Prompt Engineering Documentation

Spriha Ghosh

28.09.2023

Data Preprocessing

Some lengthy codes were generated using ChatGPT

Prompt 1

You:

I want you to act as a data scientist. You are given a dataset of 10 candidates who have submitted introductory videos to a tech startup company specializing in the use of AI technology for creating mental health solutions. The dataset of each candidate consists of the following: In "emotion_data" folder > "1" folder (This dataset contains emotions of the candidate throughout the video): emotion.csv - movie_id image_seq angry disgust fear happy sad surprise neutral dominant_emotion gaze.csv - movie_id image_seq gaze blink eye_offset metadata.csv – movie_id image_seq participant_id elapsed_time upload_time distance In "transcript_data" folder (This dataset contains scores extracted from the transcripts throughout the video): 1.csv – id seek start end text tokens temperature avg_logprob compression_ratio no_speech_prob positive negative neutral confident hesitant concise enthusiastic speech_speed The actual text of the transcript of the video is provided in "transcripts" folder > 1.txt

---

ChatGPT:

Great, it looks like you have provided a detailed dataset for candidates who submitted introductory videos to a tech startup specializing in AI-based mental health solutions. As a data scientist, I can help you with various tasks related to this dataset, such as data exploration, analysis, and modeling. Please let me know what specific tasks or analyses you'd like to perform on this dataset, or if you have any questions or objectives in mind for this project.

Prompt 2 – to perform general data cleaning on provided tables

You:

Write a Python code in a Jupyter Notebook to perform several data preprocessing tasks on 10 datasets related to candidates. 1) Run a for loop to read all 10 datasets of the candidates. Remove participant_id, upload_time and distance from the metadata.csv for all the 10 candidates. Remove seek, tokens, temperature, avg_logprob, compression_ratio and no_speech_prob from transcript_data destination folder of all 10 candidates. 2) Display the first two rows of metadata.csv and the tables in the transcript_data folder, of all the 10 candidates after the removal of these columns. 3) Additionally, find and handle any missing values in emotion.csv, gaze.csv, metadata.csv, in all 10 candidates' datasets, as well as transcript_data tables of all 10 candidates. 4) Find and remove duplicate rows in emotion.csv, gaze.csv, metadata.csv, for all 10 candidates and transcript_data tables for all 10 candidates. 5) Apply <table>.describe(), <table>.info() for emotion.csv, gaze.csv, metadata.csv, for all 10 candidates and transcript_data tables for all 10 candidates. 6) Replace all the old tables with modified tables (emotion.csv, gaze.csv, metadata.csv, transcript_data tables) Write the code in 6 code cells.

ChatGPT:

To perform the specified data preprocessing tasks, we'll follow these steps:

Code Cell 1: Importing Libraries and Setting Up File Paths

We'll start by importing necessary libraries and defining file paths for the datasets.

```
python
import os
```

```
# Define the base directory path
base_dir = '/path/to/your/dataset/'

# List of candidate folders (assuming they are named '1', '2', ..., '10')
candidate_folders = [str(i) for i in range(1, 11)]

# Define file paths for each candidate's datasets
emotion_paths = [os.path.join(base_dir, folder, 'emotion_data', '1', 'emotion.csv') for folder in candidate_folders]
gaze_paths = [os.path.join(base_dir, folder, 'emotion_data', '1', 'gaze.csv') for folder in candidate_folders]
metadata_paths = [os.path.join(base_dir, folder, 'emotion_data', '1', 'metadata.csv') for folder in candidate_folders]
transcript_paths = [os.path.join(base_dir, folder, 'transcript_data', '1.csv') for folder in candidate_folders]
```

Code Cell 2: Data Preprocessing - Remove Columns and Display First Rows

We'll remove specified columns from metadata.csv and display the first two rows of each dataset.

```
python
# Define columns to remove
metadata_columns_to_remove = ['participant_id', 'upload_time', 'distance']
transcript_columns_to_remove = ['seek', 'tokens', 'temperature', 'avg_logprob', 'compression_ratio', 'no_speech_prob']

# Create empty lists to store modified DataFrames
metadata_dfs = []
transcript_dfs = []
```

```
# Loop through candidate folders
for folder in candidate_folders:
    metadata_path = os.path.join(base_dir, folder, 'emotion_data', '1', 'metadata.csv')
    transcript_path = os.path.join(base_dir, folder, 'transcript_data', '1.csv')

    # Read metadata.csv and remove specified columns
    metadata_df = pd.read_csv(metadata_path)
    metadata_df = metadata_df.drop(columns=metadata_columns_to_remove)

    # Read transcript_data and remove specified columns
    transcript_df = pd.read_csv(transcript_path)
    transcript_df = transcript_df.drop(columns=transcript_columns_to_remove)

    # Append modified DataFrames to the lists
    metadata_dfs.append(metadata_df)
    transcript_dfs.append(transcript_df)

    # Display the first two rows
    print(f"Metadata.csv - Candidate {folder}:\n", metadata_df.head(2))
    print(f"Transcript Data - Candidate {folder}:\n", transcript_df.head(2))
```

Code Cell 3: Handling Missing Values

We'll handle missing values in emotion.csv, gaze.csv, metadata.csv, and transcript_data tables for all candidates.

```
python
# Create empty lists to store modified DataFrames
emotion_dfs = []
gaze_dfs = []

# Loop through candidate folders
for folder in candidate_folders:
    emotion_path = os.path.join(base_dir, folder, 'emotion_data', '1', 'emotion.csv')
    gaze_path = os.path.join(base_dir, folder, 'emotion_data', '1', 'gaze.csv')

    # Read emotion.csv and handle missing values
    emotion_df = pd.read_csv(emotion_path)
    emotion_df = emotion_df.fillna(0)  # Replace missing values with 0

    # Read gaze.csv and handle missing values
    gaze_df = pd.read_csv(gaze_path)
    gaze_df = gaze_df.fillna(0)  # Replace missing values with 0

    # Append modified DataFrames to the lists
    emotion_dfs.append(emotion_df)
    gaze_dfs.append(gaze_df)

# Display a message confirming missing value handling
print("Missing values handled in emotion.csv and gaze.csv.")
```

Code Cell 4: Removing Duplicate Rows

We'll remove duplicate rows in emotion.csv, gaze.csv, metadata.csv, and transcript_data tables for all candidates.

```
python
# Create empty lists to store modified DataFrames
emotion_dfs_no_duplicates = []
gaze_dfs_no_duplicates = []

# Loop through candidate folders
for i, folder in enumerate(candidate_folders):
    # Remove duplicates in emotion.csv
    emotion_df_no_duplicates = emotion_dfs[i].drop_duplicates()

    # Remove duplicates in gaze.csv
    gaze_df_no_duplicates = gaze_dfs[i].drop_duplicates()

    # Append modified DataFrames to the lists
    emotion_dfs_no_duplicates.append(emotion_df_no_duplicates)
    gaze_dfs_no_duplicates.append(gaze_df_no_duplicates)

# Display a message confirming duplicate removal
print("Duplicate rows removed from emotion.csv and gaze.csv.")
```

Code Cell 5: Descriptive Statistics and Info

We'll apply `describe()` and `info()` to the datasets.

```
python
# Function to display describe() and info() for a DataFrame
def describe_info(dataframe, name):
    print(f"Summary Statistics for {name}:\n", dataframe.describe())
    print(f"Info for {name}:\n", dataframe.info())

# Apply describe() and info() for emotion.csv, gaze.csv, metadata.csv, and transcript_data tables
for i, folder in enumerate(candidate_folders):
    print(f"\nCandidate {folder} Data:")
    describe_info(emotion_dfs_no_duplicates[i], f"Emotion.csv (Candidate {folder})")
    describe_info(gaze_dfs_no_duplicates[i], f"Gaze.csv (Candidate {folder})")
    describe_info(metadata_dfs[i], f"Metadata.csv (Candidate {folder})")
    describe_info(transcript_dfs[i], f"Transcript Data (Candidate {folder})")
```

Code Cell 6: Replace Original Tables with Modified Tables

Now, we'll replace the original tables (emotion.csv, gaze.csv, metadata.csv, and transcript_data tables) with the modified tables (without duplicates and missing values).

```
python
# Replace original files with modified DataFrames
for i, folder in enumerate(candidate_folders):
    emotion_dfs_no_duplicates[i].to_csv(os.path.join(base_dir, folder, 'emotion_data', '1', 'emotion.csv'), index=False)
    gaze_dfs_no_duplicates[i].to_csv(os.path.join(base_dir, folder, 'emotion_data', '1', 'gaze.csv'), index=False)
    metadata_dfs[i].to_csv(os.path.join(base_dir, folder, 'emotion_data', '1', 'metadata.csv'), index=False)
    transcript_dfs[i].to_csv(os.path.join(base_dir, folder, 'transcript_data', '1.csv'), index=False)

print("Modified tables replaced the original tables.")
```

Make sure to replace `/path/to/your/dataset/` with the actual path to your dataset directory. This code performs the specified data preprocessing tasks on all 10 candidates' datasets and displays the requested information.

Modifications: The file paths are changed.

Prompt 3 – To create overall_data tables by merging merged_emotion and transcript data tables

You:

In continuation with the above codes, merge merged_emotion_i.csv file in merged_emotion folder with i_updated.csv file in transcript_data folder for all the 10 candidates by checking "elapsed_time" column in merged_emotion_i.csv is within the range of values of "start" and "end" columns (both inclusive) in i_updated.csv. If "n" number of rows of merged_emotion_i.csv is connected to one row of i_updated.csv, then the values of merged_emotion_i columns (angry, disgust, fear, happy, sad, surprise, neutral, gaze, blink, eye_offset, image_seq, elapsed_time) for one row is taken as the average of the values of the respective columns in "n" number of rows. Value of "movie_id" is the same as the value of first row of merged_emotion_i.csv. Remove "image_seq" and "elapsed_time" columns. Add the respective "start" and "end" values in columns. Create a new folder called "overall_data". For each candidate, save the merged table as overall_data_i.csv file (for I in range(1,11)) in the overall_data folder. Column descriptions: elapsed_time - Timestamp in seconds (9 means at 9th sec), start - start time of the text, end - end time of the text.

ChatGPT:

Certainly! Here's the code to merge "merged_emotion_i.csv" files from the "merged_emotion" folder with "i_updated.csv" files from the "transcript_data" folder for all 10 candidates. The merging is done based on the "elapsed_time" column in "merged_emotion_i.csv" within the range of values of the "start" and "end" columns (both inclusive) in "i_updated.csv". If "n" number of rows in "merged_emotion_i.csv" are connected to one row in "i_updated.csv", the values in columns (angry, disgust, fear, happy, sad, surprise, neutral, gaze, blink, eye_offset) for one row are taken as the average of the values of the respective columns in "n" rows. The "movie_id" value is set to be the same as the value in the first row of "merged_emotion_i.csv". Additionally, we remove the "image_seq" and "elapsed_time" columns and add the respective "start" and "end" values in new columns. Finally, we save the merged tables as "overall_data_i.csv" files in the "overall_data" folder.

Code Cell 11: Merging Data and Saving in "overall_data" Folder

```
python
# Create a new folder for overall data if it doesn't exist
overall_data_dir = os.path.join(base_dir, 'overall_data')
os.makedirs(overall_data_dir, exist_ok=True)

# Loop through candidate folders
for i, folder in enumerate(candidate_folders):
    # Read merged_emotion_i.csv and i_updated.csv
    merged_emotion_path = os.path.join(base_dir, 'merged_emotion', f'merged_emotion_{folder}.csv')
    updated_transcript_path = os.path.join(base_dir, 'transcript_data', f'{folder}_updated.csv')

    if os.path.exists(merged_emotion_path) and os.path.exists(updated_transcript_path):
        # Read merged_emotion_i.csv and i_updated.csv
        merged_emotion_df = pd.read_csv(merged_emotion_path)
        updated_transcript_df = pd.read_csv(updated_transcript_path)

        # Initialize lists to store rows for the merged table
        merged_rows = []

        # Loop through rows in updated_transcript_df
        for index, row in updated_transcript_df.iterrows():
            # Extract the start and end times from the current row
            start_time = row['start']
            end_time = row['end']
```

```
            # Filter rows in merged_emotion_df based on elapsed_time within the range
            filtered_rows = merged_emotion_df[
                (merged_emotion_df['elapsed_time'] >= start_time) &
                (merged_emotion_df['elapsed_time'] <= end_time)
            ]

            # If there are matching rows in merged_emotion_df, calculate averages
            if not filtered_rows.empty:
                avg_row = filtered_rows.mean(numeric_only=True)
                avg_row['start'] = start_time
                avg_row['end'] = end_time
                merged_rows.append(avg_row)

        # Create the merged DataFrame for the candidate
        candidate_merged_df = pd.DataFrame(merged_rows)

        # Set 'movie_id' to the value from the first row of merged_emotion_i.csv
        candidate_merged_df['movie_id'] = merged_emotion_df['movie_id'].iloc[0]

        # Remove 'image_seq' and 'elapsed_time' columns
        candidate_merged_df = candidate_merged_df.drop(columns=['image_seq', 'elapsed_time'])
```

```
        # Reorder columns to match the desired order
        column_order = ['movie_id', 'start', 'end', 'angry', 'disgust', 'fear', 'happy', 'sad',
                        'surprise', 'neutral', 'gaze', 'blink', 'eye_offset']
        candidate_merged_df = candidate_merged_df[column_order]

        # Save the merged DataFrame as overall_data_i.csv
        overall_data_path = os.path.join(overall_data_dir, f'overall_data_{folder}.csv')
        candidate_merged_df.to_csv(overall_data_path, index=False)

        print(f"Merged data saved for Candidate {folder} in overall_data_{folder}.csv.")
    else:
        print(f"Files not found for Candidate {folder}. Skipping.")

print("Merging and saving completed.")
```

This code will create a new folder called "overall_data" if it doesn't exist, then perform the merging, column reordering, and saving for each candidate's data as described. Make sure to run this code after running the previous code cells.

Modifications:

Code is expanded to add “dominant_emotion” and “subsequent_emotion” columns to the overall_data tables. Merged_emotion rows with null values for some time intervals also added to overall_data table.

Data Visualizations

Prompt 1 – To find the dominant emotion of all the candidates over time

You:

I want you to act as a data scientist. You are given two datasets for each of the 10 candidates. The datasets are obtained from the introductory video submitted by the candidates to a tech startup company which specializes in the use of AI technology for mental health and human resource management solutions. merged_emotion_i.csv and i_updated.csv (for i in range(1,11)) files are given for candidates 1 to 10. merged_emotion_i.csv files contain emotions of the candidate throughout the video. File path: "C:\Users\HP\Desktop\I'mBesideYou\merged_emotion\merged_emotion_i.csv" (for i in range(1,11)) merged_emotion_i.csv information: movie_id - Unique ID for a video image_seq - Image number angry - angry emotion score disgust - disgust emotion score fear - fear emotion score happy - happy emotion score sad - sad emotion score surprise - surprise emotion score neutral - neutral emotion score dominant_emotion - dominant emotion among other emotions gaze - Candidate is looking at the camera or not. 1 for looking and 0 for not looking blink - Eye blink (1 -> blink, 0 -> No blink) eye_offset - Deviation of eye from the camera elapsed_time - Timestamp in seconds (9 means at 9th sec) i_updated.csv files contain scores extracted from the transcripts throughout the video. File Path: "C:\Users\HP\Desktop\I'mBesideYou\transcript_data\i_updated.csv" i_updated.csv information: start - start time of the text end - end time of the text text - transcript spoken from start to end duration positive - positive score negative - negative score neutral - neutral score confident - confidence score hesitant - hesitance score concise - concise score enthusiastic - enthusiasm score speech_speed - Speed of the speech spoken We will obtain insights by performing data visualizations on both the tables and compare among the 10 candidates. Data preprocessing on both the tables performed. Please write a code in jupyter notebook to visualize the dominant emotions over time for all the 10 candidates using line chart from merged_emotion_i.csv files (for i in range(1,11)). What are the insights obtained from the charts obtained?

ChatGPT:

To visualize the dominant emotions over time for all 10 candidates using line charts from the `merged_emotion_i.csv` files, you can follow these steps in a Jupyter Notebook environment. Make sure to replace the file paths accordingly.

First, you'll need to import the necessary libraries:

```
python
import pandas as pd
import matplotlib.pyplot as plt
```

Next, you can write the code to read and visualize the data:

```
python
# Create a list to store the dataframes for each candidate
dfs = []

# Read and store the data for each candidate
for i in range(1, 11):
    file_path = f"C:\\Users\\HP\\Desktop\\I'mBesideYou\\merged_emotion\\merged_emotion_{i}.csv"
    df = pd.read_csv(file_path)
    dfs.append(df)
```

```
# Create a dictionary to map emotion codes to their full names
emotion_mapping = {
    'angry': 'Angry',
    'disgust': 'Disgust',
    'fear': 'Fear',
    'happy': 'Happy',
    'sad': 'Sad',
    'surprise': 'Surprise',
    'neutral': 'Neutral'
}

# Create subplots for each candidate
fig, axs = plt.subplots(5, 2, figsize=(15, 20))
fig.suptitle('Dominant Emotions Over Time for 10 Candidates', fontsize=16)

for i, df in enumerate(dfs):
    row = i // 2
    col = i % 2
    ax = axs[row, col]

    for emotion_code, emotion_name in emotion_mapping.items():
        candidate_emotion_data = df[df['dominant_emotion'] == emotion_code]
        ax.plot(candidate_emotion_data['elapsed_time'], candidate_emotion_data[emotion_code], label=emotion_name)

    ax.set_title(f'Candidate {i + 1}')
    ax.set_xlabel('Time (seconds)')
    ax.set_ylabel('Emotion Score')
    ax.legend()

# Adjust layout and show the plot
```

```
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

This code will create a line chart for each candidate, displaying the dominant emotions over time. The x-axis represents time in seconds, and the y-axis represents the emotion score. Different emotions are represented by different colored lines.

The insights obtained from these charts may include:

1. **Dominant Emotion Trends**: You can observe how the dominant emotions change over time for each candidate. Are there specific moments where certain emotions are more prevalent?
2. **Individual Differences**: Compare the dominant emotion patterns among candidates. Do some candidates consistently exhibit certain emotions, while others show more variability?
3. **Engagement**: Check if there are patterns related to gaze (looking at the camera) or eye blink behavior. Do candidates display different emotions when looking at the camera compared to when they are not?
4. **Emotional Dynamics**: Analyze the dynamics of emotions throughout the videos. Are there moments of transition from one dominant emotion to another, and do these transitions vary among candidates?

These insights can provide valuable information for evaluating the emotional expressions of candidates during their introductory videos, which may be relevant for assessing their suitability for roles in a tech startup specializing in mental health and human resource management solutions.

1. Prompt 2 – Code to compare the emotion scores of all the candidates

You:

Please write a code to calculate the average percentage of all the emotions for each of the candidates. Multiply them with predefined emotion scores and plot a chart to compare the scores of all the candidates obtained (excluding candidates 5 and 6). emotion_scores: 'angry': -1, 'disgust': -1, 'fear': -2, 'happy': 3, 'sad': -1, 'surprise': 1, 'neutral': 2

ChatGPT:

To calculate the average percentage of all the emotions for each of the candidates, multiply them with predefined emotion scores, and plot a chart to compare the scores of all the candidates (excluding candidates 5 and 6), you can use the following code in a Jupyter Notebook:

```
python
import pandas as pd
import matplotlib.pyplot as plt

# Define the emotion scores
emotion_scores = {
    'angry': -1,
    'disgust': -1,
    'fear': -2,
    'happy': 3,
    'sad': -1,
    'surprise': 1,
    'neutral': 2
}

# Create a list to store the dataframes for each candidate
dfs = []

# Read and store the data for each candidate
for i in range(1, 11):
```

```
    file_path = f"C:\\Users\\HP\\Desktop\\I'mBesideYou\\merged_emotion\\merged_emotion_{i}.csv"
    df = pd.read_csv(file_path)
    dfs.append(df)

# Create a dictionary to store the average percentage of emotions multiplied by scores for each candidate
candidate_scores = {}

# Calculate the average percentage of emotions for each candidate
for i, df in enumerate(dfs):
    if i + 1 not in [5, 6]:  # Exclude candidates 5 and 6
        emotion_percentages = df[['angry', 'disgust', 'fear', 'happy', 'sad', 'surprise', 'neutral']]
        average_percentages = emotion_percentages.mean()
        candidate_score = (average_percentages * pd.Series(emotion_scores)).sum()
        candidate_scores[f'Candidate {i + 1}'] = candidate_score
```

```
# Create a bar chart to compare the scores of candidates (excluding candidates 5 and 6)
plt.figure(figsize=(10, 6))
plt.bar(candidate_scores.keys(), candidate_scores.values())
plt.title('Emotion Scores for Candidates (Excluding Candidates 5 and 6)')
plt.xlabel('Candidate')
plt.ylabel('Emotion Score')
plt.xticks(rotation=45)
plt.show()
```

In this code:

1. We define the predefined emotion scores in the `emotion_scores` dictionary.
2. We read the `merged_emotion_i.csv` files for each of the 10 candidates and store them in a list of dataframes.
3. We create a dictionary `candidate_scores` to store the calculated scores for each candidate.
4. We calculate the average percentage of each emotion for each candidate, multiply them by the predefined emotion scores, and sum the results to obtain the candidate's total emotion score.
5. We create a bar chart to visualize and compare the emotion scores of all candidates, excluding candidates 5 and 6.

The resulting bar chart will show the emotion scores for each candidate, and you can use it to compare how candidates performed in terms of their emotional expression.

Prompt 3 - Correlation between gaze and dominant emotion

You:

Please write a code in jupyter notebook to find the correlation between gaze and the dominant_emotion columns in merged_emotion_i.csv files for all the 10 candidates.

ChatGPT:

To find the correlation between the `gaze` column and the `dominant_emotion` columns in the `merged_emotion_i.csv` files for all 10 candidates, you can use the following code in a Jupyter Notebook environment:

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

# Create a list to store the correlation coefficients for each candidate
correlation_coefficients = []

# Read and calculate correlation for each candidate
for i in range(1, 11):
    file_path = f"C:\\Users\\HP\\Desktop\\I'mBesideYou\\merged_emotion\\merged_emotion_{i}.csv"
    df = pd.read_csv(file_path)

    # Calculate the correlation between 'gaze' and each dominant emotion column
    correlation_data = df[['gaze', 'angry', 'disgust', 'fear', 'happy', 'sad', 'surprise', 'neutral']].corr()

    # Store the correlation coefficient between 'gaze' and each emotion in a dictionary
    correlation_dict = {
        'Candidate': f'Candidate {i}',
        'Angry': correlation_data.loc['gaze', 'angry'],
        'Disgust': correlation_data.loc['gaze', 'disgust'],
        'Fear': correlation_data.loc['gaze', 'fear'],
        'Happy': correlation_data.loc['gaze', 'happy'],
        'Sad': correlation_data.loc['gaze', 'sad'],
        'Surprise': correlation_data.loc['gaze', 'surprise'],
        'Neutral': correlation_data.loc['gaze', 'neutral']
    }

    # Append the correlation data to the list
    correlation_coefficients.append(correlation_dict)

# Create a DataFrame from the correlation data
correlation_df = pd.DataFrame(correlation_coefficients)
```

```
# Set the 'Candidate' column as the index
correlation_df.set_index('Candidate', inplace=True)

# Plot a heatmap of correlation coefficients
```

```
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_df, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Between Gaze and Dominant Emotions for 10 Candidates')
plt.show()
```

This code reads each `merged_emotion_i.csv` file for the 10 candidates, calculates the correlation coefficients between the `gaze` column and each dominant emotion column (e.g., `angry`, `disgust`, etc.), and stores the correlation coefficients in a DataFrame. It then creates a heatmap using Seaborn to visualize the correlations.

The resulting heatmap will show the correlation coefficients between gaze and each dominant emotion for each candidate. Positive values indicate a positive correlation, while negative values indicate a negative correlation. This visualization can help you understand the relationship between gaze behavior and dominant emotions for the candidates.

1. Prompt 4 - Comparison of blink rate with dominant emotions for all the candidates, using bar charts

You:

Please write a code in jupyter notebook to compare blink rate and dominant emotion for all the 10 candidates. What are the possible insights that can be obtained from the charts?

ChatGPT:

To compare the blink rate and dominant emotion for all 10 candidates, you can create bar charts that show the average blink rate for each dominant emotion. Here's a code to do that in a Jupyter Notebook:

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

# Create a list to store the dataframes for each candidate
dfs = []

# Read and store the data for each candidate
for i in range(1, 11):
    file_path = f"C:\\Users\\HP\\Desktop\\I'mBesideYou\\merged_emotion\\merged_emotion_{i}.csv"
    df = pd.read_csv(file_path)
    dfs.append(df)

# Create a dictionary to map emotion codes to their full names
emotion_mapping = {
    'angry': 'Angry',
    'disgust': 'Disgust',
    'fear': 'Fear',
    'happy': 'Happy',
    'sad': 'Sad',
    'surprise': 'Surprise',
    'neutral': 'Neutral'
}

# Calculate the average blink rate for each dominant emotion for each candidate
avg_blink_rates = []

for i, df in enumerate(dfs):
    candidate_avg_blink_rate = {}
    for emotion_code, emotion_name in emotion_mapping.items():
        avg_blink_rate = df[df['dominant_emotion'] == emotion_code]['blink'].mean()
        candidate_avg_blink_rate[emotion_name] = avg_blink_rate
    avg_blink_rates.append(candidate_avg_blink_rate)
```

```
# Create a DataFrame from the average blink rate data
avg_blink_rates_df = pd.DataFrame(avg_blink_rates)
avg_blink_rates_df.index = [f'Candidate {i + 1}' for i in range(10)]

# Transpose the DataFrame for plotting
avg_blink_rates_df = avg_blink_rates_df.transpose()

# Create a bar chart to compare blink rate and dominant emotion for each candidate
plt.figure(figsize=(12, 8))
sns.set_palette("husl")
avg_blink_rates_df.plot(kind='bar', ax=plt.gca())
plt.title('Average Blink Rate for Dominant Emotions (by Candidate)')
```

```
plt.xlabel('Dominant Emotion')
plt.ylabel('Average Blink Rate')
plt.xticks(rotation=45)
plt.legend(title='Candidate')
plt.show()
```

This code calculates the average blink rate for each dominant emotion for each candidate and then creates a bar chart to visualize the comparison. Each bar represents a candidate, and the different colors within each bar represent the average blink rate associated with each dominant emotion.

Possible insights that can be obtained from the charts include:

1. **Blink Rate Variation**: You can observe how blink rates vary for different dominant emotions across candidates. Are there emotions that tend to be associated with higher or lower blink rates?
2. **Individual Differences**: Compare how candidates' blink rates differ when expressing different dominant emotions. Do some candidates exhibit more noticeable changes in blink rate based on their dominant emotion?
3. **Emotional Engagement**: Analyze whether there is a correlation between blink rate and dominant emotions. Do candidates blink more or less frequently when expressing specific emotions, and does this pattern vary among candidates?
4. **Consistency**: Examine if certain candidates consistently exhibit similar blink rate patterns for specific emotions, or if there is a wider range of variability in their blink rates.

These insights can provide valuable information about how candidates' blink rates relate to their expressions of dominant emotions and may be relevant in understanding their non-verbal communication during the introductory videos.

Modifications : Seaborn palette changed from ‘husl’ to ‘tab10’

Prompt 5 - Visualize the deviation of the eye offset over time with respect to the average eye offset for all the candidates

You:

Please write a code in jupyter notebook to check deviation of eye offset with respect to average eye offset over time for all the 10 candidates.

ChatGPT:

To check the deviation of eye offset with respect to the average eye offset over time for all 10 candidates, you can calculate and visualize this information using line charts. Here's a code to do that in a Jupyter Notebook:

```
python
import pandas as pd
import matplotlib.pyplot as plt

# Create a list to store the dataframes for each candidate
dfs = []

# Read and store the data for each candidate
for i in range(1, 11):
    file_path = f"C:\\Users\\HP\\Desktop\\I'mBesideYou\\merged_emotion\\merged_emotion_{i}.csv"
    df = pd.read_csv(file_path)
    dfs.append(df)

# Create a figure and axes for subplots
fig, axs = plt.subplots(5, 2, figsize=(15, 20))
fig.suptitle('Eye Offset Deviation from Average Over Time for 10 Candidates', fontsize=16)
```

```
# Calculate and plot eye offset deviation for each candidate
for i, df in enumerate(dfs):
    row = i // 2
    col = i % 2
    ax = axs[row, col]

    # Calculate average eye offset for the candidate
    avg_eye_offset = df['eye_offset'].mean()

    # Calculate eye offset deviation from the average over time
    eye_offset_deviation = df['eye_offset'] - avg_eye_offset

    # Plot eye offset deviation over time
    ax.plot(df['elapsed_time'], eye_offset_deviation)
    ax.set_title(f'Candidate {i + 1}')
    ax.set_xlabel('Time (seconds)')
    ax.set_ylabel('Eye Offset Deviation')
    ax.axhline(0, color='black', linestyle='--', linewidth=1)  # Add a horizontal line at y=0
```

```
# Adjust layout and show the plot
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

In this code:

1. We read the `merged_emotion_i.csv` files for each of the 10 candidates and store them in a list of dataframes.
2. We create subplots for each candidate to visualize the eye offset deviation from the average over time.
3. For each candidate, we calculate the average eye offset (`avg_eye_offset`) and then compute the eye offset deviation by subtracting the average from the actual eye offset values over time.
4. We plot the eye offset deviation over time, with a horizontal dashed line at y=0 to indicate when the eye offset is equal to the average.

The resulting line charts will show how the eye offset deviates from the average over time for each candidate. Insights that can be obtained from these charts may include:

- **Consistency**: Observing whether candidates generally maintain a consistent eye offset (deviation near zero) or exhibit more noticeable deviations over time.
- **Individual Patterns**: Identifying individual patterns where some candidates consistently deviate above or below the average eye offset.
- **Time-Dependent Deviations**: Analyzing if eye offset deviations tend to occur at specific times during the video, which may indicate certain behaviors or reactions.

These insights can be valuable for understanding how candidates' eye movements vary in comparison to their average eye offset during the video presentation.

Modifications: We calculate the average eye offset deviation for each candidate by finding the mean of the absolute deviations from the candidate's average eye offset over time.

Prompt 6 – Visualization and time for all the candidates

You:

Please write a code in jupyter notebook to visualize between gaze (looking or not looking) and time for all the candidates. Also, rank the candidates who looked at the camera for the highest to the least percentage of time, only considering the candidates with total number of rows higher than the median value of the number of rows of all the candidates.

ChatGPT:

To visualize the relationship between gaze (looking or not looking) and time for all candidates and rank the candidates who looked at the camera for the highest to the least percentage of time, considering only candidates with a total number of rows higher than the median value of the number of rows of all the candidates, you can use the following code in a Jupyter Notebook:

```
python
import pandas as pd
import matplotlib.pyplot as plt

# Create a list to store the dataframes for each candidate
dfs = []

# Read and store the data for each candidate
for i in range(1, 11):
    file_path = f"C:\\Users\\HP\\Desktop\\I'mBesideYou\\merged_emotion\\merged_emotion_{i}.csv"
    df = pd.read_csv(file_path)
    dfs.append(df)

# Calculate the median number of rows across all candidates
median_row_count = int(pd.concat(dfs)['elapsed_time'].count() / len(dfs))

# Create a figure and axes for subplots
fig, axs = plt.subplots(5, 2, figsize=(15, 20))
fig.suptitle('Gaze vs. Time for 10 Candidates', fontsize=16)

# Initialize a dictionary to store the percentage of time looking at the camera for each candidate
percentage_time_looking = {}

# Visualize gaze vs. time and calculate percentage time looking for each candidate
for i, df in enumerate(dfs):
    row = i // 2
    col = i % 2
    ax = axs[row, col]

    # Plot gaze (looking or not looking) over time
    ax.plot(df['elapsed_time'], df['gaze'], label='Gaze', color='b')

    ax.set_title(f'Candidate {i + 1}')
    ax.set_xlabel('Time (seconds)')
    ax.set_ylabel('Gaze (1 for looking, 0 for not looking)')

    # Calculate the percentage of time looking at the camera
    total_rows = df.shape[0]
    time_looking = df['gaze'].sum()
    percentage_looking = (time_looking / total_rows) * 100
    percentage_time_looking[f'Candidate {i + 1}'] = percentage_looking

# Adjust layout and show the plot
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()

# Rank candidates by the percentage of time looking at the camera
sorted_candidates = sorted(percentage_time_looking.items(), key=lambda x: x[1], reverse=True)

# Filter candidates with total row counts higher than the median
filtered_candidates = [candidate for candidate, percentage in sorted_candidates if dfs[int(candidate.split()[-1]) - 1].shape

# Print the ranked candidates who looked at the camera for the highest to least percentage of time
print("Candidates ranked by the percentage of time looking at the camera (High to Low):")
for candidate in filtered_candidates:
    print(f"{candidate}: {percentage_time_looking[candidate]:.2f}%")
```

In this code:

1. We read the `merged_emotion_i.csv` files for each of the 10 candidates and store them in a list of dataframes.
2. We calculate the median number of rows across all candidates as `median_row_count`.
3. We create subplots for each candidate to visualize gaze (looking or not looking) over time.
4. We calculate the percentage of time looking at the camera for each candidate and store it in the `percentage_time_looking` dictionary.
5. After plotting the gaze vs. time for all candidates, we rank the candidates by the percentage of time looking at the camera and consider only candidates with a total number of rows higher than the median value.
6. Finally, we print the ranked candidates in descending order of the percentage of time looking at the camera for the selected candidates.

Modifications:
"total_rows = df.shape[0]" changed to "total_rows = df['elapsed_time'].max()".
Bar chart plotted to visualize the rank of candidates on the basis of gaze over time.

Prompt 7 - Comparison of mean positive, confident and enthusiastic scores among all the candidates

You:

From i_updated.csv files (for i in range(1,11)) in transcript_data folder, please write a code to compare mean positive, confident, entusiastic among all the 10 candidates in three different charts. Alsso, show the average values using dotted lines for all the charts

---

ChatGPT:

To compare the mean positive, confident, and enthusiastic scores among all 10 candidates from the `i_updated.csv` files in the `transcript_data` folder and show the average values using dotted lines in three different charts, you can use the following code in a Jupyter Notebook:

```
python
import pandas as pd
import matplotlib.pyplot as plt

# Create a list to store the dataframes for each candidate
dfs = []

# Read and store the data for each candidate
for i in range(1, 11):
    file_path = f"C:\\Users\\HP\\Desktop\\I'mBesideYou\\transcript_data\\i_updated_{i}.csv"
    df = pd.read_csv(file_path)
    dfs.append(df)

# Calculate the mean positive, confident, and enthusiastic scores for each candidate
mean_positive_scores = [df['positive'].mean() for df in dfs]
mean_confident_scores = [df['confident'].mean() for df in dfs]
mean_enthusiastic_scores = [df['enthusiastic'].mean() for df in dfs]

# Calculate the average values for each metric across all candidates
average_mean_positive = sum(mean_positive_scores) / len(mean_positive_scores)
average_mean_confident = sum(mean_confident_scores) / len(mean_confident_scores)
```

```
average_mean_enthusiastic = sum(mean_enthusiastic_scores) / len(mean_enthusiastic_scores)

# Create subplots for three different charts
fig, axs = plt.subplots(3, 1, figsize=(10, 15))

# Chart 1: Mean Positive Scores
axs[0].bar(range(1, 11), mean_positive_scores, label='Mean Positive Scores')
axs[0].axhline(average_mean_positive, color='r', linestyle='--', label='Average Mean Positive', linewidth=2)
axs[0].set_title('Mean Positive Scores Among 10 Candidates')
axs[0].set_xlabel('Candidate')
axs[0].set_ylabel('Mean Positive Score')
axs[0].legend()

# Chart 2: Mean Confident Scores
axs[1].bar(range(1, 11), mean_confident_scores, color='g', label='Mean Confident Scores')
axs[1].axhline(average_mean_confident, color='r', linestyle='--', label='Average Mean Confident', linewidth=2)
axs[1].set_title('Mean Confident Scores Among 10 Candidates')
axs[1].set_xlabel('Candidate')
axs[1].set_ylabel('Mean Confident Score')
axs[1].legend()

# Chart 3: Mean Enthusiastic Scores
axs[2].bar(range(1, 11), mean_enthusiastic_scores, color='b', label='Mean Enthusiastic Scores')
axs[2].axhline(average_mean_enthusiastic, color='r', linestyle='--', label='Average Mean Enthusiastic', linewidth=2)
axs[2].set_title('Mean Enthusiastic Scores Among 10 Candidates')
axs[2].set_xlabel('Candidate')
axs[2].set_ylabel('Mean Enthusiastic Score')
axs[2].legend()

# Adjust layout and show the plots
plt.tight_layout()
plt.show()
```

In this code:

- We read the `i_updated.csv` files for each of the 10 candidates and store them in a list of dataframes.
- We calculate the mean positive, confident, and enthusiastic scores for each candidate and store them in separate lists.
- We calculate the average values for each metric across all candidates.
- We create three different subplots for each metric (positive, confident, and enthusiastic) and plot the mean scores for each candidate as bars.
- Dotted lines are added to indicate the average mean scores for each metric.
- Finally, we display the three charts to compare the mean scores and the average values across all candidates.

Prompt 8 – Comparing the overall sentiment scores of all the candidates

You:

Please write a code to find the sentiment scores of all the candidates, by first finding means of positive, negative, neutral, multiplying scores for positive, negative, neutral (1, -1, 0 respectively). Plot a chart to compare the sentiment scores of all the candidates

---

ChatGPT:

To find the sentiment scores of all the candidates by first finding the means of positive, negative, and neutral scores and then multiplying them by the respective scores (1 for positive, -1 for negative, and 0 for neutral) and plotting a chart to compare the sentiment scores, you can use the following code in a Jupyter Notebook:

```
python
import pandas as pd
import matplotlib.pyplot as plt

# Create a list to store the dataframes for each candidate
dfs = []

# Read and store the data for each candidate
for i in range(1, 11):
    file_path = f"C:\\Users\\HP\\Desktop\\I'mBesideYou\\transcript_data\\i_updated_{i}.csv"
    df = pd.read_csv(file_path)
    dfs.append(df)

# Calculate the mean positive, negative, and neutral scores for each candidate
mean_positive_scores = [df['positive'].mean() for df in dfs]
mean_negative_scores = [df['negative'].mean() for df in dfs]
mean_neutral_scores = [df['neutral'].mean() for df in dfs]

# Define the sentiment scores (1 for positive, -1 for negative, 0 for neutral)
sentiment_scores = {
    'positive': 1,
    'negative': -1,
    'neutral': 0
}

# Calculate the sentiment scores for each candidate
sentiment_scores_candidates = []

for i in range(len(dfs)):
    candidate_score = sum([
        mean_positive_scores[i] * sentiment_scores['positive'],
        mean_negative_scores[i] * sentiment_scores['negative'],
        mean_neutral_scores[i] * sentiment_scores['neutral']
    ])
    sentiment_scores_candidates.append(candidate_score)
```

```
# Create a bar chart to compare the sentiment scores of all candidates
```

```
plt.figure(figsize=(10, 6))
plt.bar(range(1, 11), sentiment_scores_candidates, color='b', alpha=0.7)
plt.title('Sentiment Scores of 10 Candidates')
plt.xlabel('Candidate')
plt.ylabel('Sentiment Score')
plt.xticks(range(1, 11))
plt.tight_layout()
plt.show()
```

In this code:

- We read the `i_updated.csv` files for each of the 10 candidates and store them in a list of dataframes.
- We calculate the mean positive, negative, and neutral scores for each candidate.
- We define the sentiment scores (1 for positive, -1 for negative, and 0 for neutral).
- We calculate the sentiment scores for each candidate by multiplying the mean scores with the respective sentiment scores and summing them.
- Finally, we create a bar chart to visualize and compare the sentiment scores of all candidates. The chart will show how candidates perform in terms of sentiment, with positive scores indicating a more positive sentiment and negative scores indicating a more negative sentiment.

Modifications: Calculation of average sentiment score and plotted on the chart

Prompt 9 – Hesitant vs Concise and Confident vs Concise correlations for all the candidates

You:

Please write a code to show the correlation between hesitant and concise scores for each of the 10 candidates, and between confident and concise in the same plot for each candidates. Check for which candidates (more hesitant, less concise) and (more confident, more concise)

ChatGPT:

To show the correlation between hesitant and concise scores and between confident and concise scores for each of the 10 candidates and check for candidates who are more hesitant and less concise or more confident and more concise, you can use the following code to create individual scatter plots for each candidate in a Jupyter Notebook:

```
python
import pandas as pd
import matplotlib.pyplot as plt

# Create a list to store the dataframes for each candidate
dfs = []

# Read and store the data for each candidate
for i in range(1, 11):
    file_path = f"C:\\Users\\HP\\Desktop\\I'mBesideYou\\transcript_data\\i_updated_{i}.csv"
    df = pd.read_csv(file_path)
    dfs.append(df)

# Create subplots for each candidate's correlation plots
fig, axs = plt.subplots(5, 2, figsize=(12, 15))
fig.suptitle('Correlation Between Hesitant and Concise, and Confident and Concise Scores for 10 Candidates', fontsize=16)

# Initialize lists to store candidates who meet the criteria
more_hesitant_less_concise_candidates = []
more_confident_more_concise_candidates = []

# Calculate and plot the correlations for each candidate
for i, df in enumerate(dfs):
    row = i // 2
    col = i % 2
    ax = axs[row, col]

    # Calculate the correlations
    correlation_hesitant_concise = df['hesitant'].corr(df['concise'])
    correlation_confident_concise = df['confident'].corr(df['concise'])

    # Scatter plot for hesitant vs. concise
    ax.scatter(df['hesitant'], df['concise'], color='b', label=f'Hesitant vs. Concise Correlation: {correlation_hesitant_con

    # Scatter plot for confident vs. concise
    ax.scatter(df['confident'], df['concise'], color='g', label=f'Confident vs. Concise Correlation: {correlation_confident_
```

```
    ax.set_title(f'Candidate {i + 1}')
    ax.set_xlabel('Hesitant / Confident Score')
    ax.set_ylabel('Concise Score')

    # Check for candidates who meet the criteria
    if correlation_hesitant_concise > 0 and correlation_confident_concise < 0:
        more_hesitant_less_concise_candidates.append(f'Candidate {i + 1}')
    elif correlation_hesitant_concise < 0 and correlation_confident_concise > 0:
        more_confident_more_concise_candidates.append(f'Candidate {i + 1}')

# Add legends to the plots
for ax in axs.flat:
    ax.legend()

# Display the plots
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

```
# Print candidates who meet the criteria
print("Candidates who are more hesitant and less concise:", ', '.join(more_hesitant_less_concise_candidates))
print("Candidates who are more confident and more concise:", ', '.join(more_confident_more_concise_candidates))
```

In this code:

- We create subplots for each candidate's correlation plots to visualize the relationships between hesitant and concise scores and between confident and concise scores.
- We calculate the correlations for each candidate and plot them as scatter plots in the subplots.
- We check for candidates who are more hesitant and less concise or more confident and more concise based on the correlations.
- Finally, we display the plots and print the candidates who meet the criteria.

The scatter plots will help you visualize the relationships between hesitant, confident, and concise scores for each candidate and identify candidates who meet the specified criteria.

Prompt 10 – In continuation to prompt 9

You:

Also add a trendline to the hesitant vs concise and confident vs concise for all the subplots

---

ChatGPT:

To add a trendline to the scatter plots for both hesitant vs. concise and confident vs. concise for all the subplots, you can use the Seaborn library, which simplifies the process of adding trendlines. Here's the modified code with trendlines:

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

# Create a list to store the dataframes for each candidate
dfs = []

# Read and store the data for each candidate
for i in range(1, 11):
    file_path = f"C:\\Users\\HP\\Desktop\\I'mBesideYou\\transcript_data\\i_updated_{i}.csv"
    df = pd.read_csv(file_path)
    dfs.append(df)

# Create subplots for each candidate's correlation plots
fig, axs = plt.subplots(5, 2, figsize=(12, 15))
fig.suptitle('Correlation Between Hesitant and Concise, and Confident and Concise Scores for 10 Candidates', fontsize=16)

# Initialize lists to store candidates who meet the criteria
more_hesitant_less_concise_candidates = []
more_confident_more_concise_candidates = []
```

```
# Calculate and plot the correlations for each candidate
for i, df in enumerate(dfs):
    row = i // 2
    col = i % 2
    ax = axs[row, col]

    # Calculate the correlations
    correlation_hesitant_concise = df['hesitant'].corr(df['concise'])
    correlation_confident_concise = df['confident'].corr(df['concise'])

    # Scatter plot for hesitant vs. concise with trendline
    sns.regplot(x='hesitant', y='concise', data=df, ax=ax, color='b', scatter_kws={'alpha':0.5},
                line_kws={'color': 'red', 'label': f'Hesitant vs. Concise Correlation: {correlation_hesitant_concise:.2f}'})
```

```
    # Scatter plot for confident vs. concise with trendline
    sns.regplot(x='confident', y='concise', data=df, ax=ax, color='g', scatter_kws={'alpha':0.5},
                line_kws={'color': 'purple', 'label': f'Confident vs. Concise Correlation: {correlation_confident_concise:.2

    ax.set_title(f'Candidate {i + 1}')
    ax.set_xlabel('Hesitant / Confident Score')
    ax.set_ylabel('Concise Score')
```

```
    # Check for candidates who meet the criteria
    if correlation_hesitant_concise > 0 and correlation_confident_concise < 0:
        more_hesitant_less_concise_candidates.append(f'Candidate {i + 1}')
    elif correlation_hesitant_concise < 0 and correlation_confident_concise > 0:
        more_confident_more_concise_candidates.append(f'Candidate {i + 1}')

# Add legends to the plots
for ax in axs.flat:
    ax.legend()

# Display the plots
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()

# Print candidates who meet the criteria
print("Candidates who are more hesitant and less concise:", ', '.join(more_hesitant_less_concise_candidates))
print("Candidates who are more confident and more concise:", ', '.join(more_confident_more_concise_candidates))
```

In this modified code, we use the `sns.regplot` function from Seaborn to create scatter plots with trendlines for both hesitant vs. concise and confident vs. concise correlations. The trendlines are plotted in red and purple for better visibility. This code will help you visualize the relationships between the scores and easily identify candidates who meet the specified criteria.

Prompt 11 - Comparison of concise score of all the candidates

You:

Please write a code to compare the concise scores of all the candidates with a row chart and display the scores in decreasing order in the chart.

ChatGPT:

To compare the concise scores of all the candidates with a horizontal bar chart (row chart) and display the scores in decreasing order, you can use the following code in a Jupyter Notebook:

```
python
import pandas as pd
import matplotlib.pyplot as plt

# Create a list to store the dataframes for each candidate
dfs = []

# Read and store the data for each candidate
for i in range(1, 11):
    file_path = f"C:\\Users\\HP\\Desktop\\I'mBesideYou\\transcript_data\\i_updated_{i}.csv"
    df = pd.read_csv(file_path)
    dfs.append(df)
```

```
# Calculate the mean concise scores for each candidate
mean_concise_scores = [df['concise'].mean() for df in dfs]

# Create a DataFrame to store candidate names and mean concise scores
candidate_data = pd.DataFrame({
    'Candidate': [f'Candidate {i}' for i in range(1, 11)],
    'Mean Concise Score': mean_concise_scores
})

# Sort the DataFrame by mean concise scores in decreasing order
candidate_data_sorted = candidate_data.sort_values(by='Mean Concise Score', ascending=False)

# Create a horizontal bar chart (row chart) to compare the concise scores
plt.figure(figsize=(10, 6))
plt.barh(candidate_data_sorted['Candidate'], candidate_data_sorted['Mean Concise Score'], color='b', alpha=0.7)
plt.title('Mean Concise Scores of 10 Candidates (Decreasing Order)')
plt.xlabel('Mean Concise Score')
plt.ylabel('Candidate')
plt.gca().invert_yaxis()  # Invert the y-axis to display in decreasing order
plt.tight_layout()
plt.show()
```

In this code:

- We read the `i_updated.csv` files for each of the 10 candidates and store them in a list of dataframes.
- We calculate the mean concise scores for each candidate.
- We create a DataFrame `candidate_data` to store the candidate names and their mean concise scores.
- We sort the DataFrame by mean concise scores in decreasing order using `sort_values`.
- We create a horizontal bar chart (row chart) to visualize and compare the mean concise scores. The chart displays candidate names on the y-axis and their scores on the x-axis in decreasing order.
- We use `plt.gca().invert_yaxis()` to invert the y-axis so that the candidate with the highest mean concise score appears at the top of the chart.

The resulting chart will display the mean concise scores of all candidates in decreasing order, allowing for easy comparison.

Prompt 12 - **Speech speed of candidates over time**

You:

Please write a code to plot speech speeds over time for each of the candidates. Also, mention the average speech speed, speed speech range on the subplots of each of the candidates

---

ChatGPT:

To plot speech speeds over time for each of the candidates and mention the average speech speed and speech speed range on the subplots of each candidate, you can use the following code in a Jupyter Notebook:

```
python
import pandas as pd
import matplotlib.pyplot as plt

# Create a list to store the dataframes for each candidate
dfs = []

# Read and store the data for each candidate
for i in range(1, 11):
    file_path = f"C:\\Users\\HP\\Desktop\\I'mBesideYou\\transcript_data\\i_updated_{i}.csv"
    df = pd.read_csv(file_path)
    dfs.append(df)

# Create subplots for each candidate's speech speed over time
fig, axs = plt.subplots(5, 2, figsize=(12, 15))
fig.suptitle('Speech Speed Over Time for 10 Candidates', fontsize=16)

# Calculate and plot speech speed for each candidate
for i, df in enumerate(dfs):
    row = i // 2
    col = i % 2
    ax = axs[row, col]

    speech_speed = df['speech_speed']
    time = df.index  # Assuming that the index represents time
```

```
    # Plot speech speed over time
    ax.plot(time, speech_speed, color='b', label='Speech Speed')
    ax.set_title(f'Candidate {i + 1}')
    ax.set_xlabel('Time')
    ax.set_ylabel('Speech Speed')

    # Calculate average speech speed and speech range
    average_speed = speech_speed.mean()
    speed_range = speech_speed.max() - speech_speed.min()

    # Annotate the subplot with average speech speed and speech range
    ax.annotate(f'Avg Speed: {average_speed:.2f}\nSpeed Range: {speed_range:.2f}',
                xy=(0.7, 0.7), xycoords='axes fraction',
                bbox=dict(boxstyle="round,pad=0.3", edgecolor="gray", facecolor="white"),
                fontsize=10, color='black')

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

In this code:

- We read the `i_updated.csv` files for each of the 10 candidates and store them in a list of dataframes.
- We create subplots for each candidate's speech speed over time.
- We plot the speech speed over time for each candidate.
- We calculate the average speech speed and speech speed range for each candidate.
- We use `ax.annotate` to annotate each subplot with the average speech speed and speech speed range.

The resulting subplots will show the speech speed over time for each candidate, and each subplot will be annotated with the average speech speed and speech speed range for that candidate.

You:

Please find a code to analyze trends in the subplots of the candidates as obtained above

---

ChatGPT:

To analyze trends in the subplots of the candidates showing speech speed over time, you can use the following code to calculate and visualize trends using moving averages in a Jupyter Notebook:

```
python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Create a list to store the dataframes for each candidate
dfs = []

# Read and store the data for each candidate
for i in range(1, 11):
    file_path = f"C:\\Users\\HP\\Desktop\\I'mBesideYou\\transcript_data\\i_updated_{i}.csv"
    df = pd.read_csv(file_path)
    dfs.append(df)

# Create subplots for each candidate's speech speed over time
fig, axs = plt.subplots(5, 2, figsize=(12, 15))
fig.suptitle('Speech Speed Over Time for 10 Candidates with Trend Analysis', fontsize=16)

# Define the window size for the moving average
window_size = 10  # You can adjust this as needed

# Calculate and plot speech speed with moving average for each candidate
for i, df in enumerate(dfs):
    row = i // 2
    col = i % 2
    ax = axs[row, col]
```

```
    speech_speed = df['speech_speed']
    time = df.index  # Assuming that the index represents time

    # Calculate the moving average
    moving_average = speech_speed.rolling(window=window_size).mean()

    # Plot speech speed over time with moving average
    ax.plot(time, speech_speed, color='b', label='Speech Speed')
    ax.plot(time, moving_average, color='r', label=f'{window_size}-Point Moving Avg', linestyle='--')
    ax.set_title(f'Candidate {i + 1}')
    ax.set_xlabel('Time')
    ax.set_ylabel('Speech Speed')

    # Calculate average speech speed and speed range
    average_speed = speech_speed.mean()
    speed_range = speech_speed.max() - speech_speed.min()

    # Annotate the subplot with average speech speed and speed range
    ax.annotate(f'Avg Speed: {average_speed:.2f}\nSpeed Range: {speed_range:.2f}',
                xy=(0.7, 0.7), xycoords='axes fraction',
                bbox=dict(boxstyle="round,pad=0.3", edgecolor="gray", facecolor="white"),
                fontsize=10, color='black')

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

In this code:

- We calculate a moving average of speech speed using a specified window size (you can adjust `window_size` as needed).
- We plot both the original speech speed and the moving average on the subplots.
- The moving average helps smooth out fluctuations and makes it easier to analyze trends in speech speed over time for each candidate.
- We also annotate each subplot with the average speech speed and speech speed range, as in the previous code.

This code will provide subplots showing speech speed over time with trend analysis using moving averages for each candidate, making it easier to identify trends and patterns in their speech speeds.

1. Prompt 13 – Plotting charts on the basis of the relevancy of education, skills, experience and reasons to join, and finding the recruit_ability scores for each of the candidates

You:

overall_data_i.csv files (for i in range(1,11)) provided Column Description in overall_data_i.csv files: movie_id Unique ID for a video angry - angry emotion score disgust - disgust emotion score fear - fear emotion score happy - happy emotion score sad - sad emotion score surprise - surprise emotion score neutral_emotion - neutral emotion score dominant_emotion - dominant emotion among other emotions subsequent_emotion - next dominant emotion gaze - Candidate is looking at the camera or not. (gaze>0.5 is desirable) blink - Eye blink eye_offset - Deviation of eye from the camera start - start time of the text end - end time of the text text - transcript spoken from start to end duration positive - positive score negative - negative score neutral_sentiment - neutral score confident - confidence score hesitant - hesitance score concise - concise score enthusiastic - enthusiasm score speech_speed - Speed of the speech spoken cleaned_text - cleaned text education - 1 for presence of relevant education, 0 for not skills - 1 for presence of relevant skill, 0 for not experience - 1 for presence of relevant experience, 0 for not reason_to_join - 1 for presence of relevant reasons_to_join, 0 for not Please write a code in jupyter notebook comparing total number of relevant education, skills, experience and reasons_to_join in four separate bar plots. Also, rank the candidates on the basis of recruitability scores obtained by multiplying scores with number of education, skills, experience. Represent the ranking with a comparison chart Scores: education - 1 skills - 2 experience - 3 reasons_to_join - 2

ChatGPT:

To compare the total number of relevant education, skills, experience, and reasons_to_join in four separate bar plots and rank the candidates based on recruitability scores obtained by multiplying the scores with the number of education, skills, experience, you can use the following code in a Jupyter Notebook:

```
python
import pandas as pd
import matplotlib.pyplot as plt

# Create a list to store the dataframes for each candidate
dfs = []
```

```
# Read and store the data for each candidate
for i in range(1, 11):
    file_path = f"C:\\Path\\To\\overall_data_{i}.csv"  # Update with the correct file path
    df = pd.read_csv(file_path)
    dfs.append(df)

# Define the scores for education, skills, experience, and reasons_to_join
scores = {
    'education': 1,
    'skills': 2,
    'experience': 3,
    'reasons_to_join': 2
}
```

```
# Calculate and plot the total number of relevant education, skills, experience, and reasons_to_join
categories = list(scores.keys())
total_counts = []

for category in categories:
    category_counts = [df[category].sum() for df in dfs]
    total_counts.append(category_counts)

# Create separate bar plots for each category
plt.figure(figsize=(12, 8))
for i, category in enumerate(categories):
    plt.subplot(2, 2, i + 1)
    plt.bar(range(1, 11), total_counts[i], color='b', alpha=0.7)
    plt.title(f'Total {category.capitalize()} Counts for 10 Candidates')
    plt.xlabel('Candidate')
    plt.ylabel('Total Count')
    plt.xticks(range(1, 11))

# Calculate recruitability scores for each candidate
recruitability_scores = []

for i in range(10):
    candidate_score = sum([scores[category] * dfs[i][category].sum() for category in categories])
    recruitability_scores.append(candidate_score)

# Rank the candidates based on recruitability scores
sorted_candidates = [f'Candidate {i + 1}' for i in range(10)]
sorted_candidates.sort(key=lambda x: recruitability_scores[sorted_candidates.index(x)], reverse=True)

# Create a comparison chart for candidate rankings
plt.figure(figsize=(10, 6))
plt.bar(sorted_candidates, recruitability_scores, color='g', alpha=0.7)
plt.title('Recruitability Scores for 10 Candidates')
plt.xlabel('Candidate')
plt.ylabel('Recruitability Score')
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```

In this code:

- We read the `overall_data_i.csv` files for each of the 10 candidates and store them in a list of dataframes.
- We define the scores for education, skills, experience, and reasons_to_join as specified.
- We calculate the total counts of relevant education, skills, experience, and reasons_to_join for each candidate and store them in separate lists.
- We create separate bar plots for each category (education, skills, experience, and reasons_to_join) to compare the total counts for all candidates.
- We calculate the recruitability scores for each candidate by multiplying the scores with the number of relevant education, skills, experience, and reasons_to_join.
- We rank the candidates based on their recruitability scores.
- Finally, we create a comparison chart to visualize the rankings of the candidates based on their recruitability scores.

Prompt 14 –

You:

Please write a code to compare the "concise" column values of each of the candidates by taking mean for all the values when "education" value is 1 for overall_data_i.csv (for i in range(1,11)), by plotting one chart

---

ChatGPT:

To compare the "concise" column values of each candidate by taking the mean for all the values when the "education" value is 1 for the `overall_data_i.csv` files (for i in range(1,11)) and plot a single chart, you can use the following code in a Jupyter Notebook:

```
python
import pandas as pd
import matplotlib.pyplot as plt

# Create a list to store the dataframes for each candidate
dfs = []

# Read and store the data for each candidate
for i in range(1, 11):
    file_path = f"C:\\Path\\To\\overall_data_{i}.csv"  # Update with the correct file path
    df = pd.read_csv(file_path)
    dfs.append(df)

# Calculate the mean "concise" values for each candidate where "education" is 1
mean_concise_values = []

for df in dfs:
    relevant_education_df = df[df['education'] == 1]
    mean_concise = relevant_education_df['concise'].mean()
    mean_concise_values.append(mean_concise)
```

```
for df in dfs:
    relevant_education_df = df[df['education'] == 1]
    mean_concise = relevant_education_df['concise'].mean()
    mean_concise_values.append(mean_concise)

# Create a bar chart to compare the mean "concise" values for all candidates
plt.figure(figsize=(10, 6))
plt.bar(range(1, 11), mean_concise_values, color='b', alpha=0.7)
plt.title('Mean "Concise" Values for Candidates with Education=1')
plt.xlabel('Candidate')
plt.ylabel('Mean "Concise" Value')
plt.xticks(range(1, 11))
plt.tight_layout()
plt.show()
```

In this code:

- We read the `overall_data_i.csv` files for each of the 10 candidates and store them in a list of dataframes.
- We calculate the mean "concise" values for each candidate where the "education" value is 1. This filters the data to consider only rows where education is relevant.
- We create a bar chart to compare the mean "concise" values for all candidates with education equal to 1. Each candidate is represented on the x-axis, and their corresponding mean "concise" value is shown on the y-axis.

This chart will help you compare how candidates perform in terms of "concise" values when education is considered relevant.

Prompt 15 – In continuation with above prompt

You:

Now, please make a similar bar plot but this time, also add mean values of "confident" and "enthusiastic" columns along with "concise". file_path = f"C:\\Users\\HP\\Desktop\\I\'mBesideYou\\overall_data\\overall_data_{i}.csv" education_score : concise - 2, confident - 2, enthusiastic - 1. find the education_score for each of the candidates and mention it on the existing bar plot

---

ChatGPT:

To create a bar plot comparing the mean values of "concise," "confident," and "enthusiastic" columns for candidates with education equal to 1 and also display the education_score for each candidate on the same plot, you can use the following code in a Jupyter Notebook:

```
python
import pandas as pd
import matplotlib.pyplot as plt

# Create a list to store the dataframes for each candidate
dfs = []

# Read and store the data for each candidate
for i in range(1, 11):
    file_path = f"C:\\Users\\HP\\Desktop\\I'mBesideYou\\overall_data\\overall_data_{i}.csv"
    df = pd.read_csv(file_path)
    dfs.append(df)

# Define education_score weights
education_score_weights = {
    'concise': 2,
    'confident': 2,
    'enthusiastic': 1
}

# Calculate the education_score for each candidate where "education" is 1
education_scores = []

for df in dfs:
    relevant_education_df = df[df['education'] == 1]
    education_score = sum(
        education_score_weights[column] * relevant_education_df[column].mean()
        for column in education_score_weights
    )
    education_scores.append(education_score)

# Calculate the mean values for "concise," "confident," and "enthusiastic" columns
mean_concise_values = [df[df['education'] == 1]['concise'].mean() for df in dfs]
mean_confident_values = [df[df['education'] == 1]['confident'].mean() for df in dfs]
mean_enthusiastic_values = [df[df['education'] == 1]['enthusiastic'].mean() for df in dfs]

# Create a bar chart to compare the mean values and education_scores for all candidates
plt.figure(figsize=(12, 8))
x = range(1, 11)
width = 0.2

plt.bar([i - width for i in x], mean_concise_values, width=width, color='b', alpha=0.7, label='Concise')
plt.bar(x, mean_confident_values, width=width, color='g', alpha=0.7, label='Confident')
plt.bar([i + width for i in x], mean_enthusiastic_values, width=width, color='r', alpha=0.7, label='Enthusiastic')

plt.title('Mean Values of "Concise," "Confident," and "Enthusiastic" with Education=1')
plt.xlabel('Candidate')
plt.ylabel('Mean Value')
plt.xticks(x, [f'Candidate {i}' for i in x])
plt.legend(loc='upper left')

# Add education_scores as text annotations
for i, score in enumerate(education_scores):
    plt.annotate(f'Education Score: {score:.2f}', (x[i], max(mean_concise_values[i], mean_confident_values[i], mean_enthusia

plt.tight_layout()
plt.show()
```

In this code:

- We read the `overall_data_i.csv` files for each of the 10 candidates and store them in a list of dataframes.
- We define the education_score_weights for "concise," "confident," and "enthusiastic" as specified.
- We calculate the education_score for each candidate where the "education" value is 1 by multiplying the mean values of the relevant columns by their respective weights.
- We calculate the mean values for "concise," "confident," and "enthusiastic" columns for candidates with education equal to 1.
- We create a bar chart with three sets of bars (for "concise," "confident," and "enthusiastic") and annotate each candidate's bar with their education_score.

This chart will help you compare the mean values of "concise," "confident," and "enthusiastic" for candidates with education equal to 1 and also display their education scores.

Prompt 16 – In continuation with above prompt

You:

Now, again perform the previous operation but for only when skills value is 1. skills_sentiment weights : concise - 2, confident - 3, enthusiastic - 1. find the skills_sentiment for each of the candidates and mention it on the existing bar plot.

ChatGPT:

To perform a similar operation as before but only for candidates with "skills" value equal to 1 and calculate the "skills_sentiment" for each candidate, you can use the following code. This code will also display the "skills_sentiment" along with the existing bar plot that compares the mean values of "concise," "confident," and "enthusiastic":

```
python
import pandas as pd
import matplotlib.pyplot as plt

# Create a list to store the dataframes for each candidate
dfs = []

# Read and store the data for each candidate
for i in range(1, 11):
    file_path = f"C:\\Users\\HP\\Desktop\\I'mBesideYou\\overall_data\\overall_data_{i}.csv"
    df = pd.read_csv(file_path)
    dfs.append(df)

# Define skills_sentiment weights
skills_sentiment_weights = {
    'concise': 2,
    'confident': 3,
    'enthusiastic': 1
}

# Calculate the skills_sentiment for each candidate where "skills" is 1
skills_sentiments = []
```

```
for df in dfs:
    relevant_skills_df = df[df['skills'] == 1]
    skills_sentiment = sum(
        skills_sentiment_weights[column] * relevant_skills_df[column].mean()
        for column in skills_sentiment_weights
    )
    skills_sentiments.append(skills_sentiment)

# Calculate the mean values for "concise," "confident," and "enthusiastic" columns
mean_concise_values = [df[df['skills'] == 1]['concise'].mean() for df in dfs]
mean_confident_values = [df[df['skills'] == 1]['confident'].mean() for df in dfs]
mean_enthusiastic_values = [df[df['skills'] == 1]['enthusiastic'].mean() for df in dfs]

# Create a bar chart to compare the mean values and skills_sentiments for all candidates
plt.figure(figsize=(12, 8))
x = range(1, 11)
width = 0.2
```

```
plt.bar([i - width for i in x], mean_concise_values, width=width, color='b', alpha=0.7, label='Concise')
plt.bar(x, mean_confident_values, width=width, color='g', alpha=0.7, label='Confident')
plt.bar([i + width for i in x], mean_enthusiastic_values, width=width, color='r', alpha=0.7, label='Enthusiastic')

plt.title('Mean Values of "Concise," "Confident," and "Enthusiastic" with Skills=1')
plt.xlabel('Candidate')
plt.ylabel('Mean Value')
plt.xticks(x, [f'Candidate {i}' for i in x])
plt.legend(loc='upper left')

# Add skills_sentiments as text annotations
for i, sentiment in enumerate(skills_sentiments):
    plt.annotate(f'Skills Sentiment: {sentiment:.2f}', (x[i], max(mean_concise_values[i], mean_confident_values[i], mean_ent

plt.tight_layout()
plt.show()
```

In this code:

- We read the `overall_data_i.csv` files for each of the 10 candidates and store them in a list of dataframes.
- We define the `skills_sentiment_weights` for "concise," "confident," and "enthusiastic" as specified.
- We calculate the `skills_sentiment` for each candidate where the "skills" value is 1 by multiplying the mean values of the relevant columns by their respective weights.
- We calculate the mean values for "concise," "confident," and "enthusiastic" columns for candidates with "skills" equal to 1.
- We create a bar chart with three sets of bars (for "concise," "confident," and "enthusiastic") and annotate each candidate's bar with their "skills_sentiment."

This chart will help you compare the mean values of "concise," "confident," and "enthusiastic" for candidates with "skills" equal to 1 and also display their "skills_sentiment."

Prompt 17 – In continuation with above prompt

You:

Plot the education_score, skills_sentiment as obtained above for each of the candidates to compare all of the candidates.

ChatGPT:

To plot the education_score and skills_sentiment for each of the candidates to compare all of the candidates, you can use the following code in a Jupyter Notebook:

```
python
import pandas as pd
import matplotlib.pyplot as plt

# Create a list to store the dataframes for each candidate
dfs = []

# Read and store the data for each candidate
for i in range(1, 11):
    file_path = f"C:\\Users\\HP\\Desktop\\I'mBesideYou\\overall_data\\overall_data_{i}.csv"
    df = pd.read_csv(file_path)
    dfs.append(df)

# Define education_score weights
education_score_weights = {
    'concise': 2,
    'confident': 2,
    'enthusiastic': 1
}

# Define skills_sentiment weights
skills_sentiment_weights = {
    'concise': 2,
    'confident': 3,
    'enthusiastic': 1
}

# Calculate the education_score for each candidate where "education" is 1
education_scores = []

for df in dfs:
    relevant_education_df = df[df['education'] == 1]
    education_score = sum(
        education_score_weights[column] * relevant_education_df[column].mean()
        for column in education_score_weights
```

```
        for column in education_score_weights
    )
    education_scores.append(education_score)

# Calculate the skills_sentiment for each candidate where "skills" is 1
skills_sentiments = []

for df in dfs:
    relevant_skills_df = df[df['skills'] == 1]
    skills_sentiment = sum(
        skills_sentiment_weights[column] * relevant_skills_df[column].mean()
        for column in skills_sentiment_weights
    )
    skills_sentiments.append(skills_sentiment)

# Create a bar chart to compare education_score and skills_sentiment for all candidates
candidates = [f'Candidate {i}' for i in range(1, 11)]
x = range(1, 11)
```

```
plt.figure(figsize=(10, 6))
plt.bar(x, education_scores, width=0.4, color='b', alpha=0.7, label='Education Score')
plt.bar([i + 0.4 for i in x], skills_sentiments, width=0.4, color='g', alpha=0.7, label='Skills Sentiment')

plt.title('Education Score vs. Skills Sentiment for 10 Candidates')
plt.xlabel('Candidate')
plt.ylabel('Score')
plt.xticks([i + 0.2 for i in x], candidates)
plt.legend(loc='upper left')

plt.tight_layout()
plt.show()
```

In this code:

- We calculate the education_score and skills_sentiment for each candidate as previously explained.
- We create a bar chart to compare education_score and skills_sentiment for all candidates. Each candidate is represented on the x-axis, and their corresponding scores are shown on the y-axis.
- The blue bars represent the education_score, and the green bars represent the skills_sentiment.

This chart allows you to visually compare the education_score and skills_sentiment for all candidates.

Prompt 18 – In continuation with above prompt

now, also calculate experience_sentiment (when experience value is 1) (concise - 2, confident - 3, enthusiastic - 2) and reasons_to_join_sentiment (when reasons_to_join value is 1) (concise - 1, confident - 2, enthusiastic - 3) for each of the candidates. Add these to the above plot. Calculate the overall_sentiment_score (education_score - 1, skills_sentiment - 2, experience_sentiment - 4, reasons_to_join_sentiment - 3) and mention it on the existing plot

---

ChatGPT:

To calculate the experience_sentiment and reasons_to_join_sentiment for each candidate and add them to the existing plot along with the overall_sentiment_score, you can use the following code in a Jupyter Notebook:

```
python
import pandas as pd
import matplotlib.pyplot as plt

# Create a list to store the dataframes for each candidate
dfs = []

# Read and store the data for each candidate
for i in range(1, 11):
    file_path = f"C:\\Users\\HP\\Desktop\\I'mBesideYou\\overall_data\\overall_data_{i}.csv"
    df = pd.read_csv(file_path)
    dfs.append(df)

# Define weights for sentiment scores
sentiment_weights = {
    'education': {'concise': 2, 'confident': 2, 'enthusiastic': 1},
    'skills': {'concise': 2, 'confident': 3, 'enthusiastic': 1},
    'experience': {'concise': 2, 'confident': 3, 'enthusiastic': 2},
    'reasons_to_join': {'concise': 1, 'confident': 2, 'enthusiastic': 3}
}
```

```
# Calculate the sentiment scores for each candidate
sentiment_scores = {candidate: {} for candidate in range(1, 11)}

for i, df in enumerate(dfs):
    for key in sentiment_weights:
        relevant_df = df[df[key] == 1]
        sentiment_score = sum(
            sentiment_weights[key][column] * relevant_df[column].mean()
            for column in sentiment_weights[key]
        )
        sentiment_scores[i + 1][key] = sentiment_score

# Calculate the overall_sentiment_score for each candidate
overall_sentiment_scores = {}
```

```
for candidate in sentiment_scores:
    overall_sentiment_score = sum(
        (sentiment_scores[candidate][key] - weight) * sentiment_weights[key]['confident']
        for key, weight in {
            'education': 1,
            'skills': 2,
            'experience': 4,
            'reasons_to_join': 3
        }.items()
    )
    overall_sentiment_scores[candidate] = overall_sentiment_score

# Create a bar chart to compare sentiment scores for all candidates
candidates = [f'Candidate {i}' for i in range(1, 11)]
x = range(1, 11)

plt.figure(figsize=(12, 8))
plt.bar(x, [sentiment_scores[candidate]['education'] for candidate in sentiment_scores], width=0.2, color='b', alpha=0.7, la
plt.bar([i + 0.2 for i in x], [sentiment_scores[candidate]['skills'] for candidate in sentiment_scores], width=0.2, color='g
plt.bar([i + 0.4 for i in x], [sentiment_scores[candidate]['experience'] for candidate in sentiment_scores], width=0.2, colo
plt.bar([i + 0.6 for i in x], [sentiment_scores[candidate]['reasons_to_join'] for candidate in sentiment_scores], width=0.2,

# Add overall_sentiment_scores as text annotations
for i, score in enumerate(overall_sentiment_scores.values()):
    plt.annotate(f'Overall Sentiment Score: {score:.2f}', (x[i], max(sentiment_scores[i + 1].values()) + 0.02))

plt.title('Sentiment Scores for 10 Candidates')
plt.xlabel('Candidate')
plt.ylabel('Score')
plt.xticks([i + 0.3 for i in x], candidates)
plt.legend(loc='upper left')

plt.tight_layout()
plt.show()
```

In this code:

- We define weights for sentiment scores for each category: education, skills, experience, and reasons_to_join.
- We calculate the sentiment scores for each candidate for each category based on the provided weights.
- We calculate the overall_sentiment_score for each candidate by combining the sentiment scores with the specified weights.
- We create a bar chart to compare sentiment scores for all candidates, with different colors representing different sentiment categories: education, skills, experience, and reasons to join.
- We annotate each candidate's bar with their overall_sentiment_score.

This chart allows you to visually compare the sentiment scores and overall_sentiment_scores for all candidates across different categories.

Prompt 19 - Finding the correlation between “positive emotion” and “positive sentiment” for all the candidates

You:

Then, find "positive emotion" value by taking the mean of happy, surprise and neutral_emotion from every row in overall_data_i.csv files Then, find "positive sentiment" value by taking the mean of positive, neutral_sentiment and enthusiastic from every row in overall_data_i.csv files Plot the "positive emotion" and "positive sentiment" over time (taking "start" and "end" as time interval of each row, for each positive emotion and positive sentiment) for all the 10 candidates (excluding candidates 5 and 6). Plot positive emotion and positive sentiment in one chart per candidate

---

ChatGPT:

To calculate the "positive emotion" and "positive sentiment" values and plot them over time for all the 10 candidates (excluding candidates 5 and 6), you can use the following code in a Jupyter Notebook:

```
python
import pandas as pd
import matplotlib.pyplot as plt

# Create a list to store the dataframes for each candidate
dfs = []

# Read and store the data for each candidate
```

```
for i in range(1, 11):
    file_path = f"C:\\Users\\HP\\Desktop\\I'mBesideYou\\overall_data\\overall_data_{i}.csv"
    df = pd.read_csv(file_path)
    dfs.append(df)

# Define a function to calculate positive emotion and positive sentiment
def calculate_positive_values(row):
    positive_emotion = (row['happy'] + row['surprise'] + row['neutral_emotion']) / 3
    positive_sentiment = (row['positive'] + row['neutral_sentiment'] + row['enthusiastic']) / 3
    return positive_emotion, positive_sentiment

# Create subplots for each candidate (excluding candidates 5 and 6)
plt.figure(figsize=(15, 10))
for i, df in enumerate(dfs):
    if i + 1 not in [5, 6]:
        plt.subplot(3, 3, i + 1)
        plt.title(f'Candidate {i + 1}')
        plt.xlabel('Time')
        plt.ylabel('Value')
```

```
        # Calculate positive emotion and positive sentiment for each row
        positive_emotions = []
        positive_sentiments = []
        for _, row in df.iterrows():
            pos_emotion, pos_sentiment = calculate_positive_values(row)
            positive_emotions.append(pos_emotion)
            positive_sentiments.append(pos_sentiment)

        # Plot positive emotion and positive sentiment over time
        plt.plot(df['start'], positive_emotions, label='Positive Emotion', color='b')
        plt.plot(df['start'], positive_sentiments, label='Positive Sentiment', color='g')
        plt.legend(loc='upper right')

plt.tight_layout()
plt.show()
```

In this code:

- We read the `overall_data_i.csv` files for each of the 10 candidates and store them in a list of dataframes.
- We define a function `calculate_positive_values` to calculate the "positive emotion" and "positive sentiment" values for each row.
- We iterate through each candidate's dataframe, calculate the positive emotion and positive sentiment values for each row, and store them in separate lists.
- We create subplots for each candidate (excluding candidates 5 and 6) and plot the "positive emotion" and "positive sentiment" values over time, using the "start" column as the time interval.

The resulting chart will display positive emotion and positive sentiment trends over time for each of the 10 candidates (excluding candidates 5 and 6).

Prompt 20 – In continuation with the above prompt

You:

Please write a code to calculate the Pearson correlation coefficient between positive emotion and positive sentiment for each candidate's data. Also, visualize the relationship using scatterplots with a regression line for each candidate

---

ChatGPT:

To calculate the Pearson correlation coefficient between positive emotion and positive sentiment for each candidate's data and visualize the relationship using scatterplots with a regression line for each candidate, you can use the following code:

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

# Create a list to store the dataframes for each candidate
dfs = []

# Read and store the data for each candidate
for i in range(1, 11):

    file_path = f"C:\\Users\\HP\\Desktop\\I'mBesideYou\\overall_data\\overall_data_{i}.csv"
    df = pd.read_csv(file_path)
    dfs.append(df)

# Define a function to calculate positive emotion and positive sentiment
def calculate_positive_values(row):
    positive_emotion = (row['happy'] + row['surprise'] + row['neutral_emotion']) / 3
    positive_sentiment = (row['positive'] + row['neutral_sentiment'] + row['enthusiastic']) / 3
    return positive_emotion, positive_sentiment

# Calculate the Pearson correlation coefficient and visualize the relationship for each candidate
correlations = []

for i, df in enumerate(dfs):
    positive_emotions = []
    positive_sentiments = []

    for _, row in df.iterrows():
        pos_emotion, pos_sentiment = calculate_positive_values(row)
        positive_emotions.append(pos_emotion)
        positive_sentiments.append(pos_sentiment)

    # Calculate the Pearson correlation coefficient
    correlation = pd.Series(positive_emotions).corr(pd.Series(positive_sentiments))
    correlations.append(correlation)

    # Plot a scatterplot with a regression line
    plt.figure(figsize=(8, 6))
    sns.regplot(x=positive_emotions, y=positive_sentiments, scatter_kws={'alpha':0.5})
    plt.title(f'Candidate {i + 1} - Correlation: {correlation:.2f}')
    plt.xlabel('Positive Emotion')
    plt.ylabel('Positive Sentiment')
    plt.grid(True)
    plt.show()

# Print the correlations for each candidate
for i, correlation in enumerate(correlations):
    print(f'Candidate {i + 1} - Correlation: {correlation:.2f}')
```

This code calculates the Pearson correlation coefficient between positive emotion and positive sentiment for each candidate and visualizes the relationship using scatterplots with regression lines. The correlation values are displayed in the titles of the scatterplots, and they are also printed out at the end of the analysis.
