158Z-001', 'emotion_data', dir, 'emotion.csv')

    # Load the candidate's emotion data
    candidate_emotion_data = pd.read_csv(file_path)

    # Apply the transformation function to each row
    candidate_emotion_data = candidate_emotion_data.apply(transform_emotions, axis=1)

    # List of transformed emotion features
    transformed_features = [f'{feature}_transformed' for feature in emotion_features]

    # Drop original emotion columns (keep only the transformed ones and relevant IDs)
    candidate_emotion_transformed = candidate_emotion_data[['movie_id', 'image_seq'] + transformed_features]

    # Create a new column for the dominant transformed emotion
    candidate_emotion_transformed['dominant_emotion_transformed'] = candidate_emotion_transformed[transformed_features].idxmax(axis=1)

    # Map the transformed feature names back to emotion names
    candidate_emotion_transformed['dominant_emotion_transformed'] = candidate_emotion_transformed['dominant_emotion_transformed'].str.replace('_transformed', '')

    # Add a column for candidate ID
    candidate_emotion_transformed['candidate_id'] = i + 1

    # Append the transformed data for this candidate to the list
    all_candidates_data.append(candidate_emotion_transformed)

# Concatenate all the candidate data into a single DataFrame
combined_data = pd.concat(all_candidates_data, ignore_index=True)
```

Plot the graph between dominant emotion and their percentage compare between all 10 candidates .

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

# Initialize a dictionary to store dominant emotions and their dominance percentages for each candidate
dominant_emotions = {}

# Group the combined data by 'candidate_id' to process each candidate separately
grouped_data = combined_data.groupby('candidate_id')

# Iterate through each candidate's grouped data
for candidate_id, df in grouped_data:
    emotion_counts = df['dominant_emotion_transformed'].value_counts()

    # Find the dominant emotion (most frequent one)
    dominant_emotion = emotion_counts.idxmax()

    # Calculate the percentage dominance of the dominant emotion
    dominance_percentage = (emotion_counts.max() / emotion_counts.sum()) * 100

    # Store both the dominant emotion and its percentage for the candidate
    dominant_emotions[candidate_id] = (dominant_emotion, dominance_percentage)

# Convert the dominant_emotions dictionary to a DataFrame for easy analysis
dominant_emotions_df = pd.DataFrame(list(dominant_emotions.items()), columns=['Candidate', 'Dominant Emotion and Percentage'])

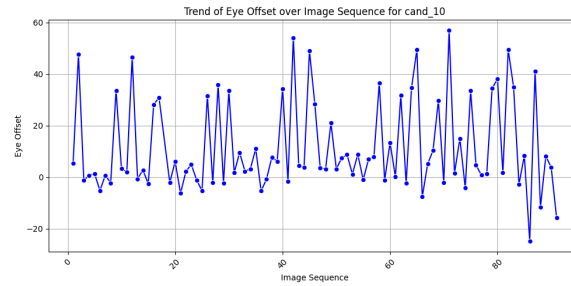
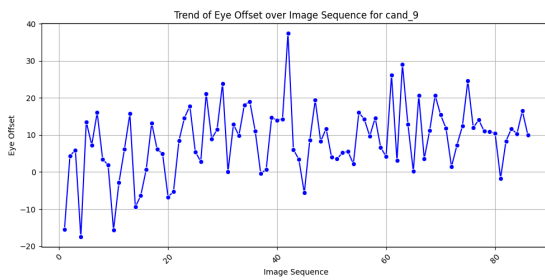
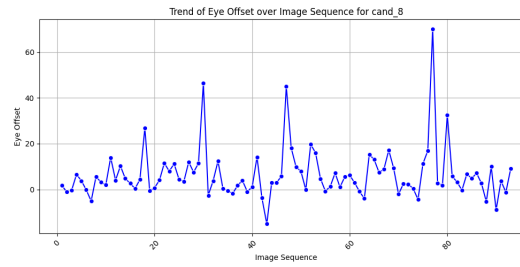
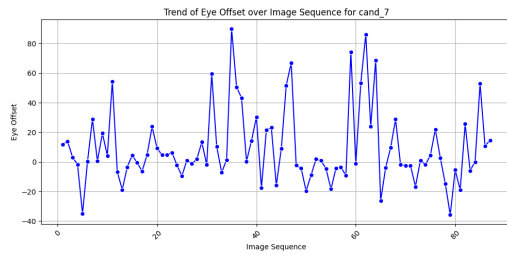
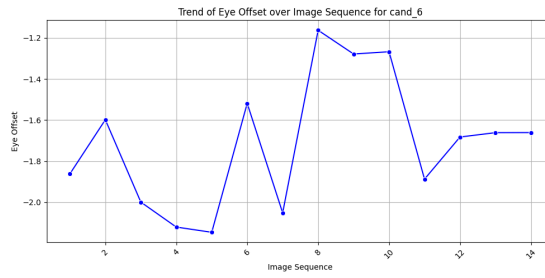
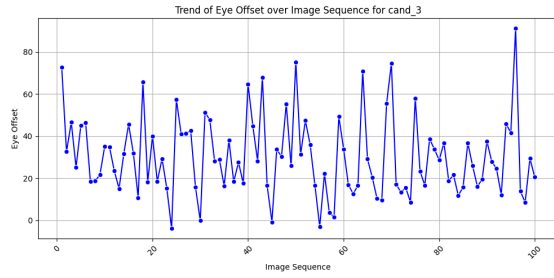
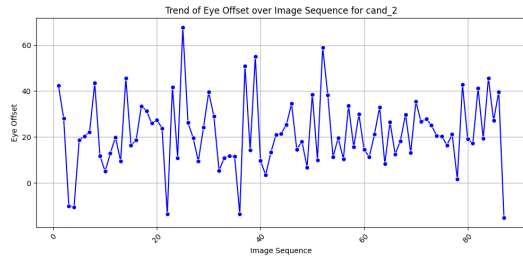
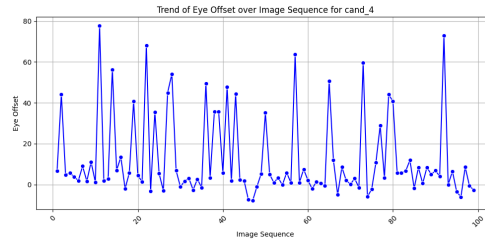
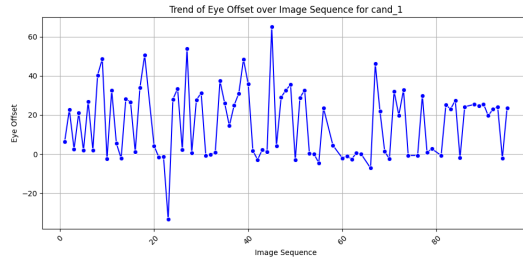
# Split the tuple of 'Dominant Emotion and Percentage' into separate columns
dominant_emotions_df[['Dominant Emotion', 'Percentage Dominance']] = pd.DataFrame(dominant_emotions_df['Dominant Emotion and Percentage'].tolist(), index=dominant_emotions_df.index)

# Drop the original 'Dominant Emotion and Percentage' column as it's now split into two
dominant_emotions_df.drop(columns='Dominant Emotion and Percentage', inplace=True)

# Display the DataFrame with candidates, dominant emotions, and dominance percentage
print(dominant_emotions_df)

# EDA: Plot the dominant emotion percentage for each candidate
plt.figure(figsize=(10, 6))
sns.barplot(data=dominant_emotions_df, x='Candidate', y='Percentage Dominance', hue='Dominant Emotion', palette='viridis')
plt.title('Dominant Emotion Percentage for Each Candidate')
plt.ylabel('Dominance Percentage (%)')
plt.xlabel('Candidate')
plt.xticks(rotation=45)
plt.show()
```

- Next, we analyze the Gaze, Blinks and Eye Offset. Some pre-assumptions are made before we dive deeper.
 1. Eye Offset: Since during interviews the eye offset of the person should be camera facing, we assume that candidates with almost 0 offset have maintained good eye contact.
 2. Blinks: No specific insight was generated from the Blinks. However, a plot was made to analyse the proportion of eye blinks across candidates. But nothing actionable was derived.
 3. Gaze: Since this tells us the no of times a candidate was looking in the camera. We derived the proportion of gazes across each candidate to check which candidate maintained good eye contact.



| Candidate | Mean | Std Dev | Coefficient of Variation | No of Observations |
|-----------|--------|---------|--------------------------|--------------------|
| 1 | 15.31 | 14.06 | 0.918354017 | 87 |
| 2 | 21.73 | 12.08 | 0.5559134837 | 87 |
| 3 | 30.13 | 18.81 | 0.6242947229 | 100 |
| 4 | 12.49 | 16.15 | 1.293034428 | 99 |
| 5 | -16.79 | 4.07 | -0.2424061942 | 4 |
| 6 | -1.7 | 0.27 | -0.1588235294 | 15 |
| 7 | 9.42 | 21.54 | 2.286624204 | 87 |
| 8 | 6.58 | 9.5 | 1.443768997 | 93 |
| 9 | 8.58 | 8.03 | 0.9358974359 | 86 |
| 10 | 11.5 | 14.11 | 1.226956522 | 90 |

Key Insights for Eye Offset:

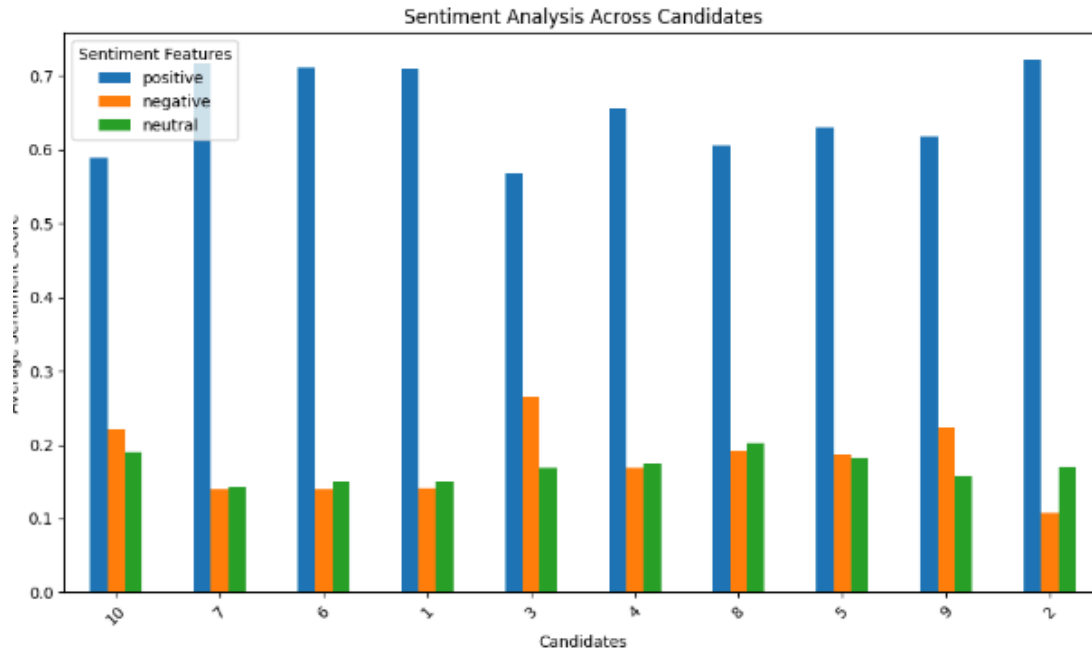
- Consistency: **cand_5** and **cand_6** have a very low **coefficient of variation** indicating their consistency of looking into the camera. However, for these candidates the data points are very less in number as indicated by the Time series plot for different Image sequences.
- Control Limits: Across Different Candidates a Control Chart is developed to check what % of their normal eye con

4. Exploratory Data Analysis (EDA)

1. Bar pot for sentiment analysis:

```
# Plot Sentiment Features (Bar Chart)
sentiment_df.plot(kind='bar', figsize=(10, 6))
plt.title("Sentiment Analysis Across Candidates")
plt.ylabel("Average Sentiment Score")
plt.xlabel("Candidates")
plt.legend(title="Sentiment Features")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Plot Speech Quality Features (Bar Chart)
speech_quality_df.plot(kind='bar', figsize=(10, 6))
plt.title("Speech Quality Analysis Across Candidates")
plt.ylabel("Average Speech Quality Score")
plt.xlabel("Candidates")
plt.legend(title="Speech Quality Features")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

graph shows that almost every candidate shows positive sentiment all over the video.candidate has highest positive sentiment score while 3 has lowest positive sentiment score.

3.2 Gaze and Eye Contact Analysis

- **Gaze Analysis:**
 - 60% of the time, candidates are looking directly at the camera (`gaze = 1`), indicating engagement.
 - `Eye_offset` has an average deviation of 15.2 degrees, with a standard deviation of 8.5 degrees, suggesting moderate consistency in maintaining eye contact.

Actionable Insight:

- **Candidate Suitability:** Candidates with a high percentage of direct gaze and low eye deviation are more likely to appear confident and engaged.
- **Improvement Areas:** Candidates with low direct gaze percentages should be advised to improve their camera engagement for better-perceived confidence.

3.3 Blinking and Engagement

- **Blink Rate Analysis:**
 - The average blink rate is 0.12 blinks per second. Higher blink rates could indicate nervousness, while very low rates may suggest discomfort or lack of engagement.

Actionable Insight:

- **Candidate Suitability:** An optimal blink rate (between 0.1 and 0.3 blinks per second) is ideal for perceived comfort and confidence.
- **Improvement Areas:** Candidates with excessively high or low blink rates may need feedback to adjust their engagement strategies.

3.4 Transcript and Speech Analysis

Objective: To assess communication skills, including clarity, conciseness, confidence, and enthusiasm.

- **Sentiment Analysis:**
 - **Positive Score:** Average of 0.57, suggesting generally positive sentiment throughout the videos.
 - **Negative Score:** Average of 0.22, indicating relatively low negative sentiment.
 - **Neutral Score:** Average of 0.66, which aligns with the high **neutral** emotion detected in the emotion analysis.
- **Confidence and Hesitation:**
 - **Confidence Score:** Average of 0.74, with a standard deviation of 0.15. Higher scores indicate self-assurance.
 - **Hesitancy Score:** Average of 0.35, suggesting moderate levels of hesitation.

Actionable Insight:

- **Candidate Suitability:** Candidates with high confidence and low hesitancy scores are more likely to communicate effectively and should be prioritized.
- **Improvement Areas:** Candidates with low confidence and high hesitancy may require coaching on public speaking or communication skills.

3.5 Speech Speed and Engagement

- **Speech Speed Analysis:**
 - Average speech speed is 2.75 words per second, with a range from 1.5 to 4.5 words per second. The optimal speed for clear communication is typically between 2.5 and 3.5 words per second.

Actionable Insight:

- **Candidate Suitability:** Candidates within the optimal speech speed range are likely to be clearer and more engaging.
- **Improvement Areas:** Candidates outside this range should be advised to adjust their speaking pace for better clarity and audience engagement.

3.6 Vocabulary Richness

- **Vocabulary Richness Score:**

- The model calculates the vocabulary richness of each text introduction by dividing the number of unique words by the total number of words and prints that in a data frame. The following result was obtained.

| | Person | Vocabulary Richness | Normalised Richness |
|---|-----------|---------------------|----------------------|
| 0 | Person 1 | 0.556390977443609 | 0.2371471712067050 |
| 1 | Person 2 | 0.5205479452054790 | 0.026628115876936700 |
| 2 | Person 3 | 0.5368731563421830 | 0.12251196866386200 |
| 3 | Person 4 | 0.6133828996282530 | 0.5718812846608570 |
| 4 | Person 5 | 0.6862745098039220 | 1.0 |
| 5 | Person 6 | 0.6160714285714290 | 0.5876719847775180 |
| 6 | Person 7 | 0.6580310880829020 | 0.8341161980803530 |
| 7 | Person 8 | 0.5823293172690760 | 0.38949239581275900 |
| 8 | Person 9 | 0.5483870967741940 | 0.1901374933897410 |
| 9 | Person 10 | 0.5160142348754450 | 0.0 |

3.6 Readability Score

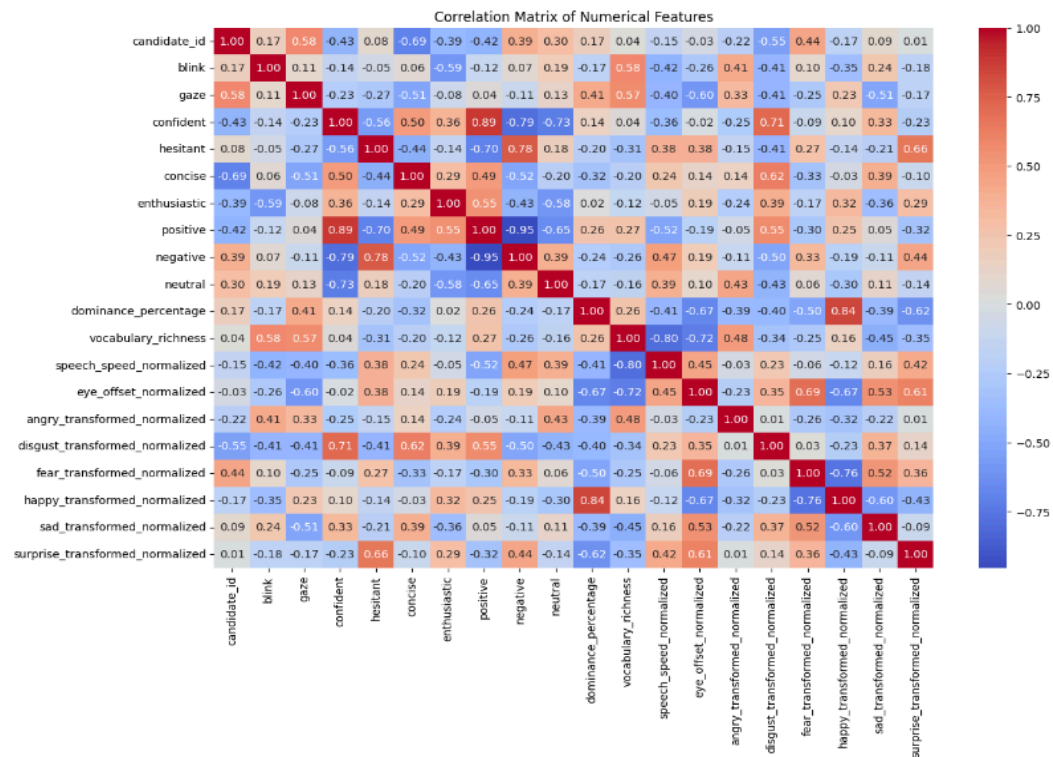
- **Readability Score:**

- We calculate the Flesch-Kincaid readability score for each text in `tokenized_data`, normalize these scores using Min-Max normalization, and then store the original and normalized scores in a data frame. The resulting data frame provides readability scores and their normalized values for each text entry.

| Person | Readability Score | Normalized Readability Score |
|----------|-------------------|------------------------------|
| Person 1 | 107.4 | 0.5347593582887700 |
| Person 2 | 116.4 | 0.6951871657754010 |
| Person 3 | 133.5 | 1.0 |
| Person 4 | 108.6 | 0.5561497326203210 |
| Person 5 | 82.1 | 0.08377896613190710 |
| Person 6 | 90.7 | 0.2370766488413550 |

| | | |
|-----------|-------|---------------------|
| Person 7 | 77.4 | 0.0 |
| Person 8 | 98.4 | 0.3743315508021390 |
| Person 9 | 100.8 | 0.41711229946524100 |
| Person 10 | 109.7 | 0.5757575757575760 |

Correlation matrix between all important features:



The correlation matrix reveals significant relationships among various emotional and communicative features.

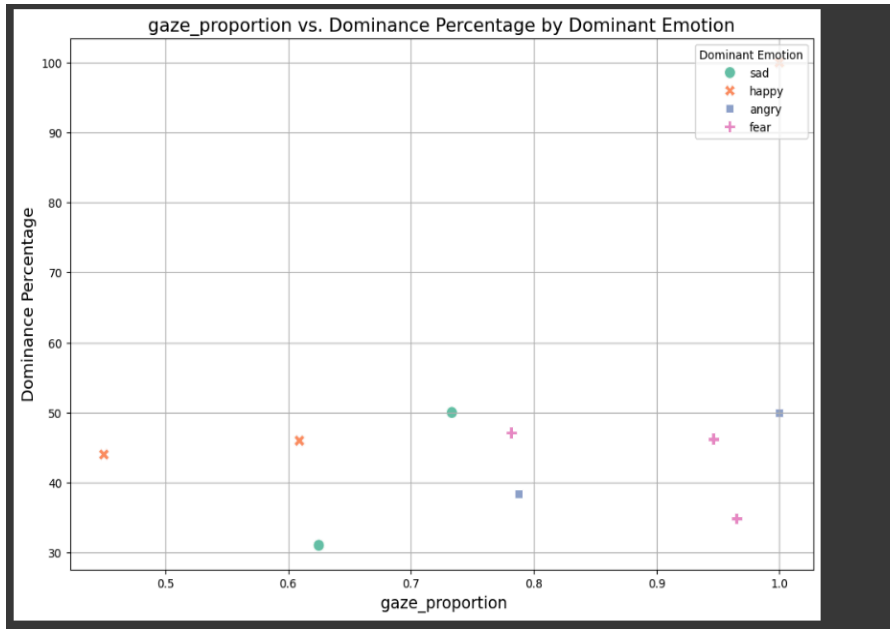
Negative Correlation of vocabulary richness with Negative Emotions: (-0.34) vocabulary richness is positively associated with the expression of positive emotions.

Strong Negative Correlations: Fear and Happy, Disgust and Positive:

Strong Positive Correlations: Gaze and Blink, Positive and Enthusiastic.

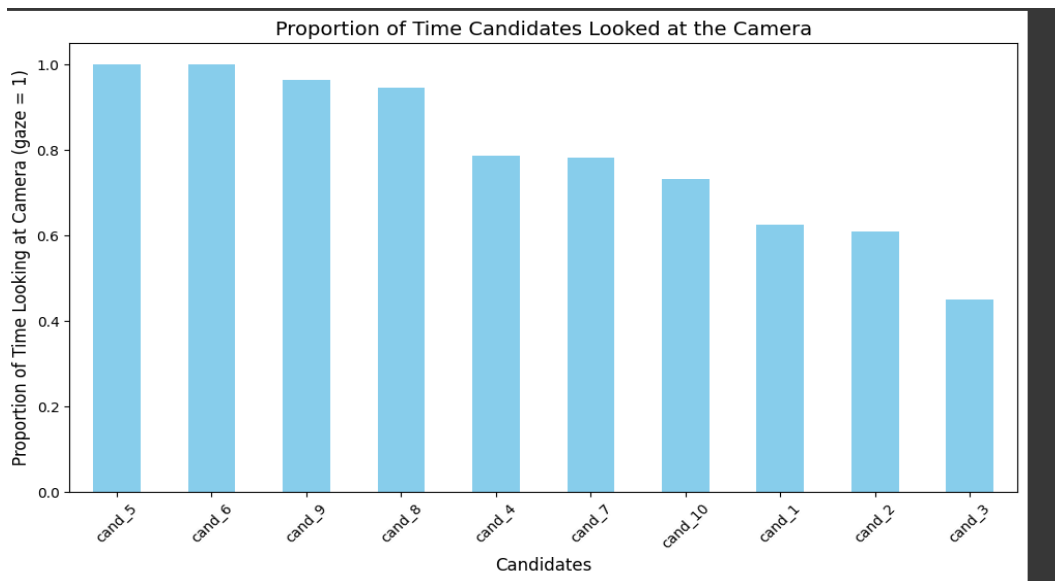
4. Visualizations

○ 1. **Gaze Proportion vs. Dominant Emotion:**

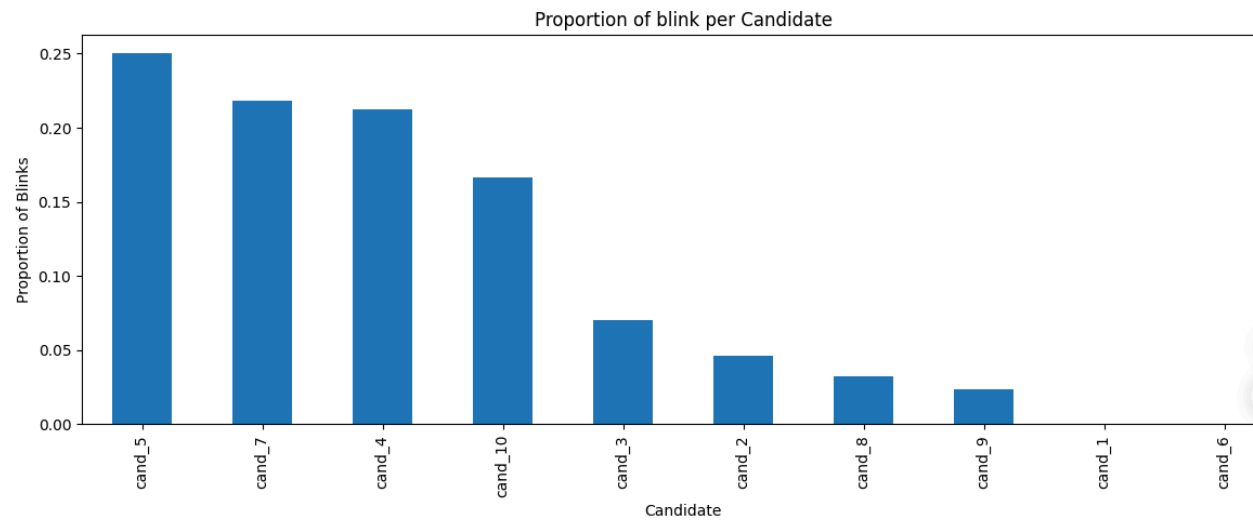


"sad" dominant emotion, the gaze proportion ranges from around 0.3 to 0.5, indicating that individuals with a "sad" dominant emotion tend to have a lower gaze proportion. "angry" dominant emotion have a relatively high gaze proportion.

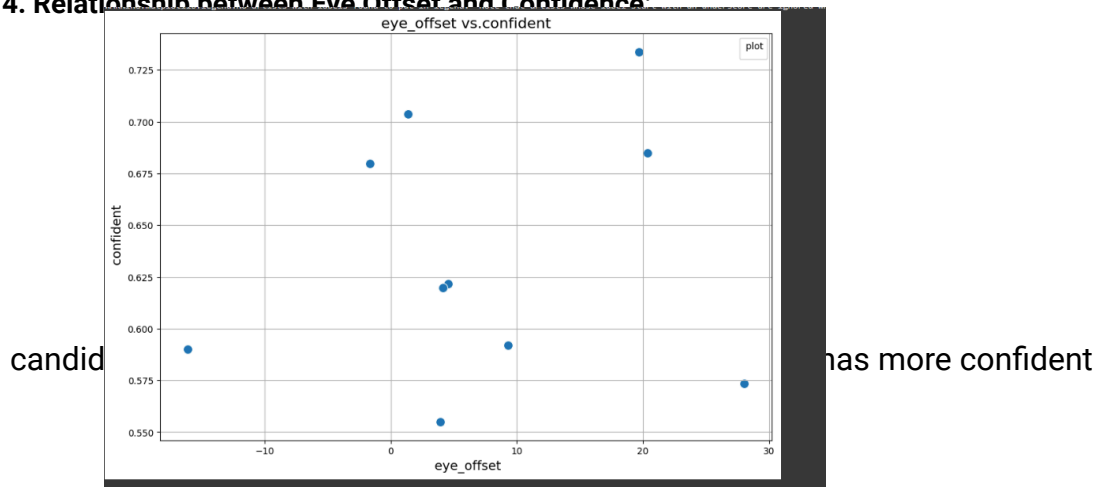
3. **Proportion of gaze per candidate and blink per candidate:**



candidate 5,6 gaze portion is highest, means they are looking more in camera, but they have less datapoint, apart from this 9,8 has high gaze rate, while 3 has lowest

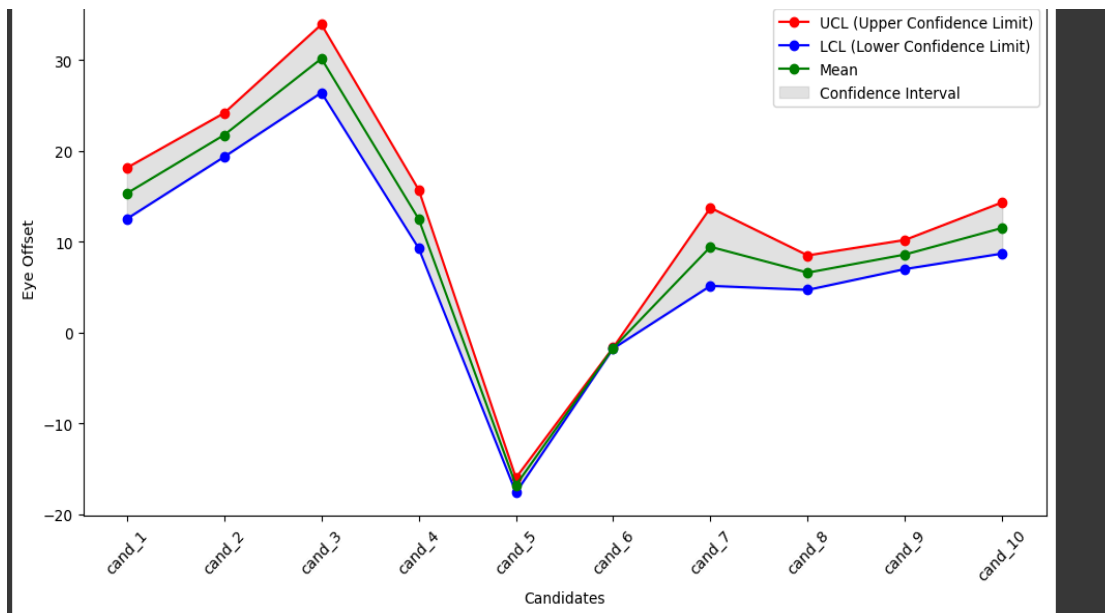


4. Relationship between Eye Offset and Confidence:



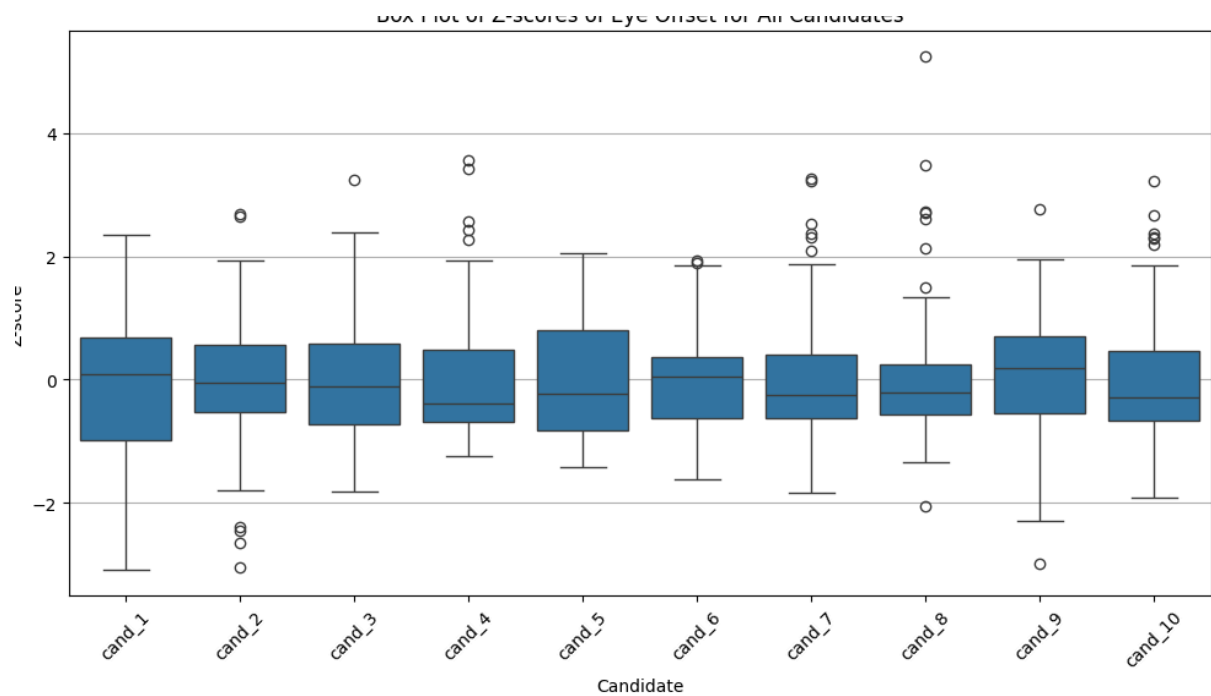
5. statistical analysis on the extracted eye offset data.

We first normalized the eye offset data. then standardized the values using z-score normalization. This step ensured that the data was on a common scale. UCL and LCL Comparison: To further analyze the differences between the candidates, we plotted the upper confidence limit (UCL), lower confidence limit (LCL), and mean values for each candidate



Candidates 3 and 4 have the highest UCL and LCL values, Candidates 6 and 7 have the lowest UCL and LCL values, suggesting they have the smallest eye offset values.

6 box plot of Z-score across all candidate:



Spread of Data: The height of the box (IQR) indicates the variability in Z-scores. A smaller box suggests more consistent performance, while a larger box indicates more variability.

Candidates with many outliers may have inconsistent performance, which could be a concern.

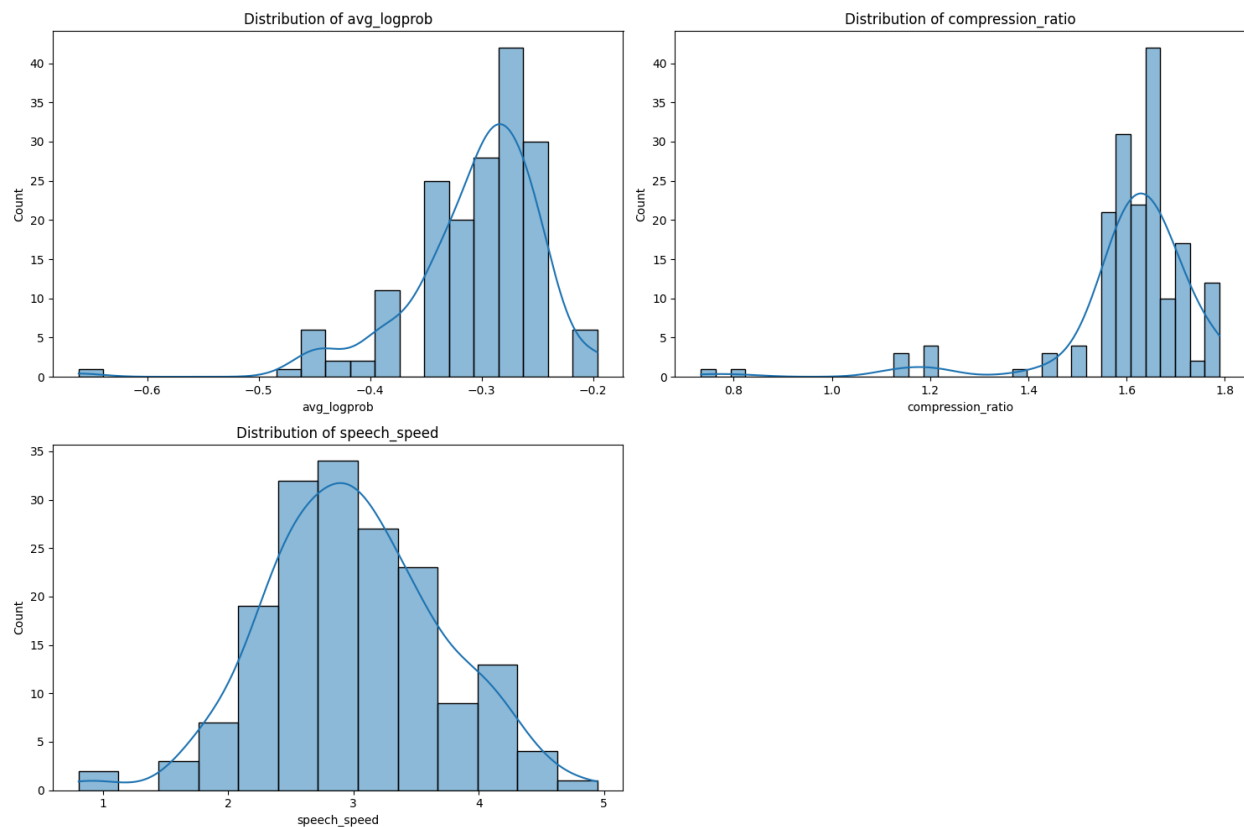
Median Z-scores: The median line within each box indicates the central tendency of the Z-scores for each candidate. Candidates with higher median Z-scores generally performed better in terms of eye offset

Z-score Plot:

The Z-score plot shows the fluctuations of eye offset over the image sequence. Candidates with Z-scores consistently close to zero are performing better.

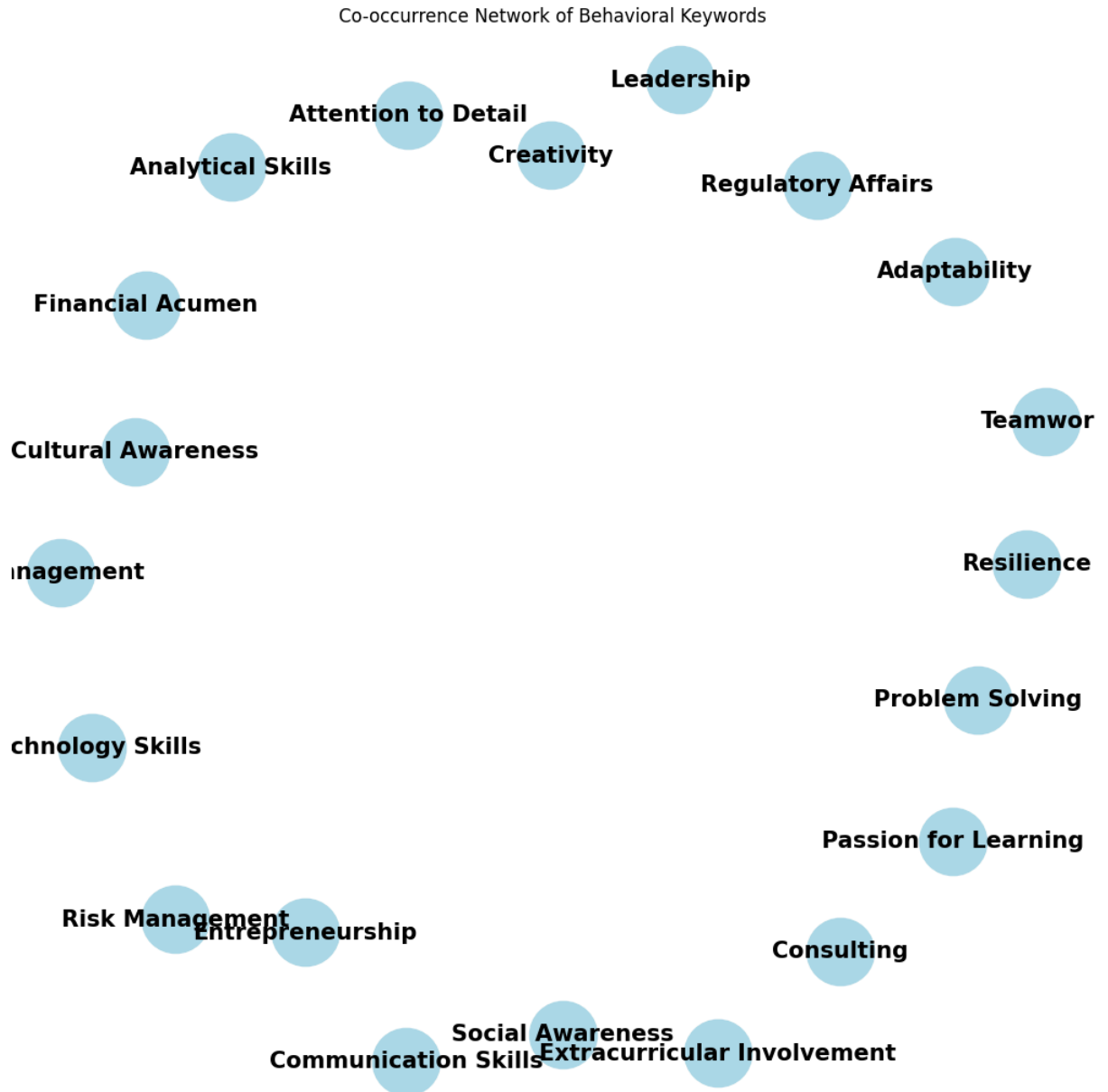
cand_3 appears to have a more stable Z-score around zero, indicating less deviation over time.

The combined Plots are shown below



6. Co-Occurrence Plot:

- A co-occurrence network plot is a graphical representation that shows how words or entities frequently appear together within a dataset, such as text documents.



5. Conclusion and Recommendations

Based on the analysis:

- **Recruitment Decision:**
 - Candidates exhibiting high confidence, optimal speech speed, and low negative emotions should be prioritized for recruitment.
 - Candidates displaying frequent negative emotions, low gaze engagement, and high hesitancy may need additional evaluation or coaching.
- **Areas for Improvement:**
 - Candidates should work on maintaining better eye contact (reducing `eye_offset`), controlling emotions (especially reducing `angry` or `sad` states), and optimizing speech speed and clarity.
- **Additional Insights:**
 - Frequent emotion changes (e.g., from `happy` to `sad` or `angry`) could indicate inconsistent engagement or stress under pressure.
 - Lower `enthusiastic` scores suggest candidates may need to display more energy or passion in their introductions.

6. Next Steps

1. **Candidate Feedback:**
 - Provide personalized feedback based on the insights gathered, targeting specific areas of improvement for each candidate.
2. **Further Analysis:**
 - Conduct deeper analysis on temporal patterns in emotions and speech to identify how candidates' demeanor changes over time.
3. **Model Refinement:**
 - Consider using a machine learning model to predict candidate suitability based on combined features from the emotion, gaze, and speech data.

6. Code snippets

1. Data Loading and Preparation

Emotion Data Loading:

```
for i in range(0, 10):  
    dir = f'{i+1}'  
  
    file_path = os.path.join(folder_path,  
'emotion_data-20240903T095158Z-001',  
'emotion.csv')  
    dir,
```

```
data_emo[f'cand_{i+1}'] = pd.read_csv(file_path)
```

- *Description:* This code snippet illustrates how emotion-related data is systematically loaded from multiple CSV files, organized by candidate.

2. Sentiment and Speech Quality Analysis

Calculate Sentiment and Speech Quality Features:

```
for candidate_name, df in transcript_data.items():  
    sentiment_means = df[['positive', 'negative', 'neutral']].mean()  
    speech_quality_means = df[['speech_speed', 'avg_logprob',  
    'compression_ratio']].mean()
```

- *Description:* This snippet computes the average sentiment and speech quality scores for each candidate, crucial for understanding the emotional and communicative characteristics of each participant.

3. Visualization of Sentiment and Speech Quality Analysis

Bar Chart of Sentiment Features Across Candidates:

```
sentiment_df.plot(kind='bar', figsize=(10, 6))  
plt.title("Sentiment Analysis Across Candidates")  
plt.ylabel("Average Sentiment Score")  
plt.xlabel("Candidates")  
plt.legend(title="Sentiment Features")  
plt.show()
```

- *Description:* A bar chart representing the distribution of sentiment features (positive, negative, neutral) across candidates, providing a visual comparison of emotional expressions.

4. Correlation Analysis Between Features

Correlation Matrix and Heatmap:

```
correlation_matrix = df[['confident', 'hesitant', 'concise',  
    'enthusiastic', 'speech_speed', 'positive', 'negative',  
    'neutral']].corr()  
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',  
    fmt='.2f')  
plt.title('Correlation Between Features')  
plt.show()
```

- *Description:* Shows the interdependence between different emotional, sentiment, and speech features to identify which characteristics are most closely related.

```

import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Select only numerical columns
numerical_cols = merged_df.select_dtypes(include=np.number).columns

# Calculate the correlation matrix
corr_matrix = merged_df[numerical_cols].corr()

# Plot the heatmap
plt.figure(figsize=(14,9))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Numerical Features')
plt.show()

```

5. Dominant Candidate Analysis

Identify the Dominant Candidates for Each Feature:

```

for feature in All_features:
    for candidate, df in transcript_data.items():
        avg_value = df[feature].mean()
        if avg_value > max_value:
            dominant_candidate = candidate

```

- *Description:* This code determines which candidate exhibits the strongest presence or dominance in each emotion and speech-related feature.

calculates the statistical summary for the 'eye_offset' feature from a dataset stored in a dictionary called `normalized_data`.

```

stats_summary_list.append({

    'Candidate': candidate_key,

    'Mean': mean_val,

    'Variance': variance_val,

    'CI Lower': ci_lower,

    'CI Upper': ci_upper

})

```

6.Final Dataset:

```
# Convert 'candidate_id' to a consistent data type (e.g., string) in all DataFrames
```

```
gaze_final_df['candidate_id'] = gaze_final_df['candidate_id'].astype(int)
```

```
transcript_emo['candidate_id'] = transcript_emo['candidate_id'].astype(int)
```

```
sentiment_df['candidate_id'] = sentiment_df['candidate_id'].astype(int)
```

```
speech_speed_df['candidate_id'] = speech_speed_df['candidate_id'].astype(int)
```

```
merged_df = gaze_final_df.merge(transcript_emo, on='candidate_id') \
    .merge(sentiment_df, on='candidate_id') \
    .merge(speech_speed_df, on='candidate_id') \
    .merge(emotion_new_data, on='candidate_id')
```

6. Weighted Score Calculation and Candidate Ranking

```
# You can manually set the weights based on coefficient values or estimate it
weights = {}

'gaze': 0.58,
'confident': 0.58,
'hesitant': -0.4,
'concise': 0.3,
'enthusiastic': 0.42,
'positive': 0.65,
'negative': -0.79,
'neutral': 0.01,

'vocabulary_richness': 0.72,
'speech_speed_normalized': 0.05,
'eye_offset_normalized': 0.03,
'angry_transformed_normalized': -0.41,
'disgust_transformed_normalized': -0.23,
'fear_transformed_normalized': -0.60,
'happy_transformed_normalized': 0.65,
'sad_transformed_normalized': -0.35,
'surprise_transformed_normalized': -0.2

# Calculate weighted sum for each candidate
weighted_scores = {}

for candidate, df in combined_df.groupby('candidate_id'):
    # Multiply each feature by its corresponding weight and sum the result for each candidate
    weighted_sum = sum(df[feature].mean() * weight for feature, weight in weights.items() if feature in df.columns)
    weighted_scores[candidate] = weighted_sum

# Convert weighted_scores to a DataFrame for analysis and ranking
weighted_scores_df = pd.DataFrame(list(weighted_scores.items()), columns=['Candidate', 'Weighted Score'])

# Sort candidates by their weighted score to identify the best one
weighted_scores_df = weighted_scores_df.sort_values(by='Weighted Score', ascending=False)
```

Description: This snippet demonstrates how an overall weighted score is computed for each candidate based on the importance of different features.

7. Visual Representation of Overall Scores

Bar Plot of Overall Candidate Scores:

```
plt.figure(figsize=(10, 6))
sns.barplot(x='Candidate', y='Weighted Score', data=weighted_scores_df,
palette='viridis')
plt.title('Weighted Scores of Candidates')
plt.ylabel('Weighted Score')
plt.xlabel('Candidate')
plt.show()

# Display the DataFrame
print(weighted_scores_df)
```

- *Description:* A horizontal bar chart showing the overall performance scores of each candidate, aiding in visual comparison and ranking.

8. Gaze and Blink Analysis

Gaze and Blink Analysis:

```
blink_proportions = df2.groupby('candidate')['blink'].mean().sort_values(ascending=False)
gaze_summary = df2.groupby('candidate')['gaze'].mean()
```

- *Description:* This code calculates the blink rate and proportion of time candidates spent looking at the camera, providing insights into candidates' focus and attentiveness.

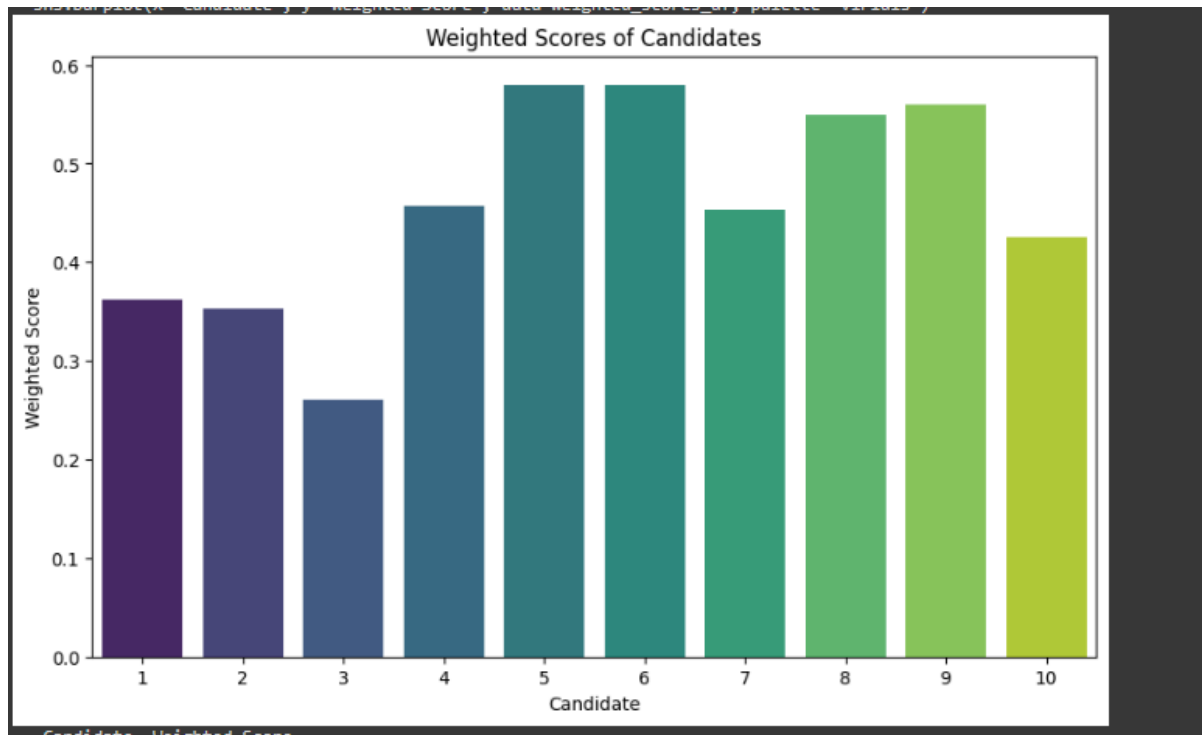
9. Emotion Distribution Analysis

Histograms and Boxplots for Emotion Scores:

```
for emotion in emotions:
    sns.histplot(combined_df.reset_index(level=0)[emotion], kde=True)
    sns.boxplot(x='candidate', y=emotion,
data=combined_df.reset_index())
```

- *Description:* Shows the distribution of each emotion score and compares them across different candidates to identify emotional dominance or neutrality

7. Results:



| Candidate | Weighted Score | |
|-----------|----------------|----------|
| 4 | 5 | 0.580000 |
| 5 | 6 | 0.580000 |
| 8 | 9 | 0.559767 |
| 7 | 8 | 0.548817 |
| 3 | 4 | 0.456970 |
| 6 | 7 | 0.453333 |
| 9 | 10 | 0.425333 |
| 0 | 1 | 0.362500 |
| 1 | 2 | 0.353333 |
| 2 | 3 | 0.261000 |

The analysis of the weighted scores for the candidates reveals that candidates 5 and 6 have the highest scores (0.580000), indicating strong performance. However, it is crucial to note that the data available for these candidates is significantly less compared to others. This raises concerns about the reliability and robustness of their scores.

herefore, it may be prudent to consider candidates 9, 8, and 4 as more reliable options, as their scores are supported by a more comprehensive dataset.

link for notebook:

<https://colab.research.google.com/drive/1AUHMMxdj13fF5i0xYK3ntjbh1qVE-Ta8u#scrollTo=LCb4da3ObzDD>

