EDA Analysis Report

There were two datasets that were provided to us:

1) Emotion data
2) Transcript data

- Emotion data:

Importing library to connect drive to colab:

```
from google.colab import drive
drive.mount('/content/drive')
%cd /content/drive/MyDrive/imby_project/Emotion_score
```

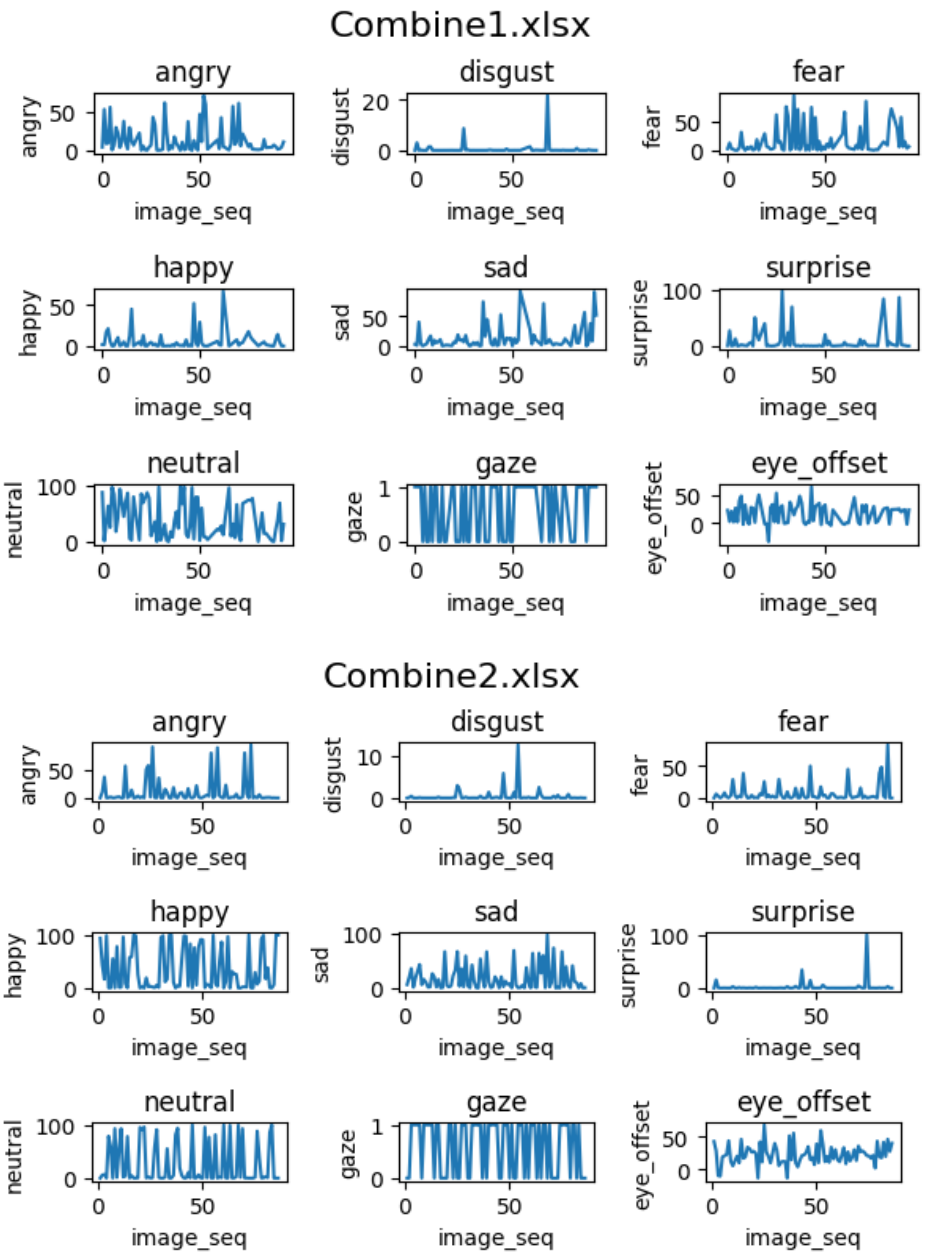
Installing and importing autotime library to see execution time of each cell

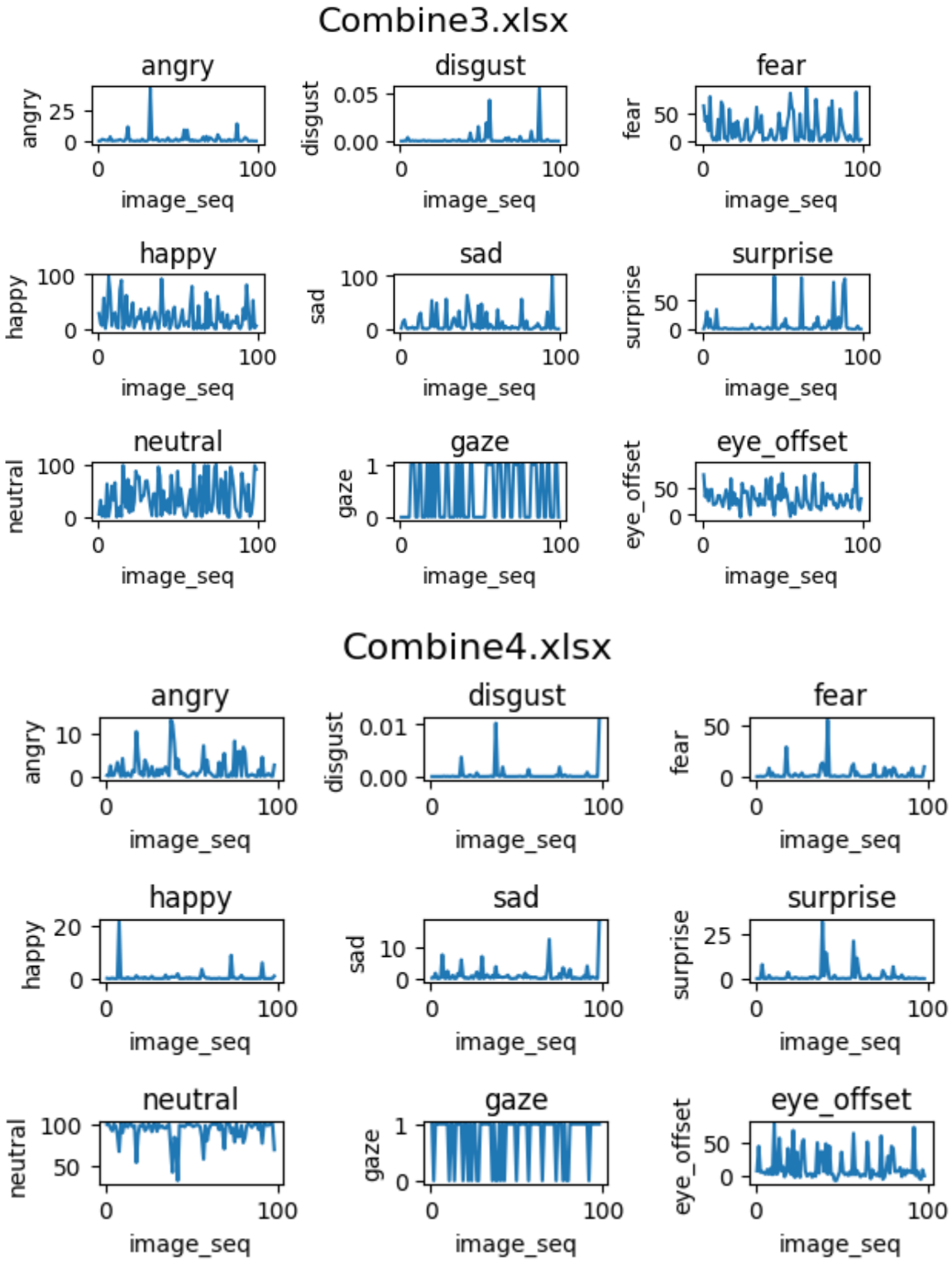
```
!pip install ipython-autotime
%load_ext autotime
```

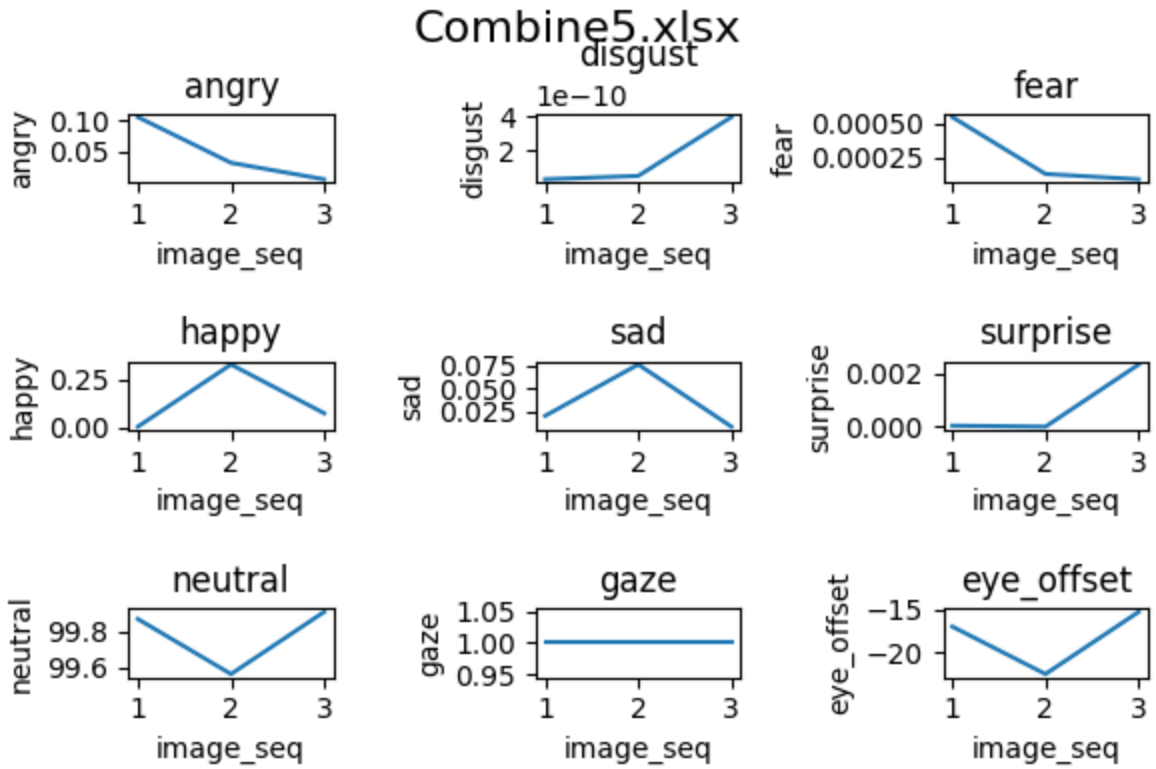
Plot of all emotions of every **combined** excel file with respect to image sequence:

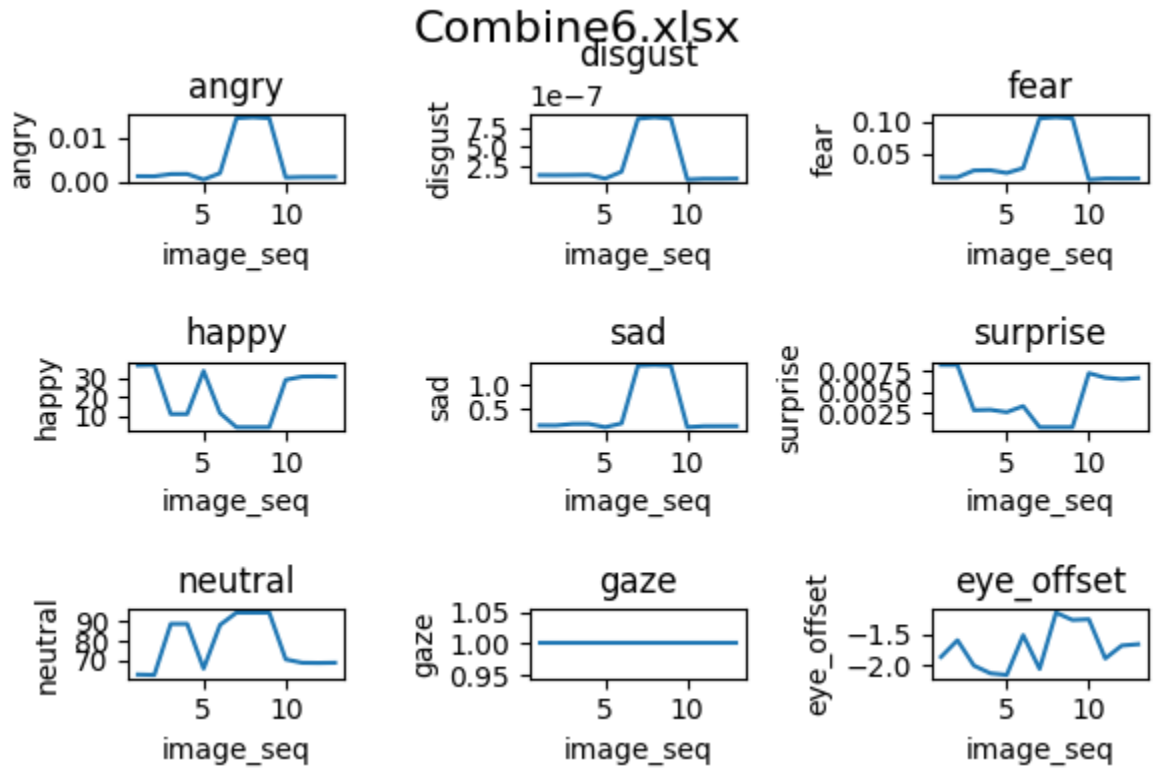
```
import pandas as pd
import matplotlib.pyplot as plt

filename =
['Combine1.xlsx','Combine2.xlsx','Combine3.xlsx','Combine4.xlsx','Combine5
.xlsx','Combine6.xlsx','Combine7.xlsx','Combine8.xlsx','Combine9.xlsx','Co
mbine10.xlsx']

for file in filename:
    df = pd.read_excel(file)
    emotions = ['angry', 'disgust', 'fear', 'happy', 'sad', 'surprise',
'neutral','gaze','eye_offset']

    plt.figure(figsize=(6, 4))  # Adjusting the figure size here

    plt.suptitle(file, fontsize=16)  # Showing the filename as the title

    for i, emotion in enumerate(emotions):
        plt.subplot(3, 3, i + 1)
        plt.plot(df['image_seq'], df[emotion])
```

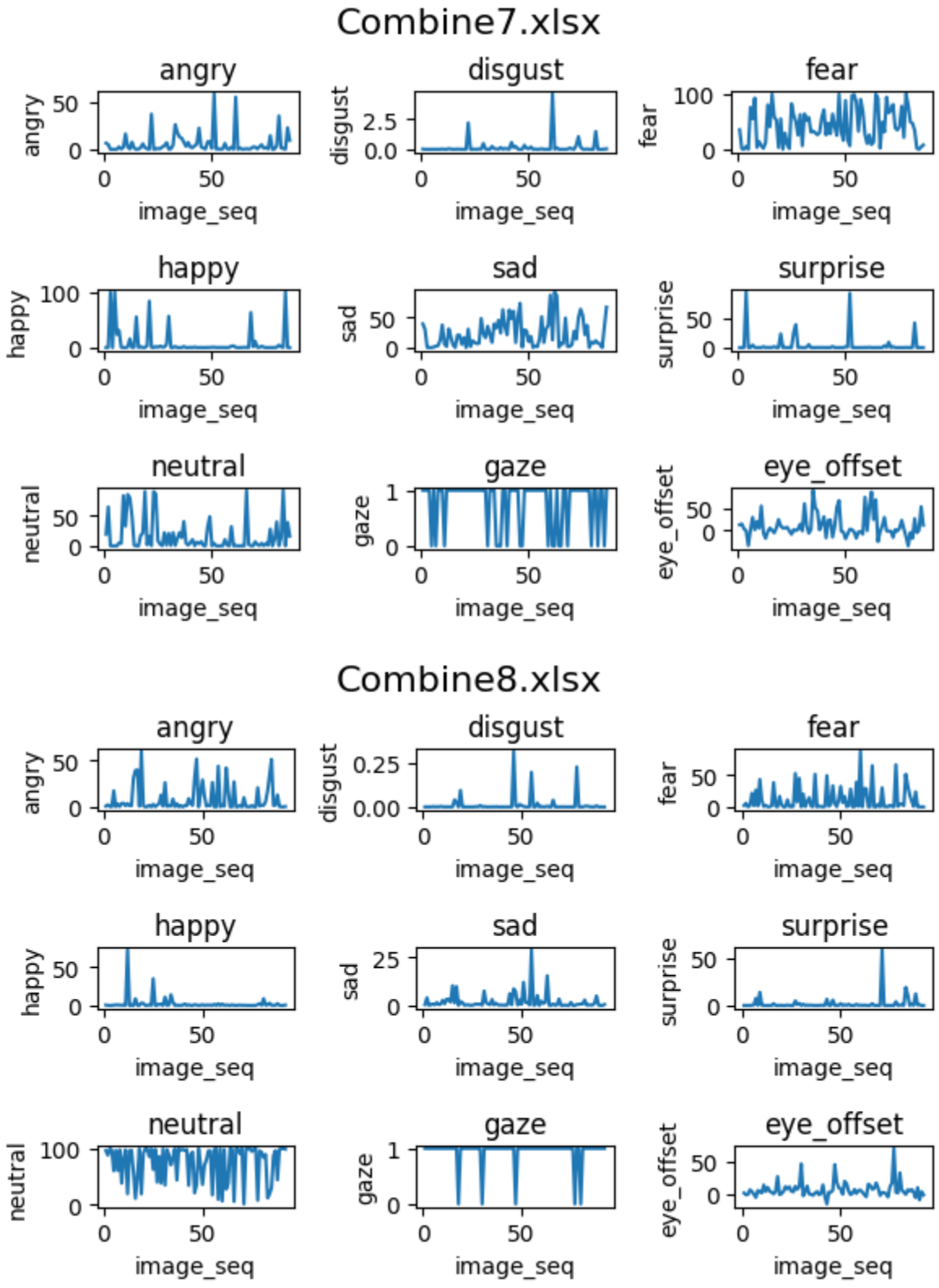
```
        plt.title(emotion)
        plt.xlabel('image_seq')
        plt.ylabel(emotion)
        plt.tight_layout()  # Adjusting spacing between subplots

    plt.subplots_adjust(top=0.85)  # Adjusting the title's position
    plt.show()
```

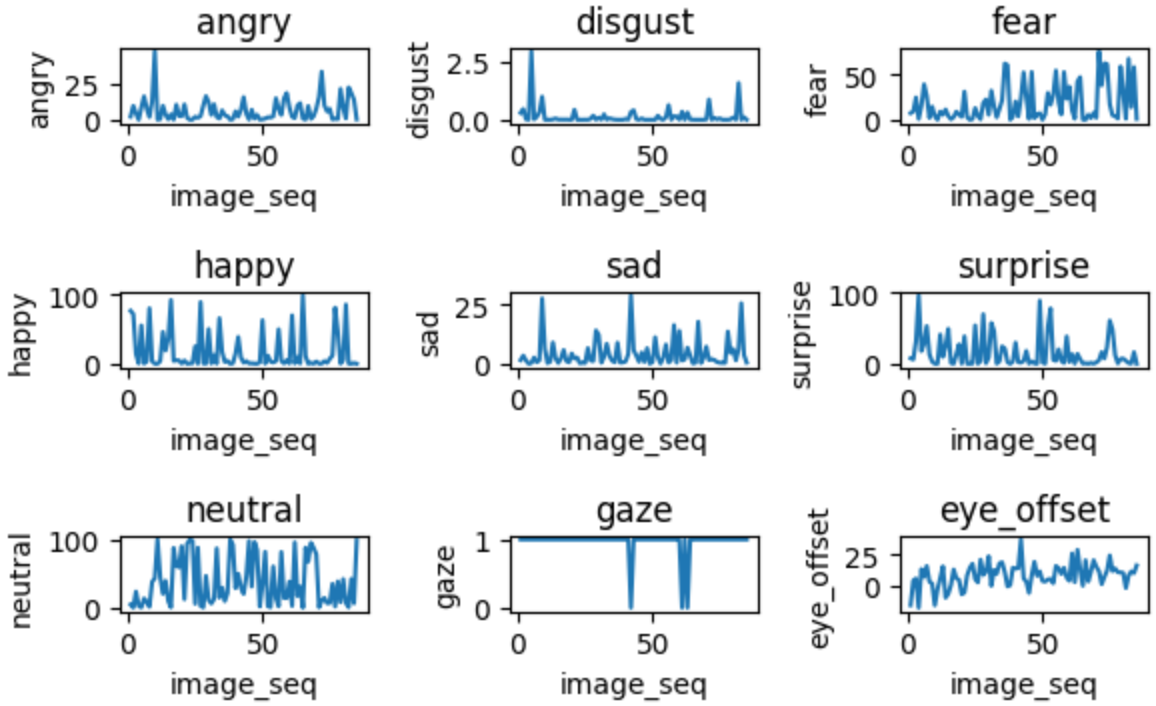


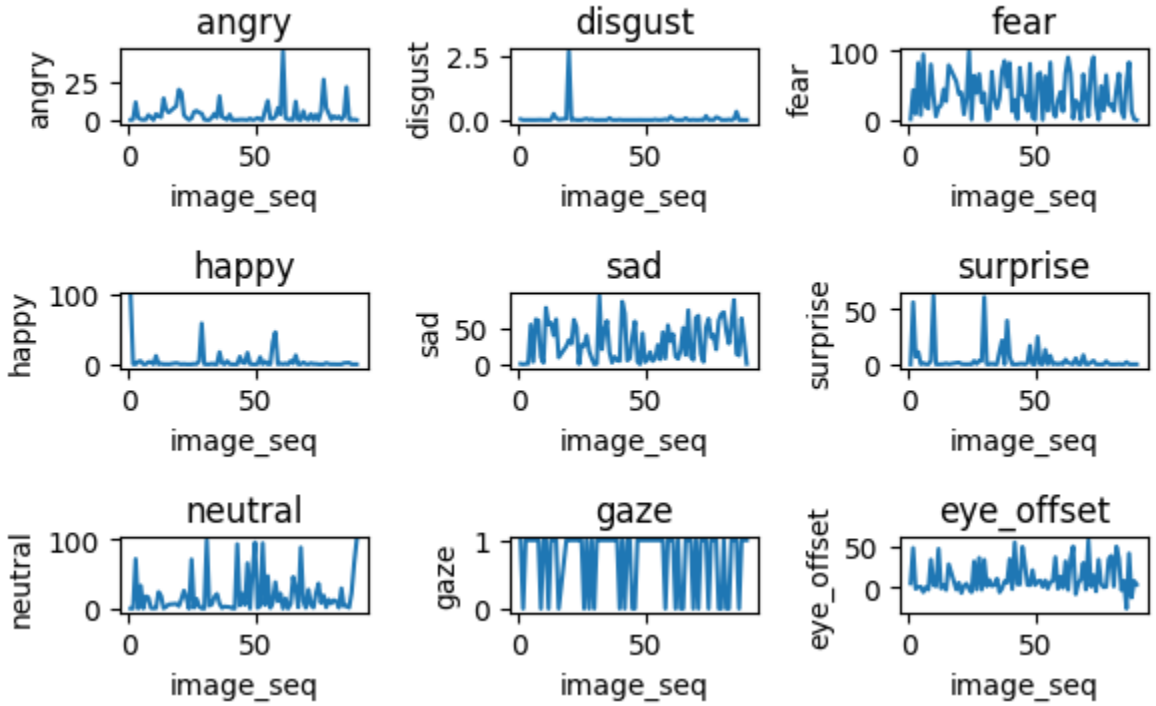

Combine3.xlsx
angry
disgust
fear
happy
sad
surprise
neutral
gaze
eye_offset
image_seq
Combine4.xlsx
angry
disgust
fear
happy
sad
surprise
neutral
gaze
eye_offset
image_seq


Combine5.xlsx
angry
disgust
1e−10
fear
happy
sad
surprise
neutral
gaze
eye_offset
image_seq

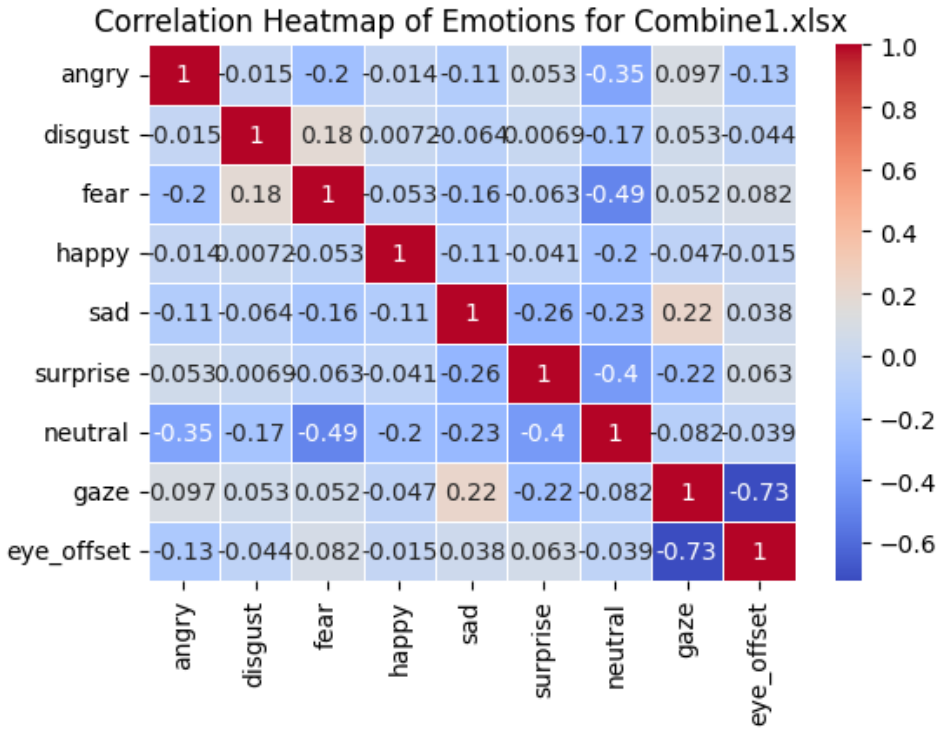

Combine6.xlsx
angry
disgust
1e−7
fear
happy
sad
surprise
neutral
gaze
eye_offset
image_seq

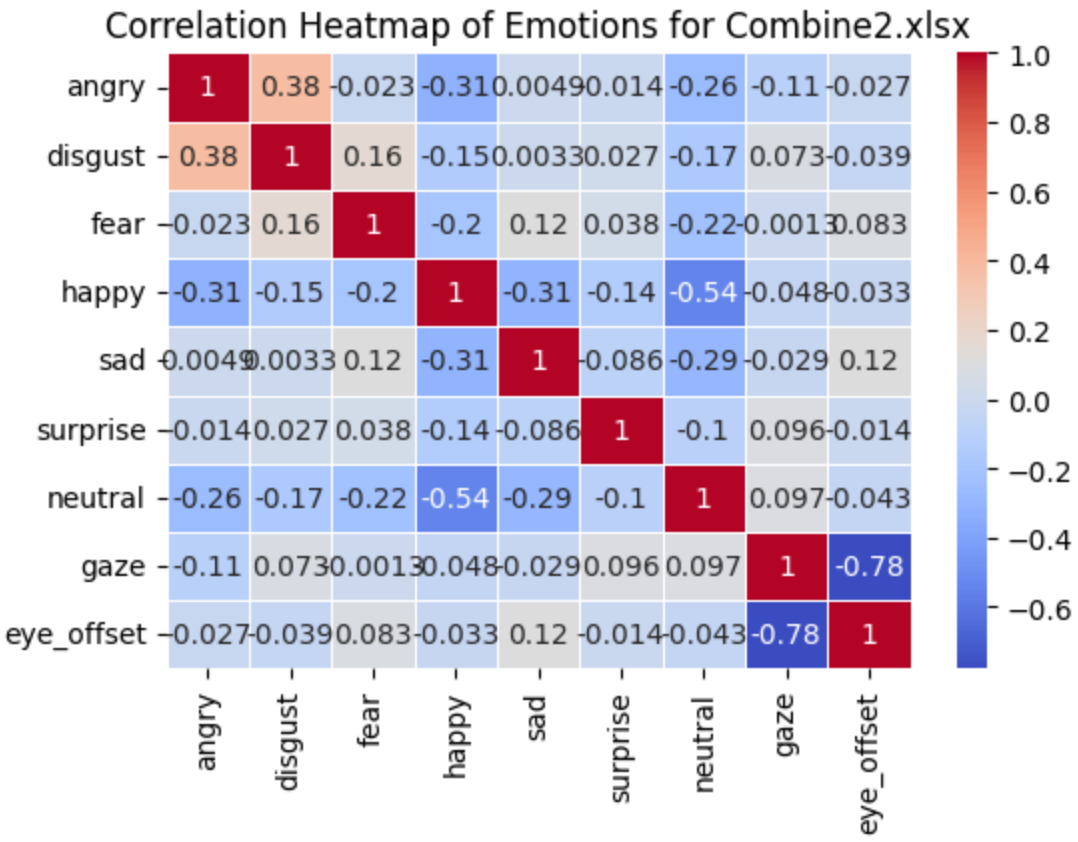

Combine7.xlsx
angry
disgust
fear
happy
sad
surprise
neutral
gaze
eye_offset
image_seq
Combine8.xlsx
angry
disgust
fear
happy
sad
surprise
neutral
gaze
eye_offset
image_seq

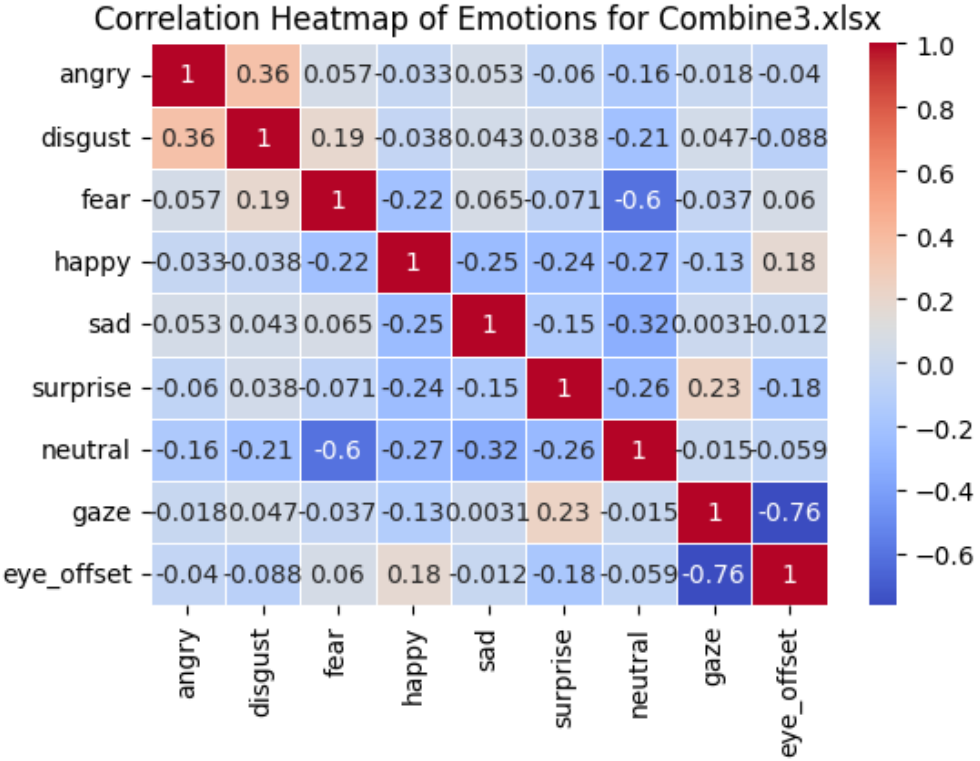

Combine9.xlsx
angry
disgust
fear
happy
sad
surprise
neutral
gaze
eye_offset
image_seq

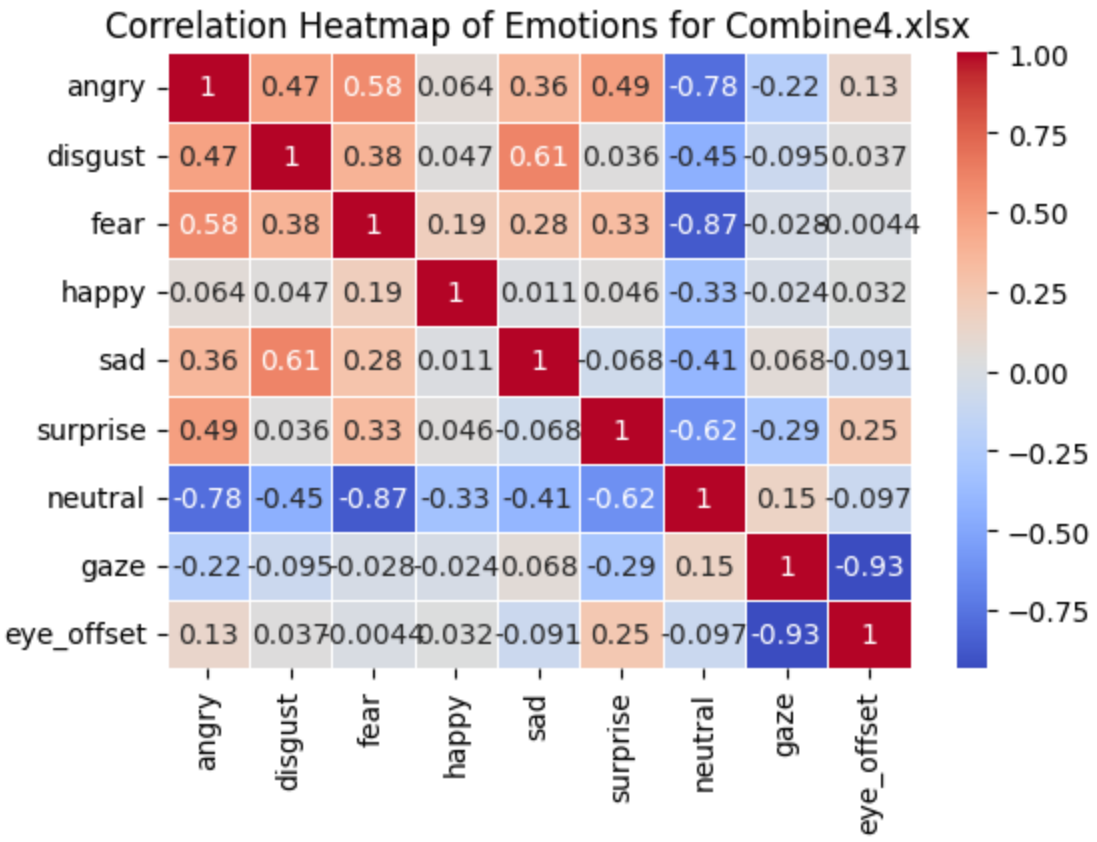

Combine10.xlsx
angry
disgust
fear
happy
sad
surprise
neutral
gaze
eye_offset
image_seq

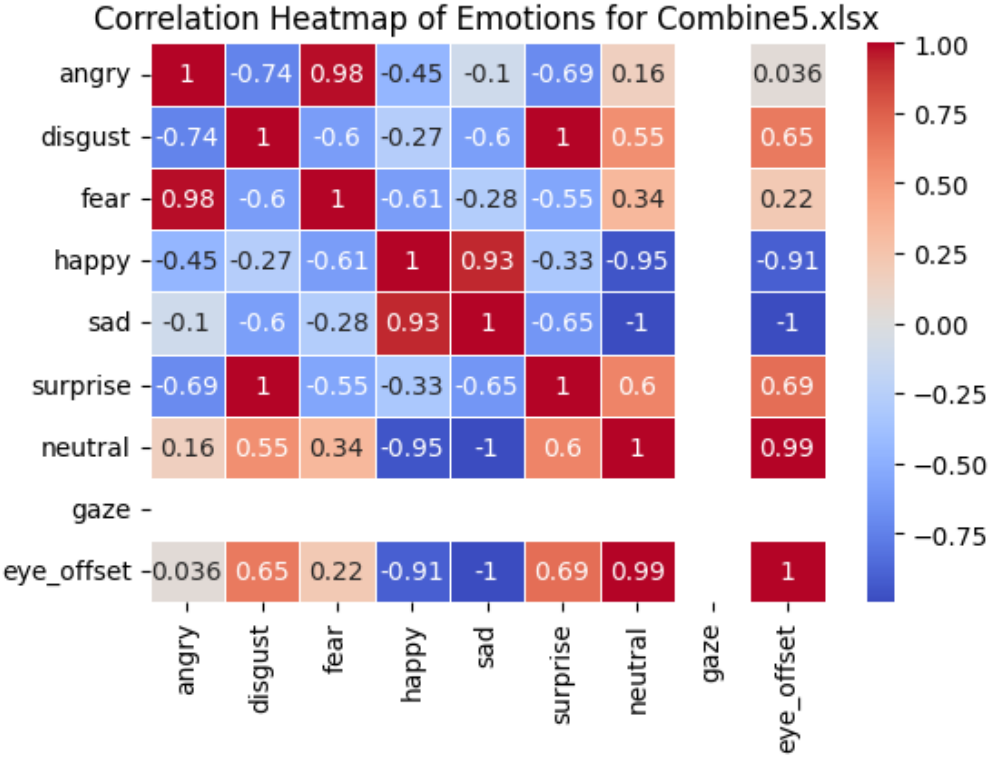
```
for file in filename:
    df = pd.read_excel(file)
    emotions = ['angry', 'disgust', 'fear', 'happy', 'sad', 'surprise',
'neutral','gaze','eye_offset']

    emotion_df = df[emotions]

    correlation_matrix = emotion_df.corr()

    # Creating a heatmap of the correlation matrix
    plt.figure(figsize=(10, 8))
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
linewidths=0.5)
    plt.title(f'Correlation Heatmap of Emotions for {file}')
    plt.show()
```

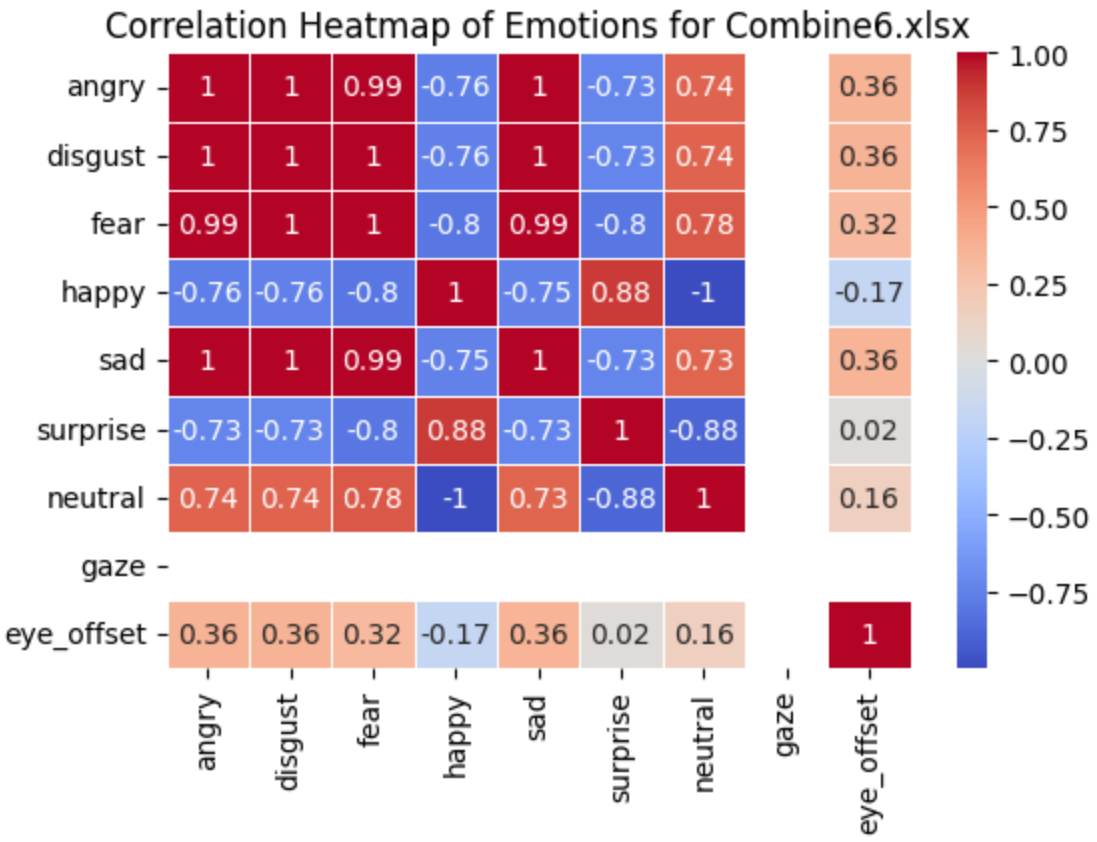


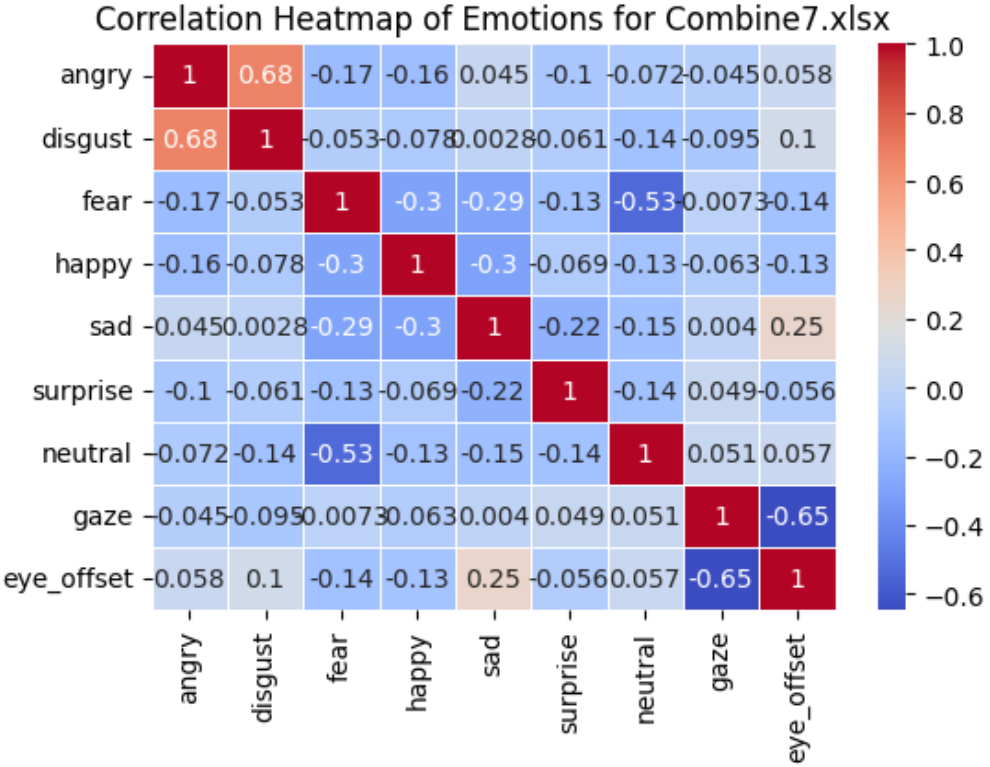

Correlation Heatmap of Emotions for Combine2.xlsx
angry 1 0.38 -0.023 -0.31 0.0049 0.014 -0.26 -0.11 -0.027
disgust 0.38 1 0.16 -0.15 0.0033 0.027 -0.17 0.073 -0.039
fear -0.023 0.16 1 -0.2 0.12 0.038 -0.22 -0.0013 0.083
happy -0.31 -0.15 -0.2 1 -0.31 -0.14 -0.54 -0.048 0.033
sad 0.0049 0.0033 0.12 -0.31 1 -0.086 -0.29 -0.029 0.12
surprise -0.014 0.027 0.038 -0.14 -0.086 1 -0.1 0.096 -0.014
neutral -0.26 -0.17 -0.22 -0.54 -0.29 -0.1 1 0.097 -0.043
gaze -0.11 0.073 0.0013 0.048 -0.029 0.096 0.097 1 -0.78
eye_offset -0.027 -0.039 0.083 -0.033 0.12 -0.014 -0.043 -0.78 1
angry disgust fear happy sad surprise neutral gaze eye_offset
1.0 0.8 0.6 0.4 0.2 0.0 −0.2 −0.4 −0.6

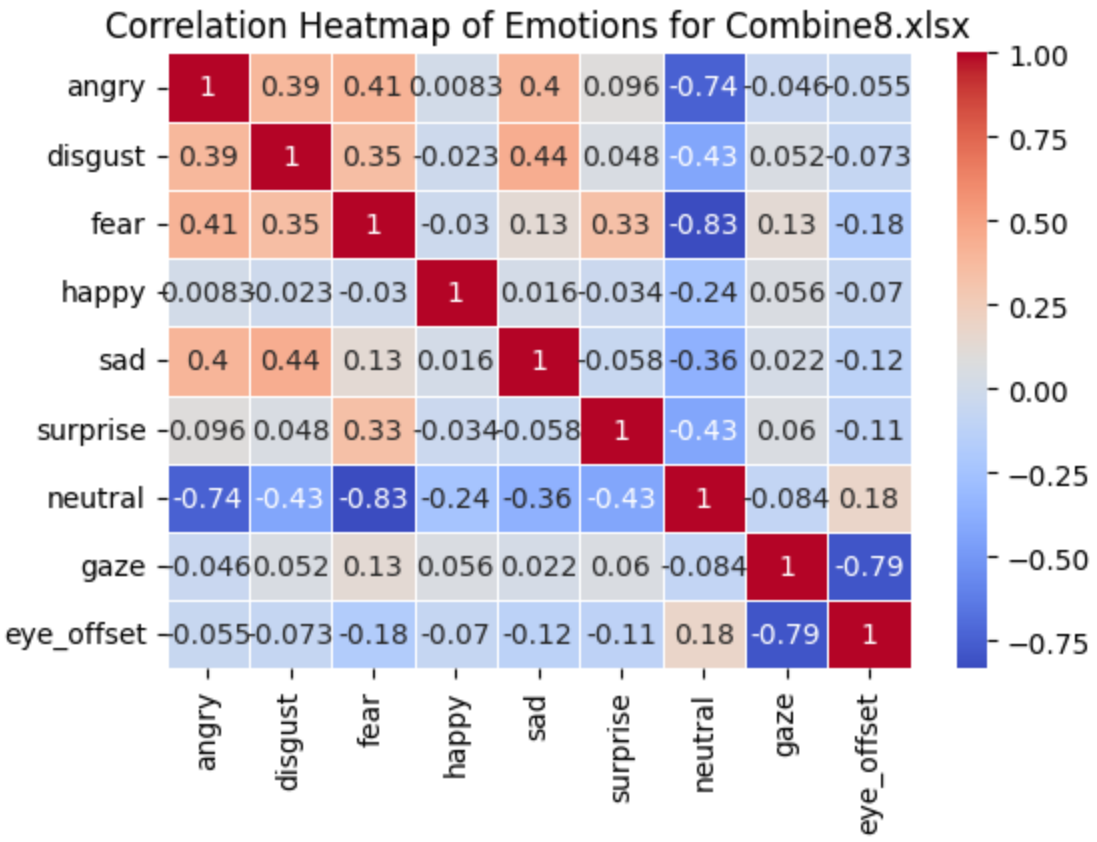

Correlation Heatmap of Emotions for Combine3.xlsx
angry 1 0.36 0.057 -0.033 0.053 -0.06 -0.16 -0.018 -0.04
disgust 0.36 1 0.19 -0.038 0.043 0.038 -0.21 0.047 -0.088
fear 0.057 0.19 1 -0.22 0.065 -0.071 -0.6 -0.037 0.06
happy -0.033 -0.038 -0.22 1 -0.25 -0.24 -0.27 -0.13 0.18
sad 0.053 0.043 0.065 -0.25 1 -0.15 -0.32 0.0031 -0.012
surprise -0.06 0.038 -0.071 -0.24 -0.15 1 -0.26 0.23 -0.18
neutral -0.16 -0.21 -0.6 -0.27 -0.32 -0.26 1 -0.015 0.059
gaze 0.018 0.047 -0.037 -0.13 0.0031 0.23 -0.015 1 -0.76
eye_offset -0.04 -0.088 0.06 0.18 -0.012 -0.18 0.059 -0.76 1
angry disgust fear happy sad surprise neutral gaze eye_offset
1.0 0.8 0.6 0.4 0.2 0.0 −0.2 −0.4 −0.6

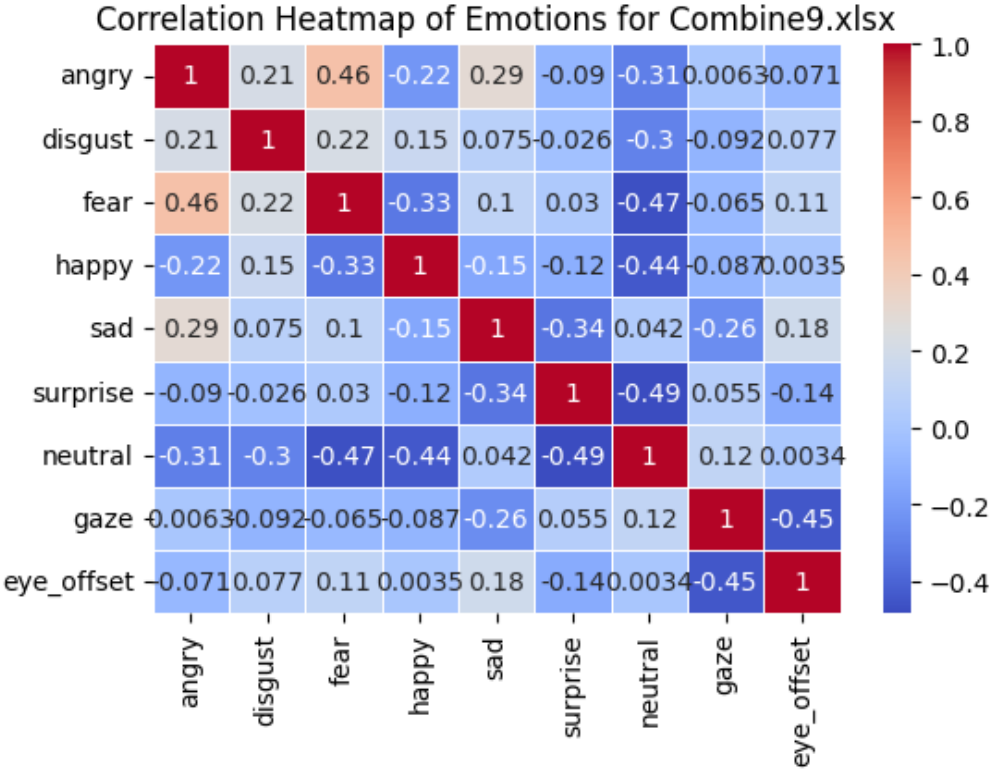

Correlation Heatmap of Emotions for Combine4.xlsx
angry 1 0.47 0.58 0.064 0.36 0.49 -0.78 -0.22 0.13
disgust 0.47 1 0.38 0.047 0.61 0.036 -0.45 -0.095 0.037
fear 0.58 0.38 1 0.19 0.28 0.33 -0.87 -0.028 0.0044
happy -0.064 0.047 0.19 1 0.011 0.046 -0.33 -0.024 0.032
sad 0.36 0.61 0.28 0.011 1 -0.068 -0.41 0.068 -0.091
surprise 0.49 0.036 0.33 0.046 -0.068 1 -0.62 -0.29 0.25
neutral -0.78 -0.45 -0.87 -0.33 -0.41 -0.62 1 0.15 -0.097
gaze -0.22 -0.095 -0.028 -0.024 0.068 -0.29 0.15 1 -0.93
eye_offset 0.13 0.037 -0.0044 0.032 -0.091 0.25 -0.097 -0.93 1
angry disgust fear happy sad surprise neutral gaze eye_offset
1.00 0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75

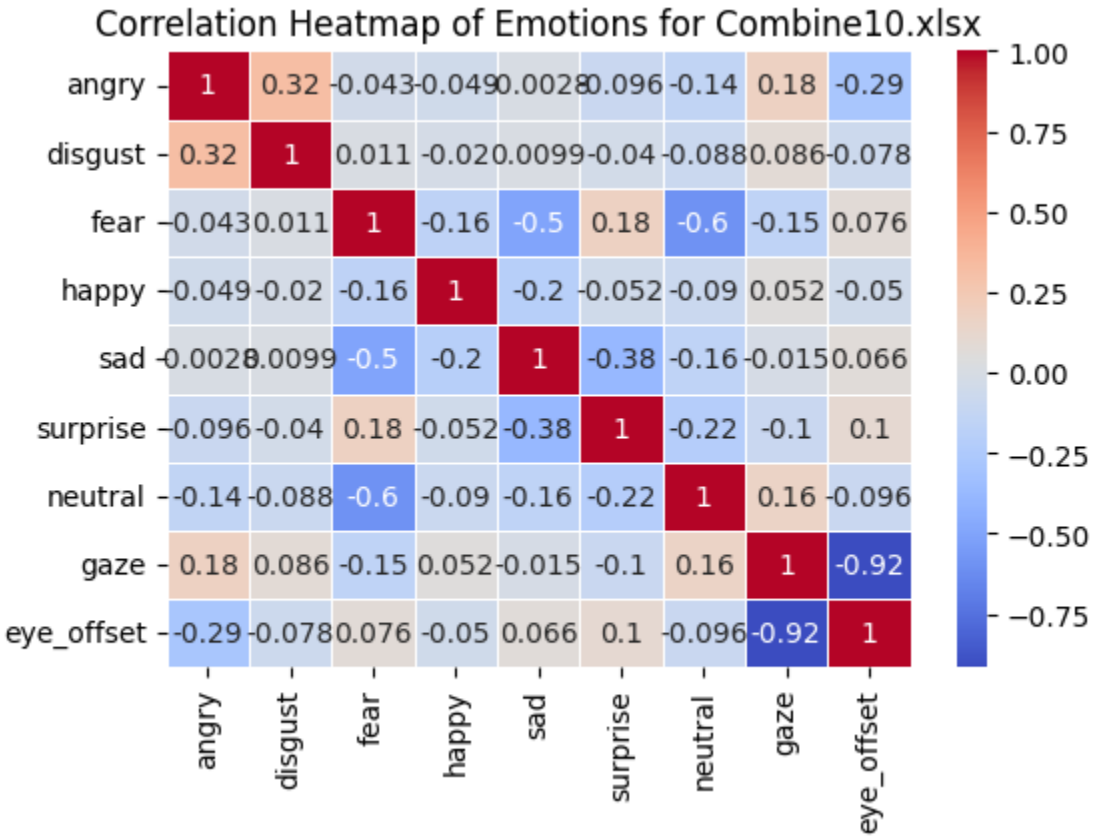

Correlation Heatmap of Emotions for Combine5.xlsx
angry 1 -0.74 0.98 -0.45 -0.1 -0.69 0.16 0.036
disgust -0.74 1 -0.6 -0.27 -0.6 1 0.55 0.65
fear 0.98 -0.6 1 -0.61 -0.28 -0.55 0.34 0.22
happy -0.45 -0.27 -0.61 1 0.93 -0.33 -0.95 -0.91
sad -0.1 -0.6 -0.28 0.93 1 -0.65 -1 -1
surprise -0.69 1 -0.55 -0.33 -0.65 1 0.6 0.69
neutral 0.16 0.55 0.34 -0.95 -1 0.6 1 0.99
gaze
eye_offset -0.036 0.65 0.22 -0.91 -1 0.69 0.99 1
angry disgust fear happy sad surprise neutral gaze eye_offset
1.00 0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75


Correlation Heatmap of Emotions for Combine6.xlsx
angry 1 1 0.99 -0.76 1 -0.73 0.74 0.36
disgust 1 1 1 -0.76 1 -0.73 0.74 0.36
fear 0.99 1 1 -0.8 0.99 -0.8 0.78 0.32
happy -0.76 -0.76 -0.8 1 -0.75 0.88 -1 -0.17
sad 1 1 0.99 -0.75 1 -0.73 0.73 0.36
surprise -0.73 -0.73 -0.8 0.88 -0.73 1 -0.88 0.02
neutral 0.74 0.74 0.78 -1 0.73 -0.88 1 0.16
gaze
eye_offset 0.36 0.36 0.32 -0.17 0.36 0.02 0.16 1
angry disgust fear happy sad surprise neutral gaze eye_offset
1.00 0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75


Correlation Heatmap of Emotions for Combine7.xlsx
angry 1 0.68 -0.17 -0.16 0.045 -0.1 -0.072 -0.045 0.058
disgust 0.68 1 -0.053 -0.078 0.0028 0.061 -0.14 -0.095 0.1
fear -0.17 -0.053 1 -0.3 -0.29 -0.13 -0.53 0.0073 -0.14
happy -0.16 -0.078 -0.3 1 -0.3 -0.069 -0.13 -0.063 -0.13
sad -0.045 0.0028 -0.29 -0.3 1 -0.22 -0.15 0.004 0.25
surprise -0.1 -0.061 -0.13 -0.069 -0.22 1 -0.14 0.049 -0.056
neutral -0.072 -0.14 -0.53 -0.13 -0.15 -0.14 1 0.051 0.057
gaze -0.045 -0.095 0.0073 0.063 0.004 0.049 0.051 1 -0.65
eye_offset -0.058 0.1 -0.14 -0.13 0.25 -0.056 0.057 -0.65 1
angry disgust fear happy sad surprise neutral gaze eye_offset
1.0 0.8 0.6 0.4 0.2 0.0 −0.2 −0.4 −0.6


Correlation Heatmap of Emotions for Combine8.xlsx
angry 1 0.39 0.41 0.0083 0.4 0.096 -0.74 -0.046 -0.055
disgust 0.39 1 0.35 -0.023 0.44 0.048 -0.43 0.052 -0.073
fear 0.41 0.35 1 -0.03 0.13 0.33 -0.83 0.13 -0.18
happy 0.0083 0.023 -0.03 1 0.016 -0.034 -0.24 0.056 -0.07
sad 0.4 0.44 0.13 0.016 1 -0.058 -0.36 0.022 -0.12
surprise -0.096 0.048 0.33 -0.034 -0.058 1 -0.43 0.06 -0.11
neutral -0.74 -0.43 -0.83 -0.24 -0.36 -0.43 1 -0.084 0.18
gaze -0.046 0.052 0.13 0.056 0.022 0.06 -0.084 1 -0.79
eye_offset -0.055 -0.073 -0.18 -0.07 -0.12 -0.11 0.18 -0.79 1
angry disgust fear happy sad surprise neutral gaze eye_offset
1.00 0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75

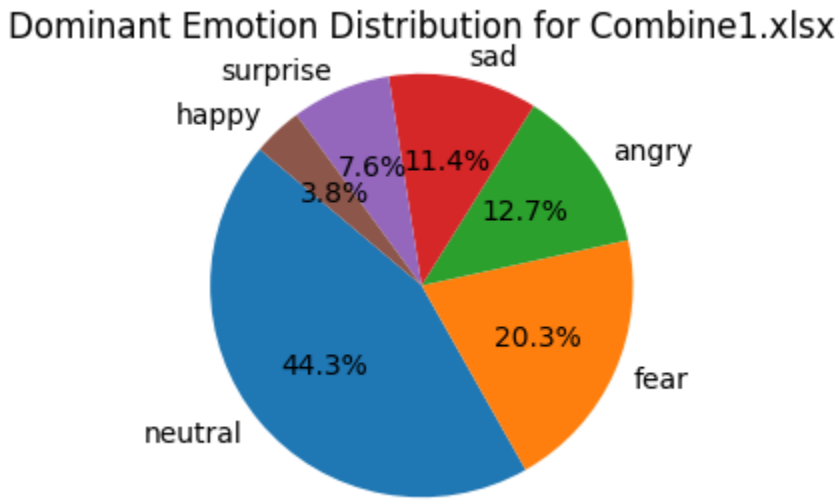

Correlation Heatmap of Emotions for Combine9.xlsx
angry 1 0.21 0.46 -0.22 0.29 -0.09 -0.31 0.0063 0.071
disgust 0.21 1 0.22 0.15 0.075 -0.026 -0.3 -0.092 0.077
fear 0.46 0.22 1 -0.33 0.1 0.03 -0.47 -0.065 0.11
happy -0.22 0.15 -0.33 1 -0.15 -0.12 -0.44 -0.087 0.0035
sad 0.29 0.075 0.1 -0.15 1 -0.34 0.042 -0.26 0.18
surprise -0.09 -0.026 0.03 -0.12 -0.34 1 -0.49 0.055 -0.14
neutral -0.31 -0.3 -0.47 -0.44 0.042 -0.49 1 0.12 0.0034
gaze 0.0063 0.092 -0.065 -0.087 -0.26 0.055 0.12 1 -0.45
eye_offset -0.071 0.077 0.11 0.0035 0.18 -0.14 0.0034 -0.45 1
angry disgust fear happy sad surprise neutral gaze eye_offset
1.0 0.8 0.6 0.4 0.2 0.0 −0.2 −0.4

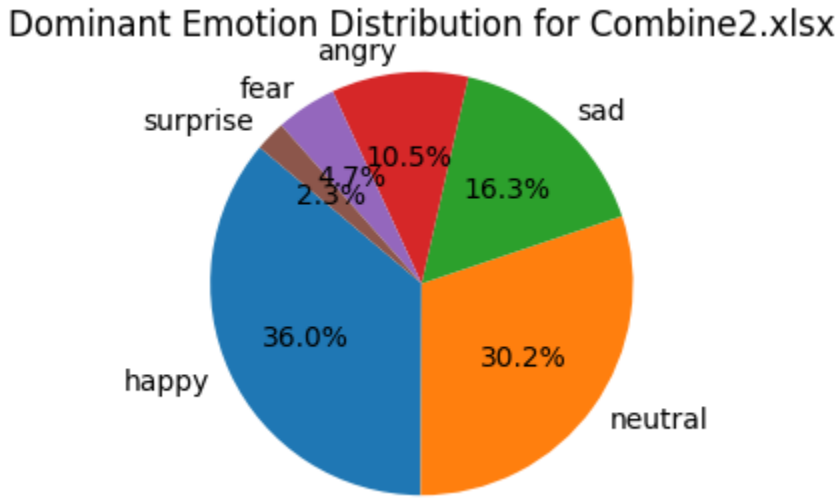


Plotting a pie-chart to demonstrate dominant emotion for each combined excel file:

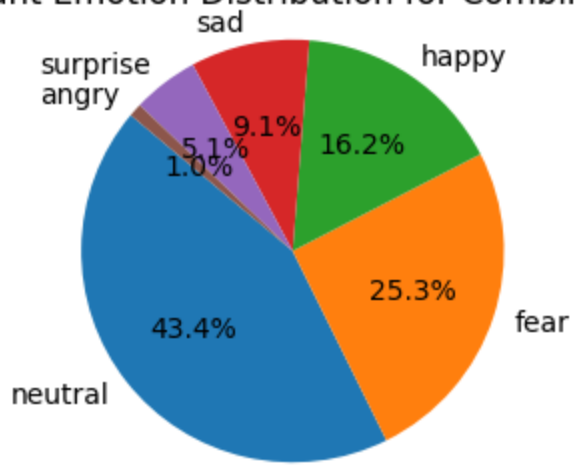
```
for file in filename:
    df = pd.read_excel(file)

    # Extracting the 'dominant_emotion' column
    dominant_emotions = df['dominant_emotion']

    # Calculating the frequency of each emotion
    emotion_counts = dominant_emotions.value_counts()

    # Creating a pie chart
    plt.figure(figsize=(3, 3))  # Adjust the figure size here
    plt.pie(emotion_counts, labels=emotion_counts.index,
autopct='%1.1f%%', startangle=140)
    plt.title(f'Dominant Emotion Distribution for {file}')
    plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a
circle.
```

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

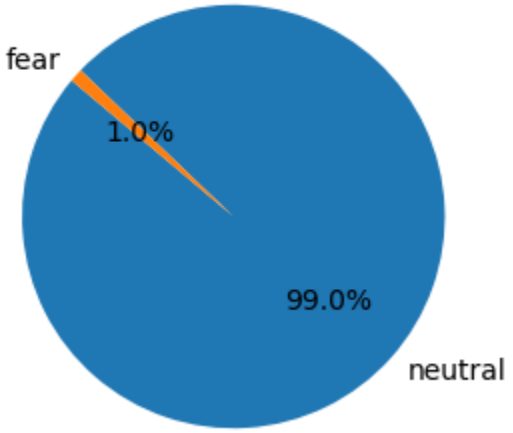




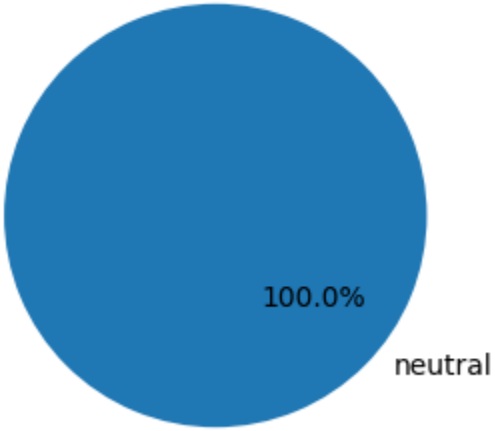
Dominant Emotion Distribution for Combine3.xlsx



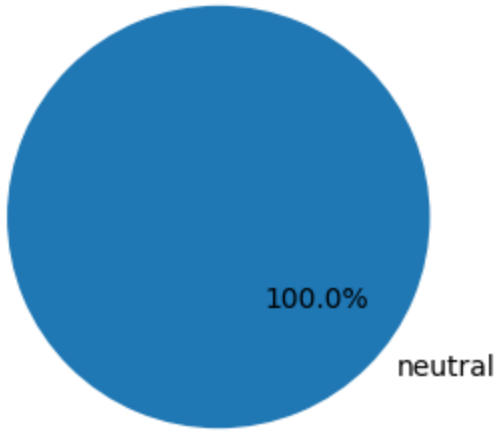
Dominant Emotion Distribution for Combine4.xlsx



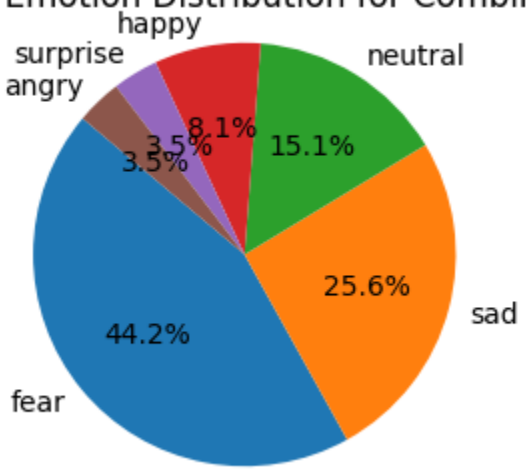
Dominant Emotion Distribution for Combine5.xlsx



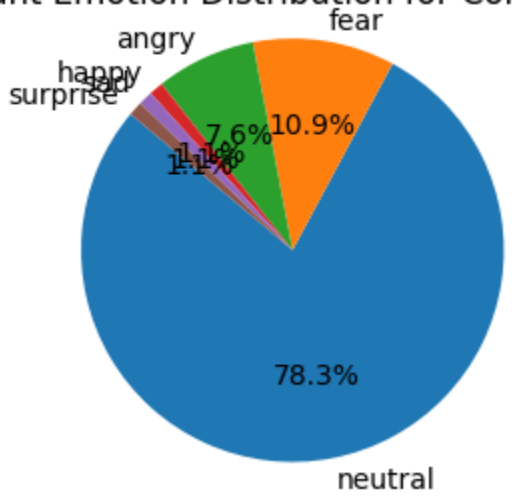
Dominant Emotion Distribution for Combine6.xlsx

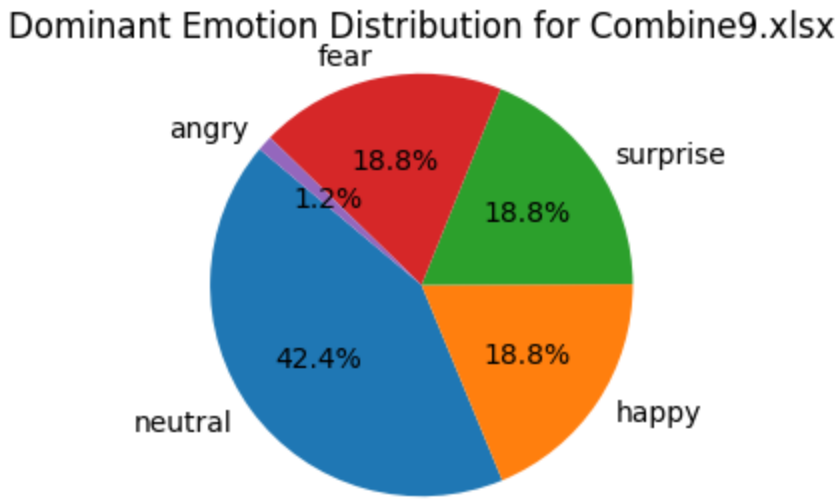


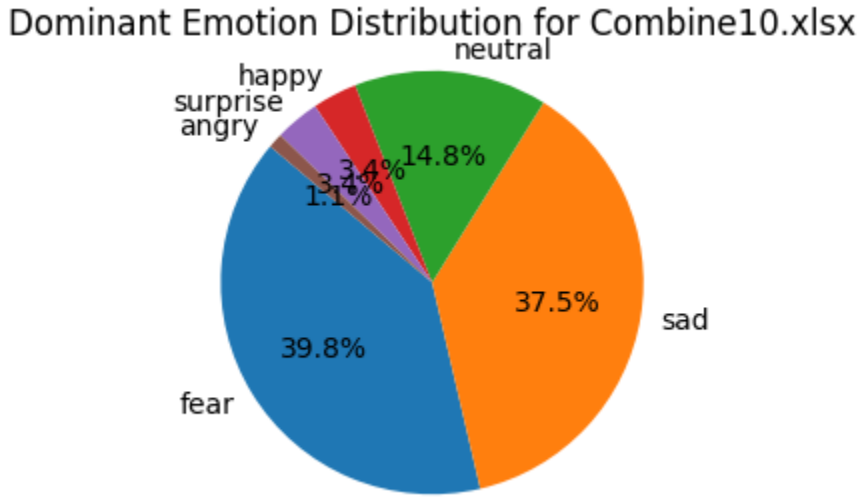
Dominant Emotion Distribution for Combine7.xlsx



Dominant Emotion Distribution for Combine8.xlsx



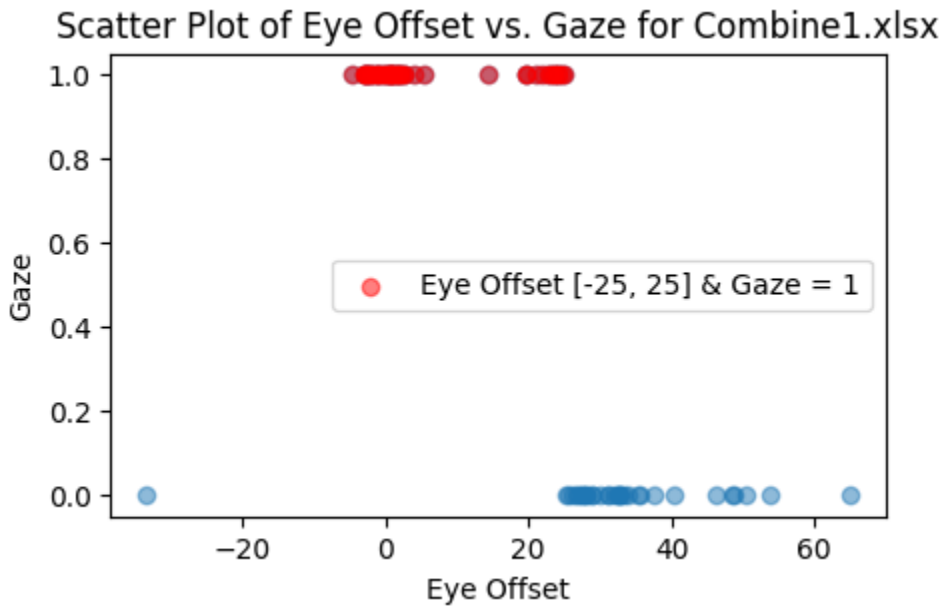


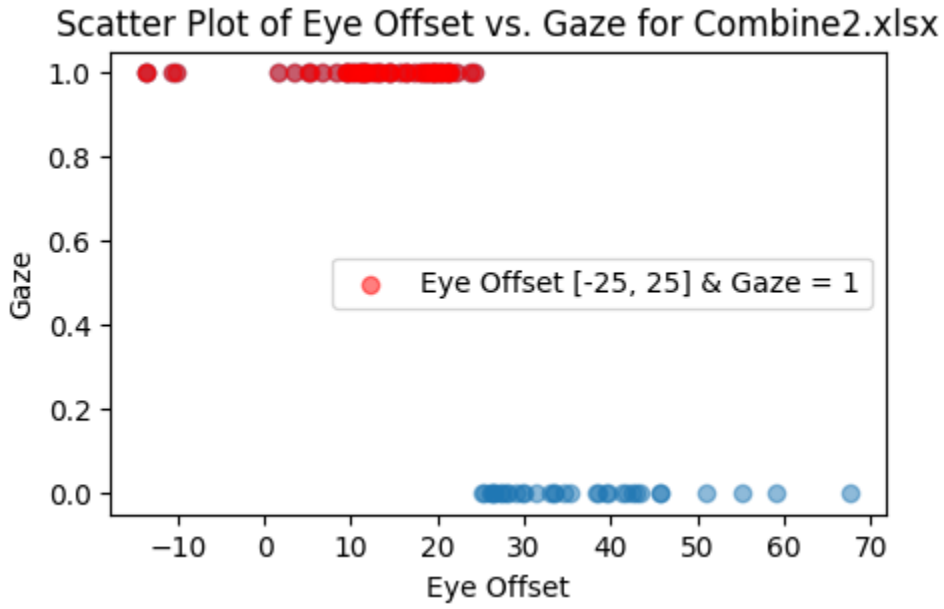


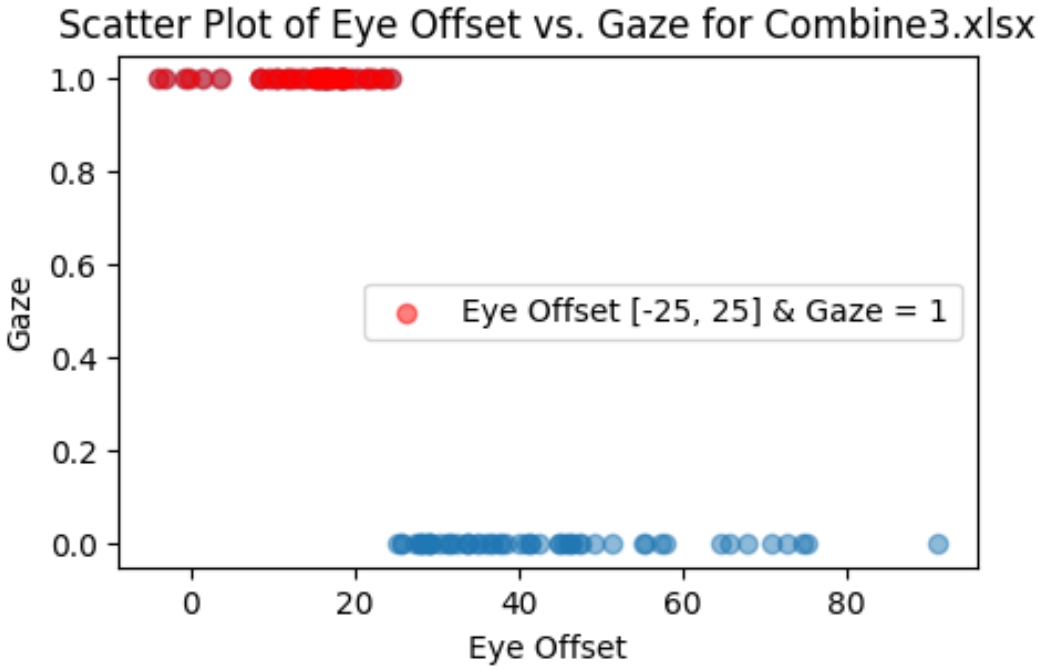
Plotting the scatter plot between gaze and eye_offset to see relation between the two:

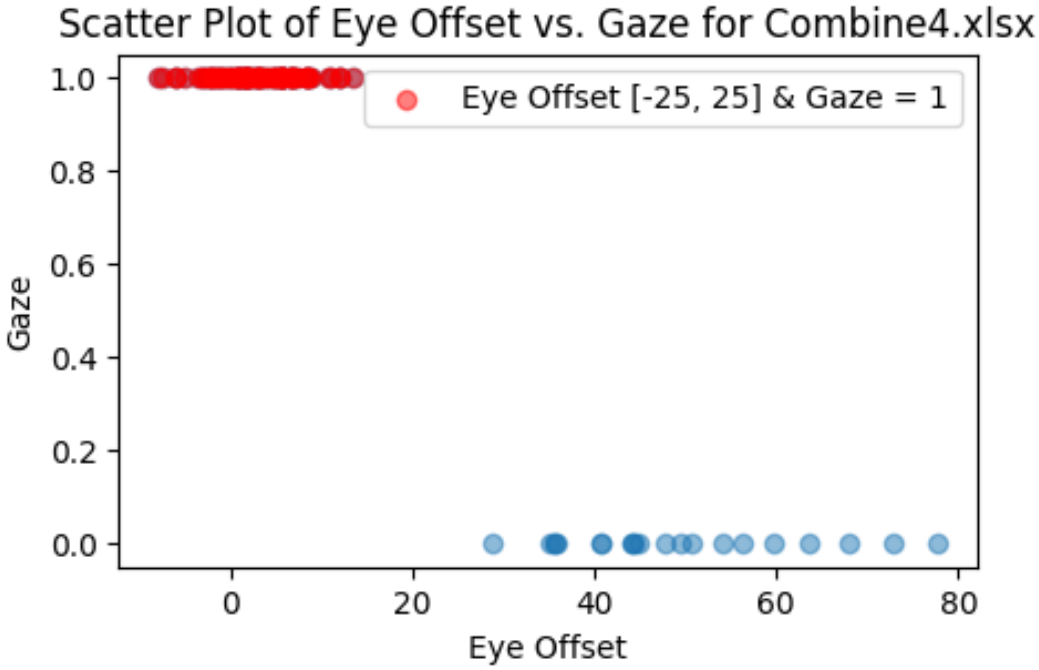
```
for file in filename:
    df = pd.read_excel(file)

    eye_offset = df['eye_offset']
    gaze = df['gaze']

    # Creating a scatter plot
    plt.figure(figsize=(5, 3))
    plt.scatter(eye_offset, gaze, alpha=0.5)
    plt.title(f'Scatter Plot of Eye Offset vs. Gaze for {file}')
    plt.xlabel('Eye Offset')
    plt.ylabel('Gaze')
```

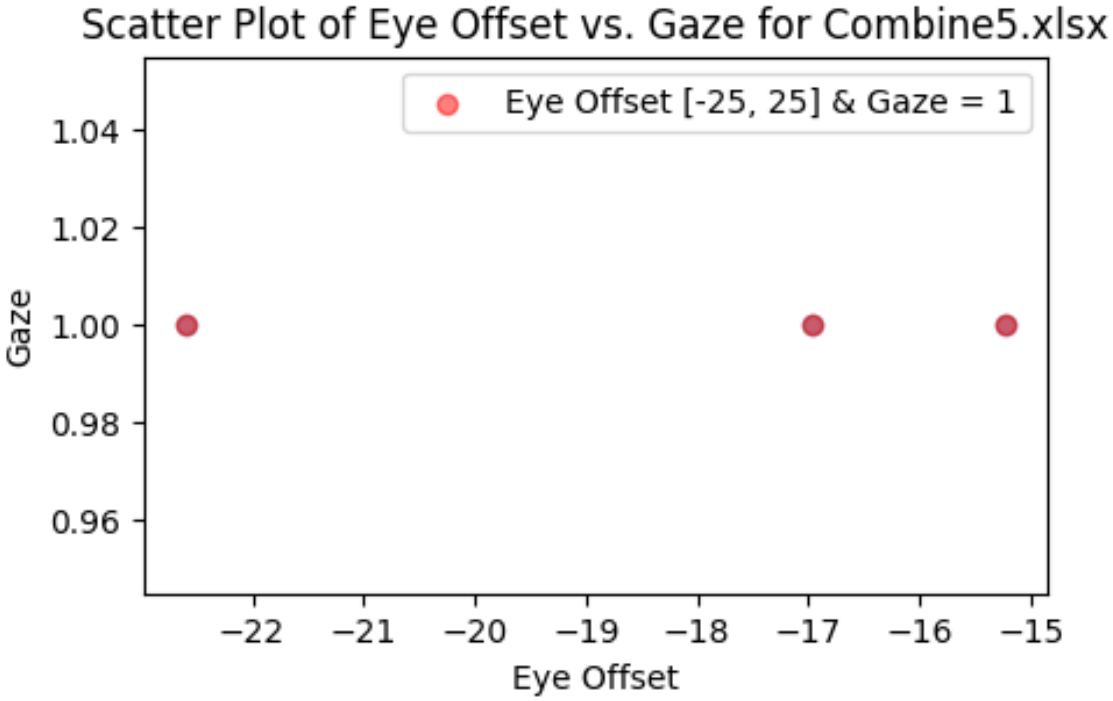
```
    # Highlighting points where eye_offset is between -25 and 25 and gaze
is 1
    mask = (eye_offset >= -25) & (eye_offset <= 25) & (gaze == 1)
    plt.scatter(eye_offset[mask], gaze[mask], color='red', alpha=0.5,
label='Eye Offset [-25, 25] & Gaze = 1')
    plt.legend()

    plt.show()
```

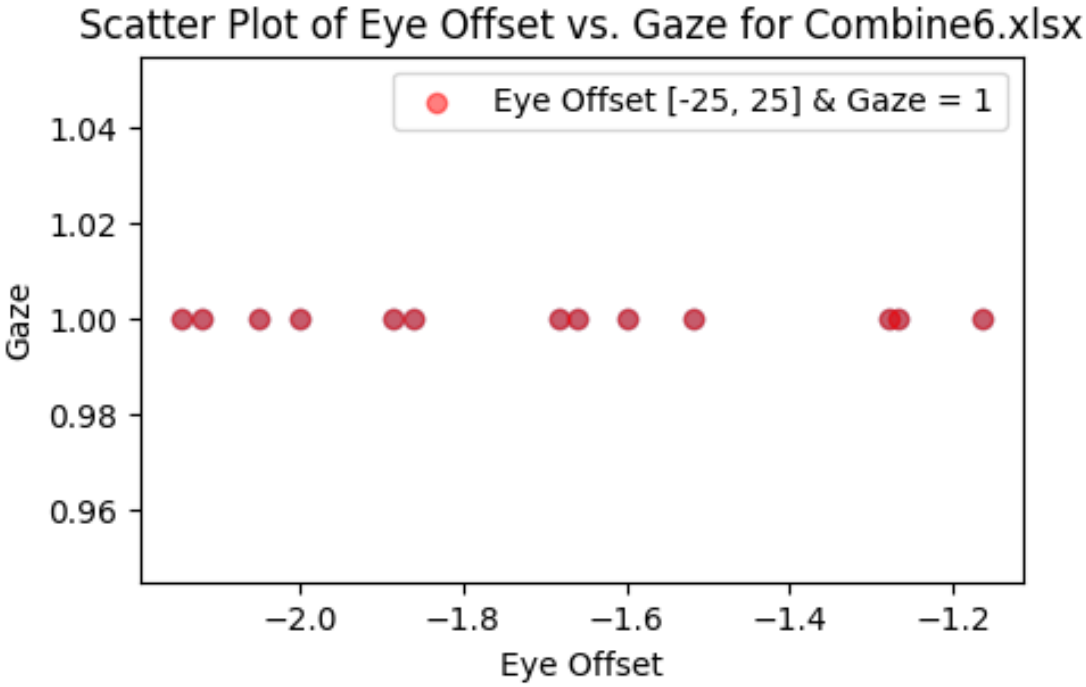


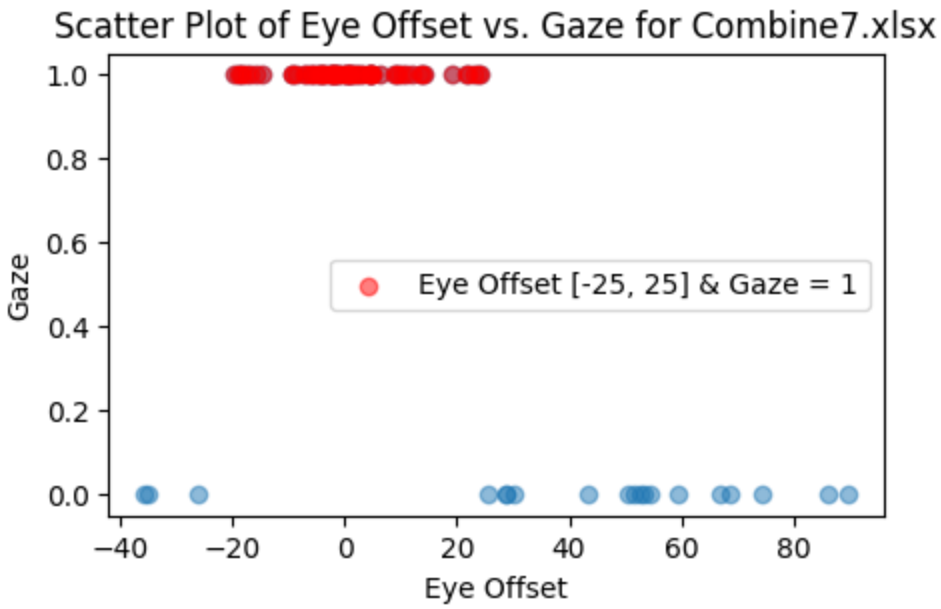


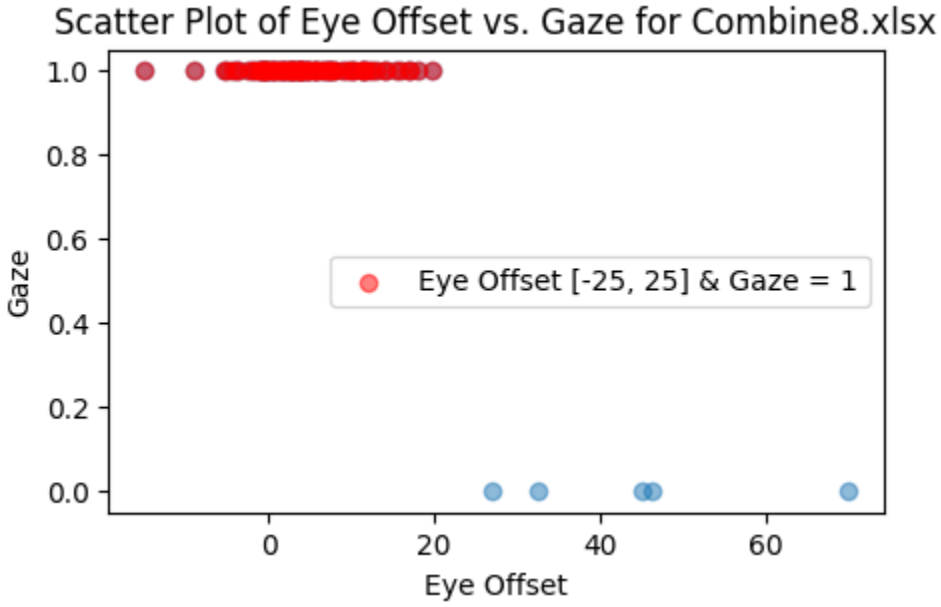

Scatter Plot of Eye Offset vs. Gaze for Combine3.xlsx
Gaze
Eye Offset
Eye Offset [-25, 25] & Gaze = 1
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80

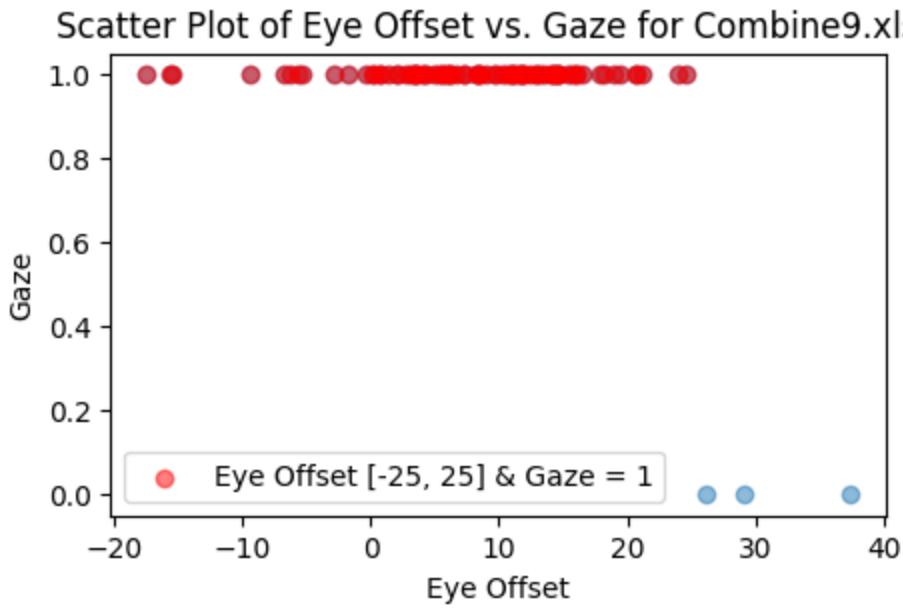

Scatter Plot of Eye Offset vs. Gaze for Combine4.xlsx
Gaze
Eye Offset
Eye Offset [-25, 25] & Gaze = 1
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80

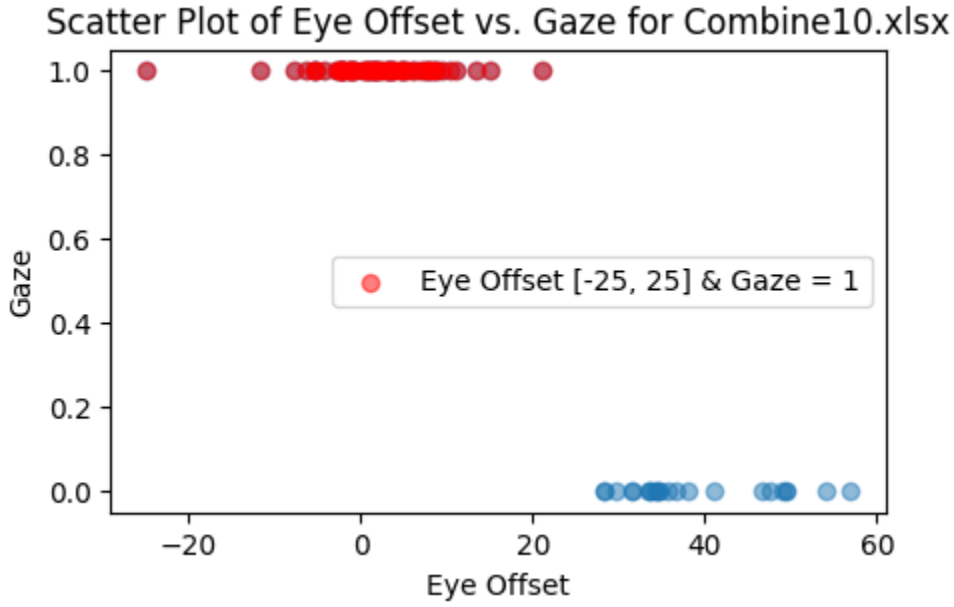

Scatter Plot of Eye Offset vs. Gaze for Combine5.xlsx
Eye Offset [-25, 25] & Gaze = 1
1.04
1.02
1.00
0.98
0.96
Gaze
−22
−21
−20
−19
−18
−17
−16
−15
Eye Offset


Scatter Plot of Eye Offset vs. Gaze for Combine6.xlsx
Eye Offset [-25, 25] & Gaze = 1
1.04
1.02
1.00
0.98
0.96
Gaze
−2.0
−1.8
−1.6
−1.4
−1.2
Eye Offset


Scatter Plot of Eye Offset vs. Gaze for Combine7.xlsx
Gaze
Eye Offset
Eye Offset [-25, 25] & Gaze = 1
1.0
0.8
0.6
0.4
0.2
0.0
−40
−20
0
20
40
60
80


Scatter Plot of Eye Offset vs. Gaze for Combine8.xlsx
Gaze
Eye Offset
Eye Offset [-25, 25] & Gaze = 1
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60





We can clearly see that the value of gaze is 1 when eye_offset value is between -25 and 25 and 0 when it is outside this range.

Now let us get an arbitrary count value that when the person was shifting his eyes away from camera and again towards the camera his confidence score is calculated. More the value of count, the more confident the person was when he was speaking.

Value of count is directly proportional to the confidence of the person. If the eyes of the person is constantly moving away from the camera it indicates that he is not confident about what he is speaking.

```
for file in filename:
    df = pd.read_excel(file)

    count = 0
    gaze = list(df['gaze'])

    for i in range(1, len(gaze)):
        if (gaze[i] == 0 and gaze[i-1] == 1) or (gaze[i] == 1 and
gaze[i-1] == 0):
            count = count - 1  # Count transitions from 0 to 1 or from 1
to 0
        else:
          count += 1

    print(f'The value of count for {file} is', count)
```

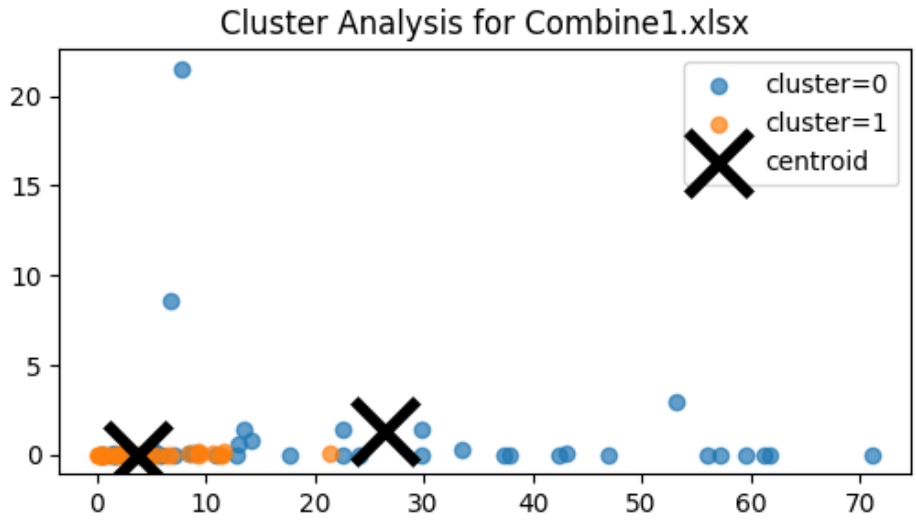
```
The value of count for Combine1.xlsx is 2
The value of count for Combine2.xlsx is 5
The value of count for Combine3.xlsx is 18
The value of count for Combine4.xlsx is 25
The value of count for Combine5.xlsx is 2
The value of count for Combine6.xlsx is 12
The value of count for Combine7.xlsx is 29
The value of count for Combine8.xlsx is 71
The value of count for Combine9.xlsx is 72
The value of count for Combine10.xlsx is 11
```

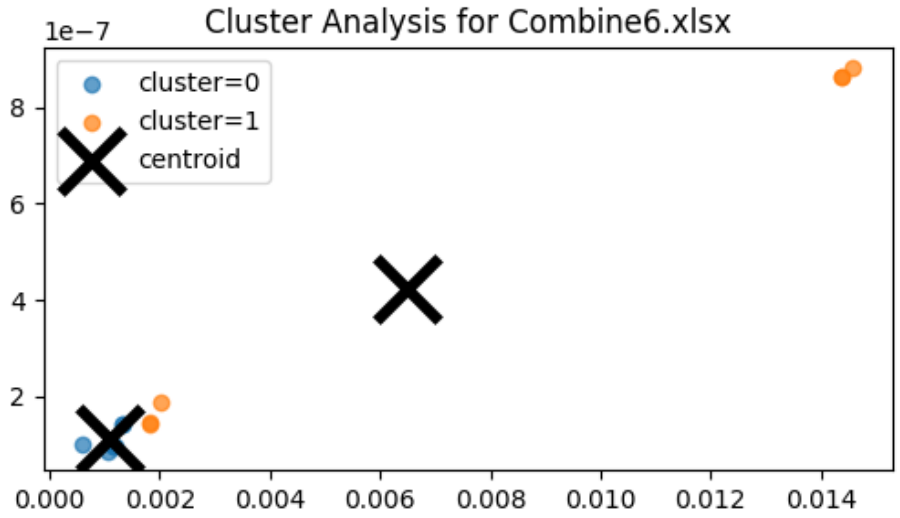
As there is no target variable, so rather than evaluating each person after its interview, we can evaluate him/her in every frame and then form a cluster based on that image sequence dataset.

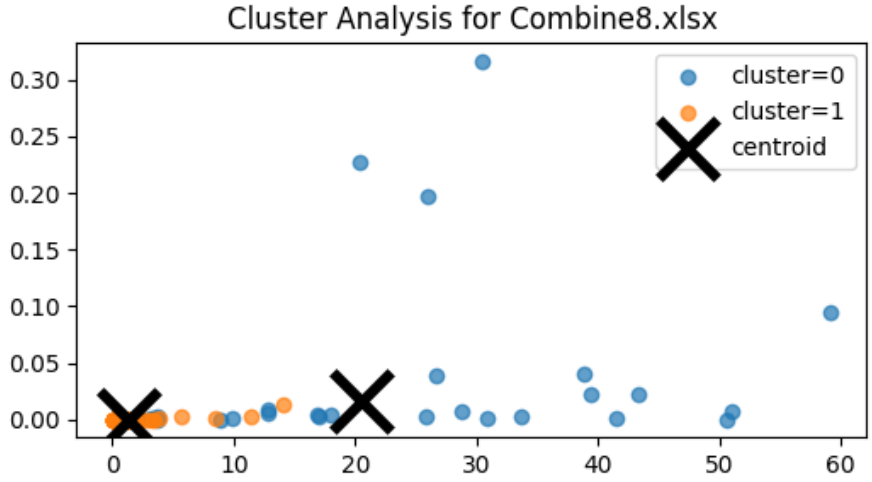
We can use **Self Organising Map (SOM)** for this task.

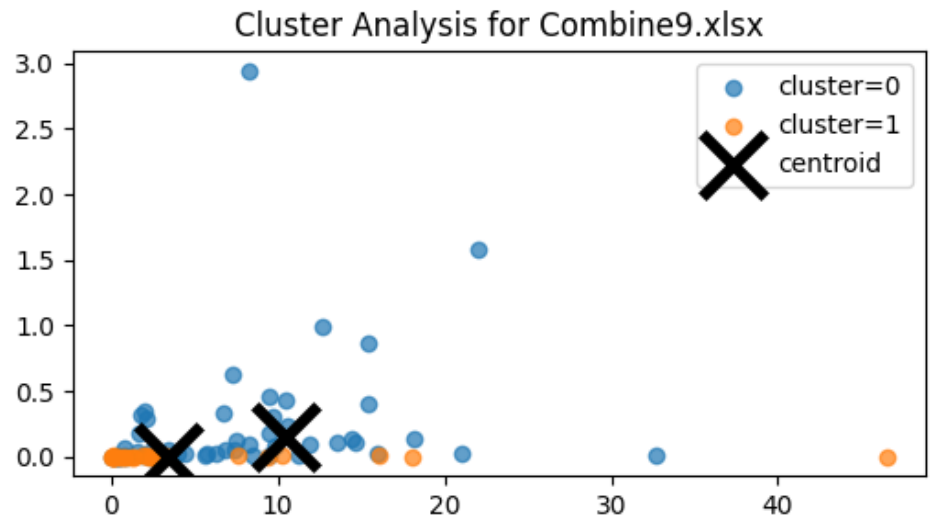
```
from minisom import MiniSom
import numpy as np
# Loop through each file
for file in filename:
    df = pd.read_excel(file)
```

```
    emotions = ['angry', 'disgust', 'fear', 'happy', 'sad', 'surprise',
'neutral']
    new_df = df[emotions].to_numpy()

    # Initialization and training
    som_shape = (1, 2)
    som = MiniSom(som_shape[0], som_shape[1], new_df.shape[1], sigma=.5,
learning_rate=.5,
                  neighborhood_function='gaussian', random_seed=10)

    som.train_batch(new_df, 500, verbose=True)

    # Each neuron represents a cluster
    winner_coordinates = np.array([som.winner(x) for x in new_df]).T
    # With np.ravel_multi_index, we convert the bidimensional
    # coordinates to a monodimensional index
    cluster_index = np.ravel_multi_index(winner_coordinates, som_shape)

    # Plotting the clusters using the first 2 dimensions of the data
    for c in np.unique(cluster_index):
        plt.scatter(new_df[cluster_index == c, 0],
                    new_df[cluster_index == c, 1], label='cluster=' +
str(c), alpha=.7)

    # Plotting centroids
    for centroid in som.get_weights():
        plt.scatter(centroid[:, 0], centroid[:, 1], marker='x',
                    s=10, linewidths=35, color='k', label='centroid')
    plt.legend()
    plt.title(f'Cluster Analysis for {file}')
    plt.show()
```


Cluster Analysis for Combine1.xlsx
cluster=0
cluster=1
centroid
20
15
10
5
0
0
10
20
30
40
50
60
70


Cluster Analysis for Combine6.xlsx
1e−7
cluster=0
cluster=1
centroid
8
6
4
2
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014


Cluster Analysis for Combine8.xlsx
cluster=0
cluster=1
centroid
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
10
20
30
40
50
60

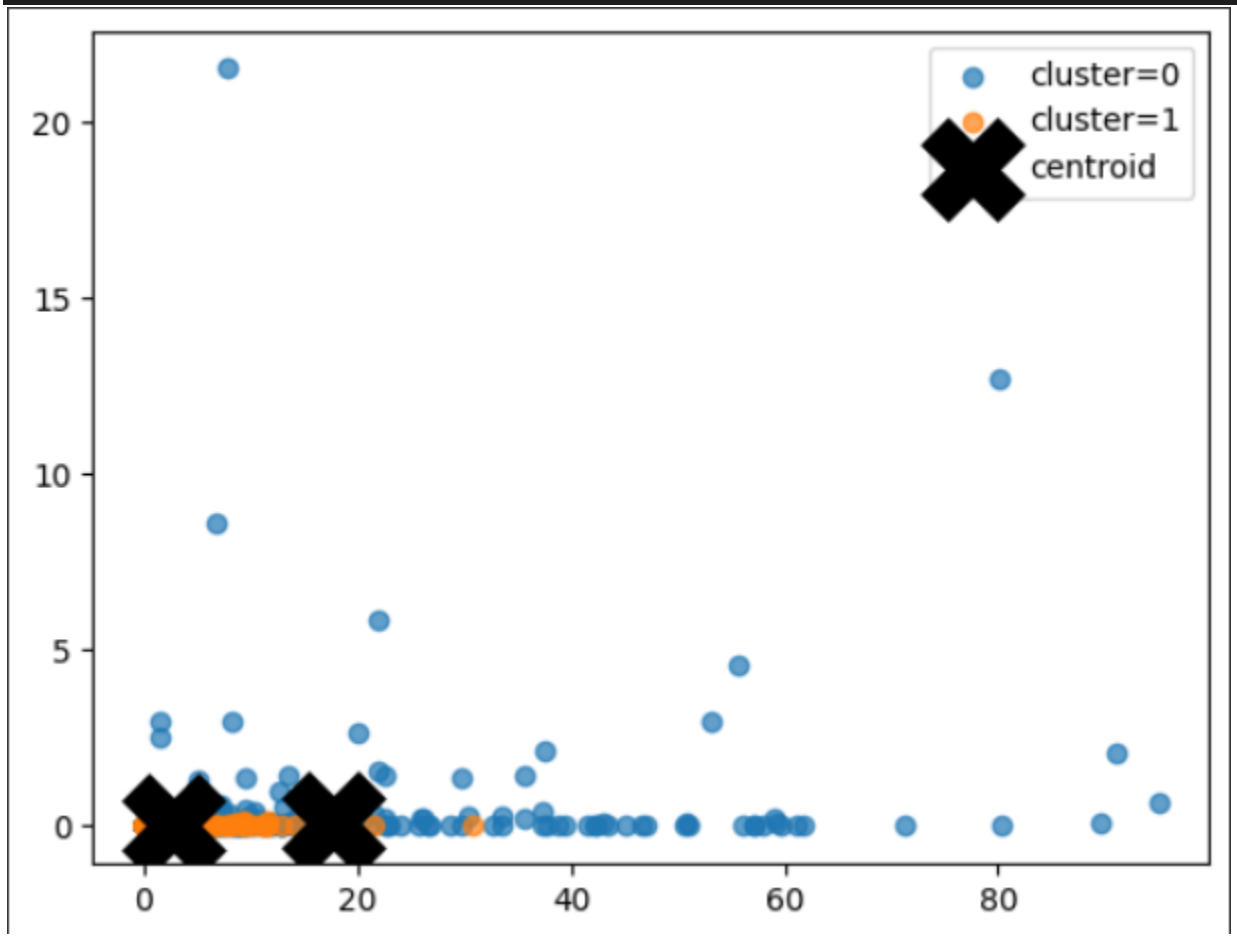


We can clearly see that we can separate every dataset into two clusters depending on their emotion score.

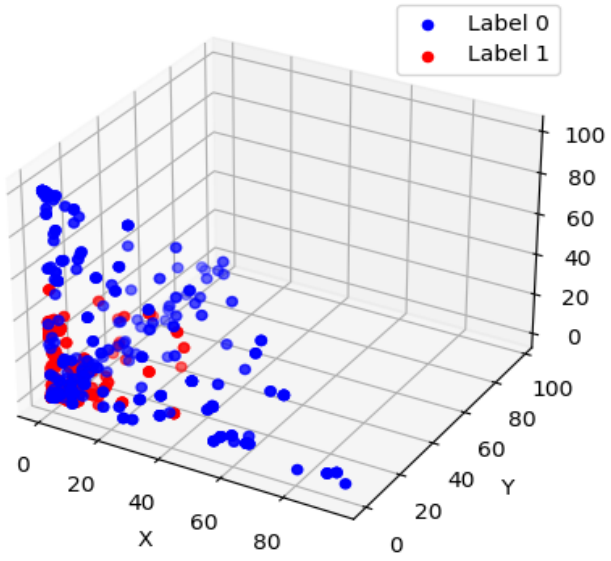
To test, we can train the SOM model on whole data of 10 people's emotion's score:

```
# Combining every person's data to train SOM on whole data
data_frames = [pd.read_excel(file) for file in filename]
combined_data = pd.concat(data_frames, ignore_index=True)
combined_data.to_excel("combined_data.xlsx", index=False)
combined_data = pd.read_excel("combined_data.xlsx")
# Training SOM on combined_data
# Select the columns for SOM clustering
emotions = ['angry', 'disgust', 'fear', 'happy', 'sad', 'surprise',
'neutral']
new_df = combined_data[emotions].to_numpy()

# Initialization and training
som_shape = (1, 2)
som = MiniSom(som_shape[0], som_shape[1], new_df.shape[1], sigma=.5,
learning_rate=.5,
              neighborhood_function='gaussian', random_seed=10)

som.train_batch(new_df, 500, verbose=True)

# each neuron represents a cluster
winner_coordinates = np.array([som.winner(x) for x in new_df]).T
```

```
# with np.ravel_multi_index we convert the bidimensional
# coordinates to a monodimensional index
cluster_index = np.ravel_multi_index(winner_coordinates, som_shape)

import matplotlib.pyplot as plt
%matplotlib inline

# plotting the clusters using the first 2 dimentions of the data
for c in np.unique(cluster_index):
    plt.scatter(new_df[cluster_index == c, 0],
                new_df[cluster_index == c, 1], label='cluster='+str(c),
alpha=.7)

# plotting centroids
for centroid in som.get_weights():
    plt.scatter(centroid[:, 0], centroid[:, 1], marker='x',
                s=80, linewidths=35, color='k', label='centroid')
plt.legend()
```



```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D  # Import the 3D plotting toolkit

# Sample 3D coordinates and color labels (replace with your data)
coordinates = new_df[:, [0, 4, 3]]  # 100 random 3D points
colors = cluster_index  # Random color labels (0 or 1)

# Create a 3D scatter plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Separate the points based on color labels and plot them
ax.scatter(coordinates[colors == 0, 0], coordinates[colors == 0, 1],
coordinates[colors == 0, 2], c='b', label='Label 0')
ax.scatter(coordinates[colors == 1, 0], coordinates[colors == 1, 1],
coordinates[colors == 1, 2], c='r', label='Label 1')

# Set labels and legend
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.legend()

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



```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Creating random features and labels for binary classification
X = new_df
y = cluster_index

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Build a simple feedforward neural network
model = keras.Sequential([
    keras.layers.Dense(32, activation='relu', input_shape=(7,)),
    keras.layers.Dense(16, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')  # Output layer for binary
classification
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Train the model
epochs = 20
batch_size = 32

model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size,
verbose=2)

# Evaluate the model on the test set
y_pred = model.predict(X_test)
y_pred_binary = (y_pred > 0.5).astype(int)

accuracy = accuracy_score(y_test, y_pred_binary)
print(f'Accuracy on the test set: {accuracy:.2f}')
```

```
Epoch 1/20
19/19 - 1s - loss: 6.9058 - accuracy: 0.4391 - 1s/epoch - 62ms/step
Epoch 2/20
19/19 - 0s - loss: 1.3024 - accuracy: 0.6621 - 31ms/epoch - 2ms/step
Epoch 3/20
19/19 - 0s - loss: 0.1241 - accuracy: 0.9451 - 35ms/epoch - 2ms/step
Epoch 4/20
19/19 - 0s - loss: 0.0741 - accuracy: 0.9708 - 33ms/epoch - 2ms/step
Epoch 5/20
19/19 - 0s - loss: 0.0543 - accuracy: 0.9828 - 46ms/epoch - 2ms/step
Epoch 6/20
19/19 - 0s - loss: 0.0450 - accuracy: 0.9863 - 55ms/epoch - 3ms/step
Epoch 7/20
19/19 - 0s - loss: 0.0392 - accuracy: 0.9914 - 38ms/epoch - 2ms/step
Epoch 8/20
19/19 - 0s - loss: 0.0351 - accuracy: 0.9914 - 41ms/epoch - 2ms/step
Epoch 9/20
19/19 - 0s - loss: 0.0314 - accuracy: 0.9914 - 37ms/epoch - 2ms/step
Epoch 10/20
19/19 - 0s - loss: 0.0288 - accuracy: 0.9931 - 36ms/epoch - 2ms/step
Epoch 11/20
19/19 - 0s - loss: 0.0270 - accuracy: 0.9949 - 33ms/epoch - 2ms/step
Epoch 12/20
19/19 - 0s - loss: 0.0252 - accuracy: 0.9949 - 33ms/epoch - 2ms/step
Epoch 13/20
19/19 - 0s - loss: 0.0233 - accuracy: 0.9949 - 35ms/epoch - 2ms/step
Epoch 14/20
19/19 - 0s - loss: 0.0218 - accuracy: 0.9949 - 34ms/epoch - 2ms/step
Epoch 15/20
19/19 - 0s - loss: 0.0205 - accuracy: 0.9949 - 39ms/epoch - 2ms/step
Epoch 16/20
19/19 - 0s - loss: 0.0194 - accuracy: 0.9983 - 34ms/epoch - 2ms/step
Epoch 17/20
19/19 - 0s - loss: 0.0187 - accuracy: 0.9983 - 35ms/epoch - 2ms/step
Epoch 18/20
19/19 - 0s - loss: 0.0174 - accuracy: 0.9966 - 35ms/epoch - 2ms/step
```

```
Epoch 19/20
19/19 - 0s - loss: 0.0162 - accuracy: 0.9983 - 32ms/epoch - 2ms/step
Epoch 20/20
19/19 - 0s - loss: 0.0149 - accuracy: 0.9983 - 34ms/epoch - 2ms/step
5/5 [==============================] - 0s 2ms/step
Accuracy on the test set: 1.00
time: 2.28 s (started: 2023-09-28 15:06:02 +00:00)
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

# Generate a synthetic dataset with two classes (replace with your own
data)
np.random.seed(42)

# Create a t-SNE model
tsne = TSNE(n_components=2, perplexity=30, random_state=42)

# Fit and transform the data
X_tsne = tsne.fit_transform(X)

# Plot the t-SNE visualization
plt.figure(figsize=(8, 6))
plt.scatter(X_tsne[y == 0, 0], X_tsne[y == 0, 1], label='Class 0',
c='blue')
plt.scatter(X_tsne[y == 1, 0], X_tsne[y == 1, 1], label='Class 1',
c='red')
plt.title('t-SNE Visualization of Two Classes')
plt.legend()
plt.show()
```

- **Transcript data:**

  Loading library to connect google drive to colab

```
from google.colab import drive
drive.mount('/content/drive')
%cd /content/drive/MyDrive/imby_project/Transcript_score
```

  Installing autotime

```
!pip install ipython-autotime
%load_ext autotime
```

  Importing libraries

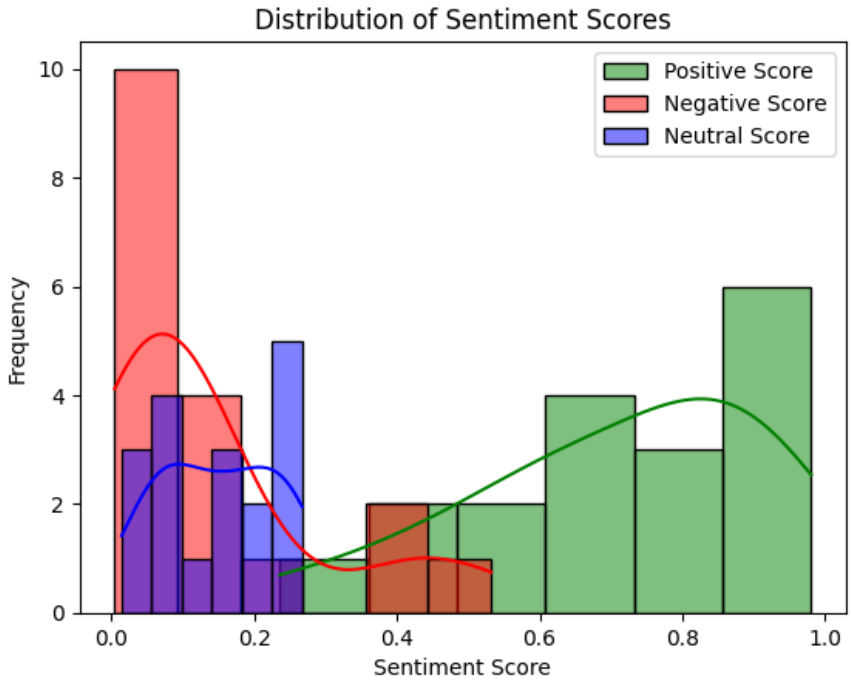
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

from sklearn.linear_model import LinearRegression

from sklearn.metrics import confusion_matrix, accuracy_score,
roc_auc_score, roc_curve
from sklearn import preprocessing
```

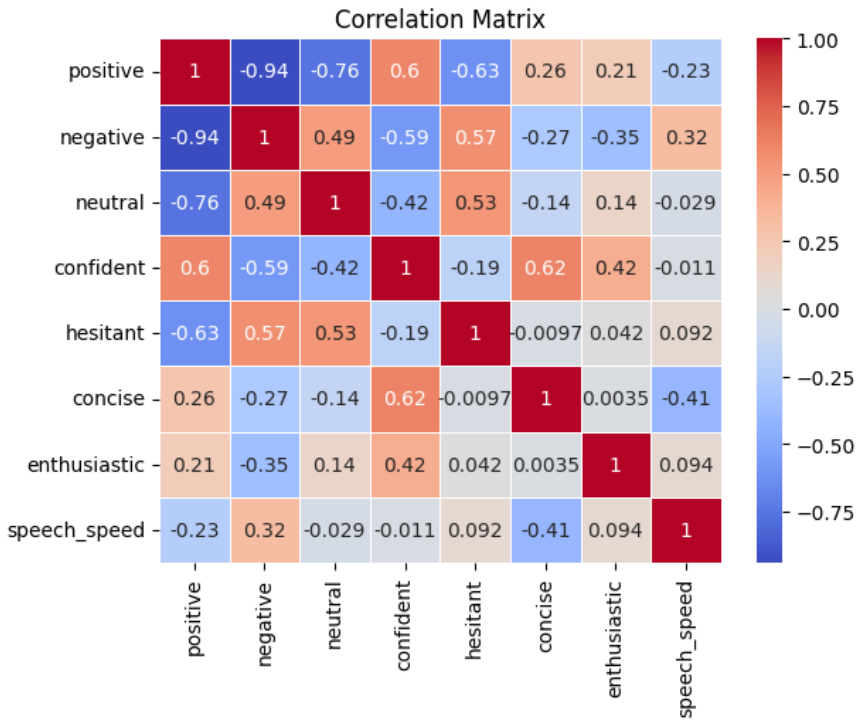
Loading dataset and performing basic EDA:

```
def EDA(df):

  # Dropping irrelevant columns
  df =
df.drop(columns=['temperature','seek','avg_logprob','compression_rat
io','no_speech_prob'])

  # Basic statistics
  summary_stats = df.describe()
  summary_stats

  # Distribution of sentiment scores
  sns.histplot(df['positive'], kde=True, label='Positive Score',
color='green')
  sns.histplot(df['negative'], kde=True, label='Negative Score',
color='red')
  sns.histplot(df['neutral'], kde=True, label='Neutral Score',
color='blue')
  plt.xlabel('Sentiment Score')
  plt.ylabel('Frequency')
  plt.legend()
  plt.title('Distribution of Sentiment Scores')
  plt.show()
```

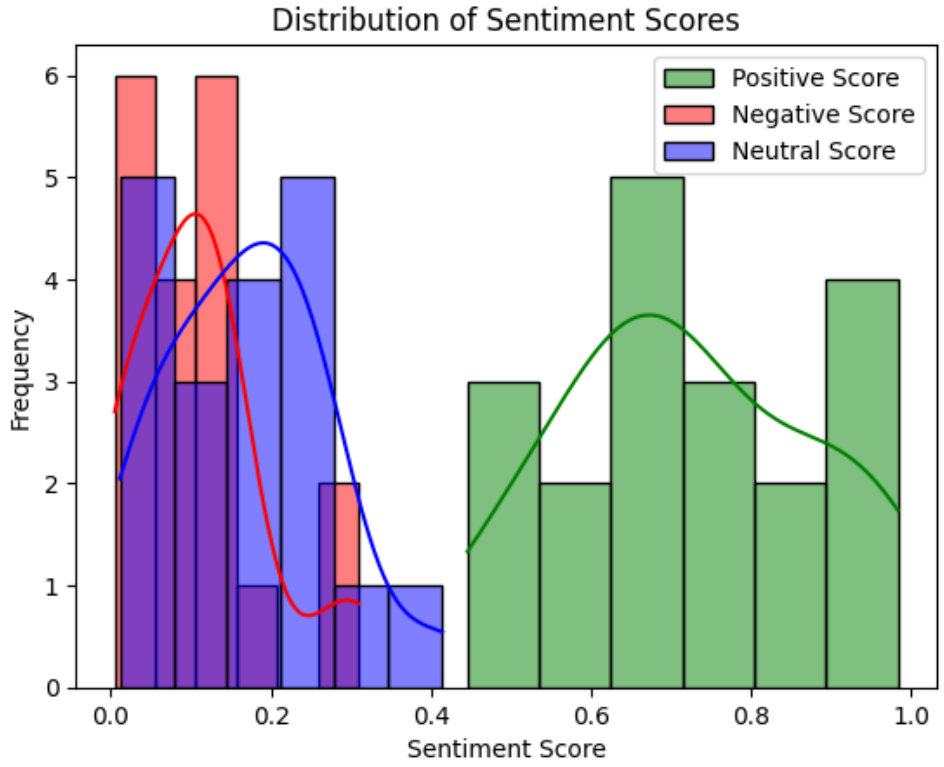
```
  # Correlation between scores
  corr_matrix = df[['positive','negative','neutral','confident',
'hesitant', 'concise', 'enthusiastic', 'speech_speed']].corr()
  sns.heatmap(corr_matrix, annot=True, cmap='coolwarm',
linewidths=0.5)
  plt.title('Correlation Matrix')
  plt.show()
```

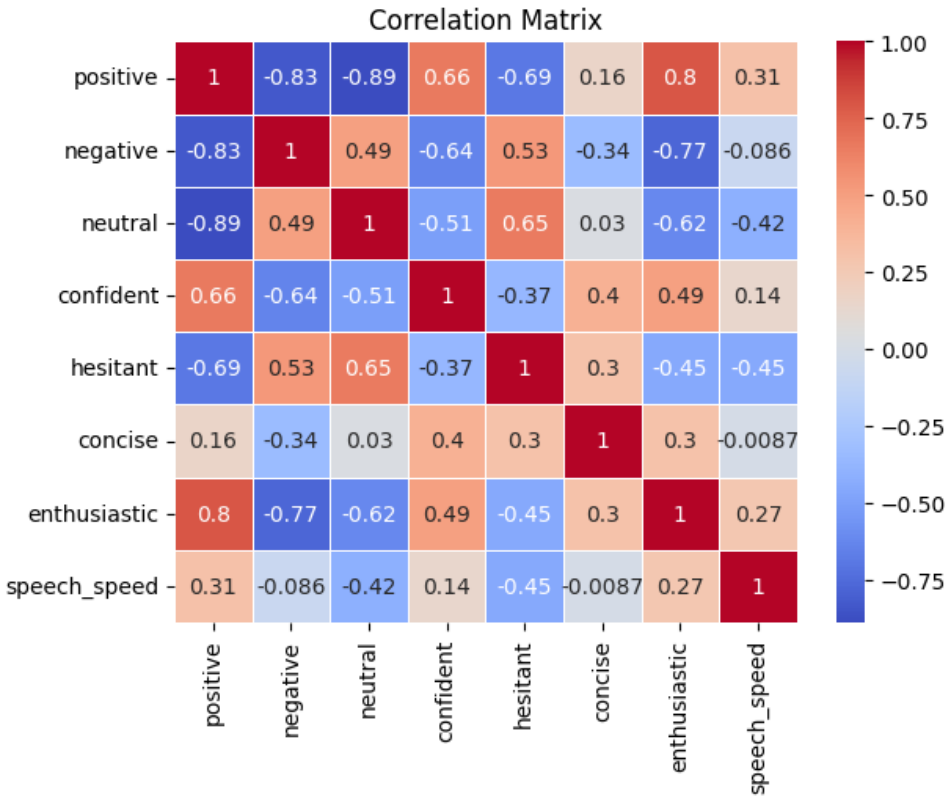
```
eda1 = EDA(pd.read_csv('1.csv'))
```



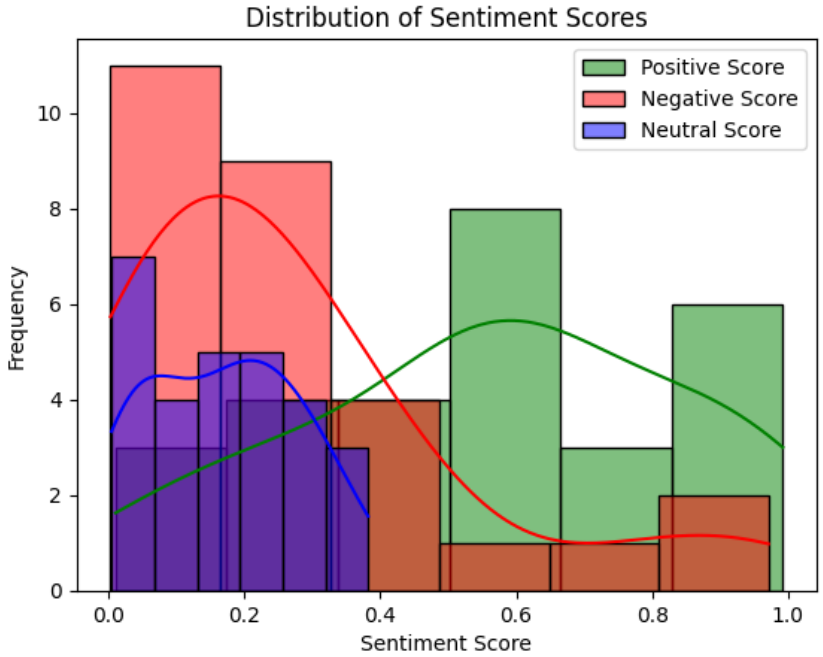


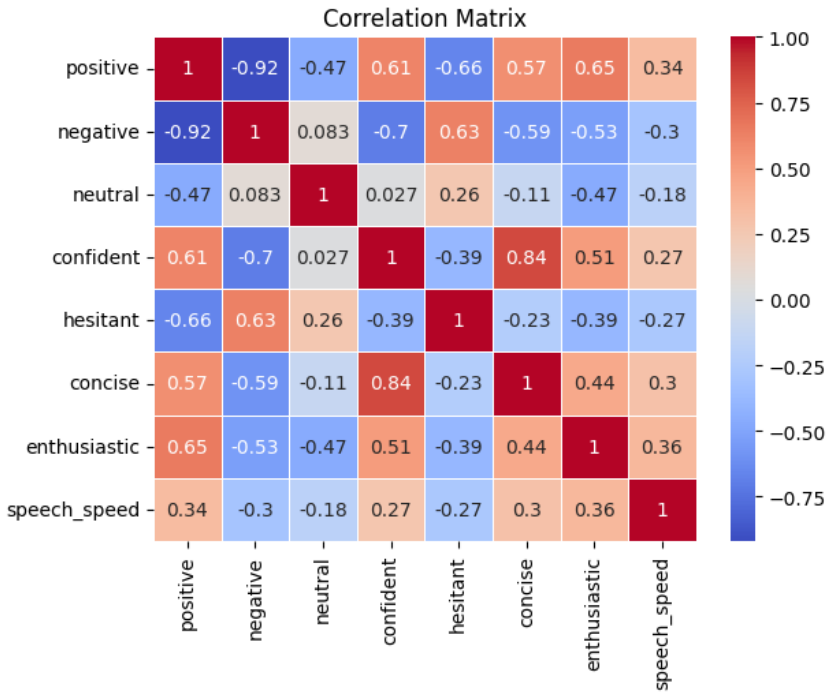

```
eda2 = EDA(pd.read_csv('2.csv'))
```



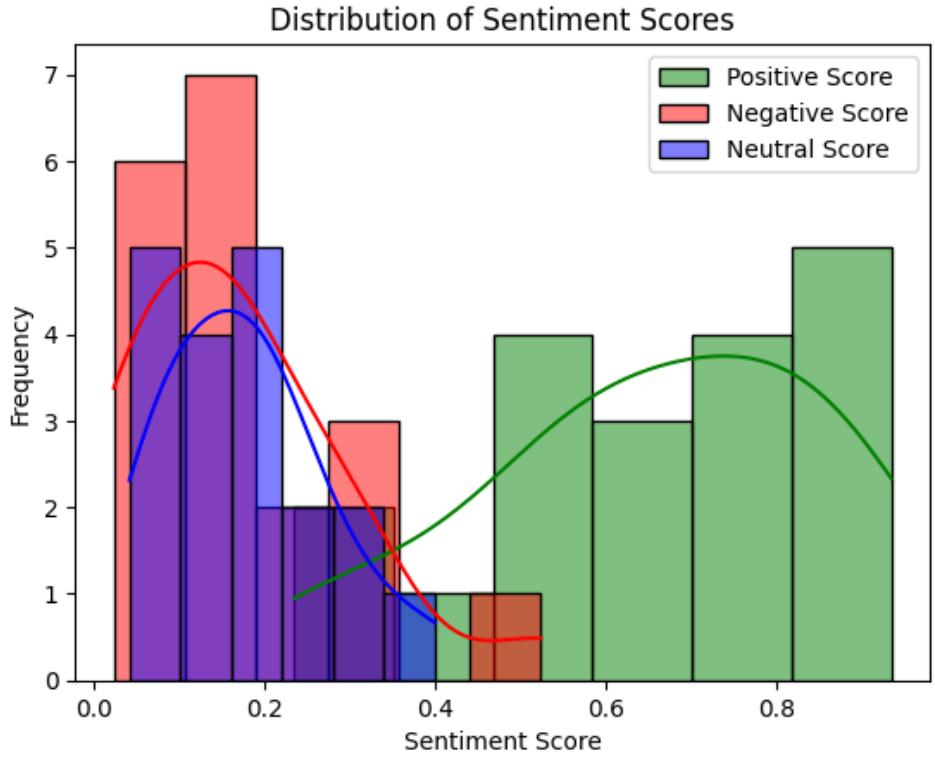


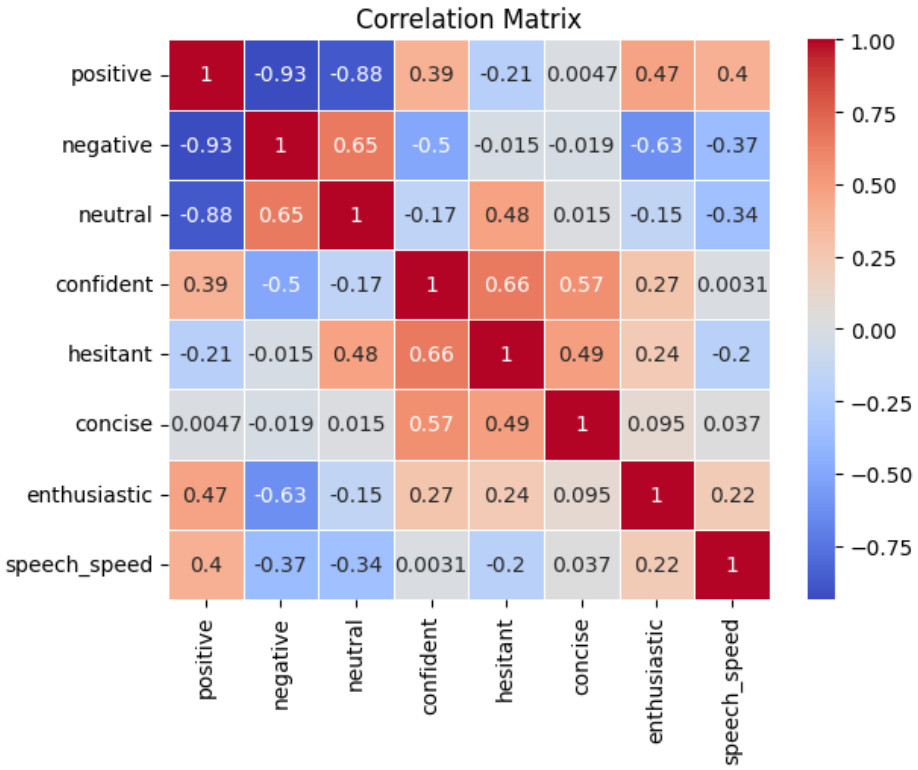

```
eda3 = EDA(pd.read_csv('3.csv'))
```



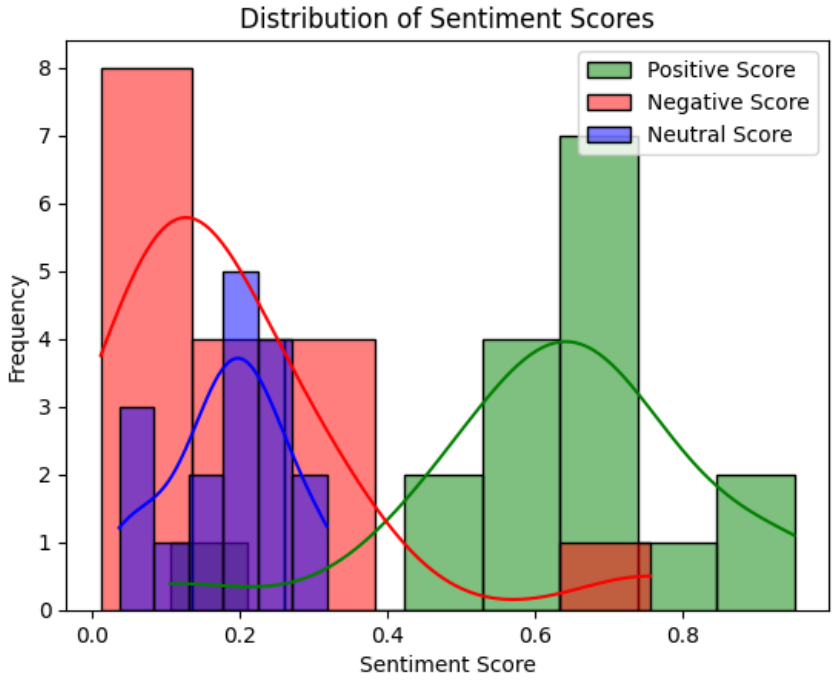


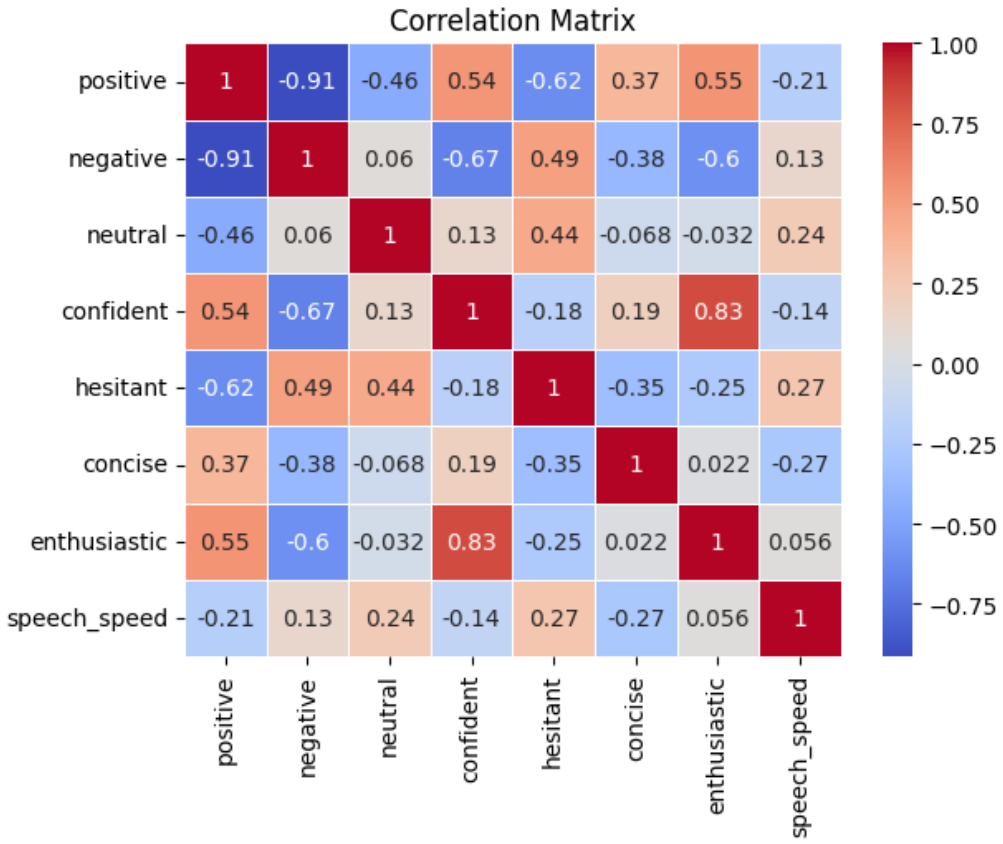

```
eda4 = EDA(pd.read_csv('4.csv'))
```



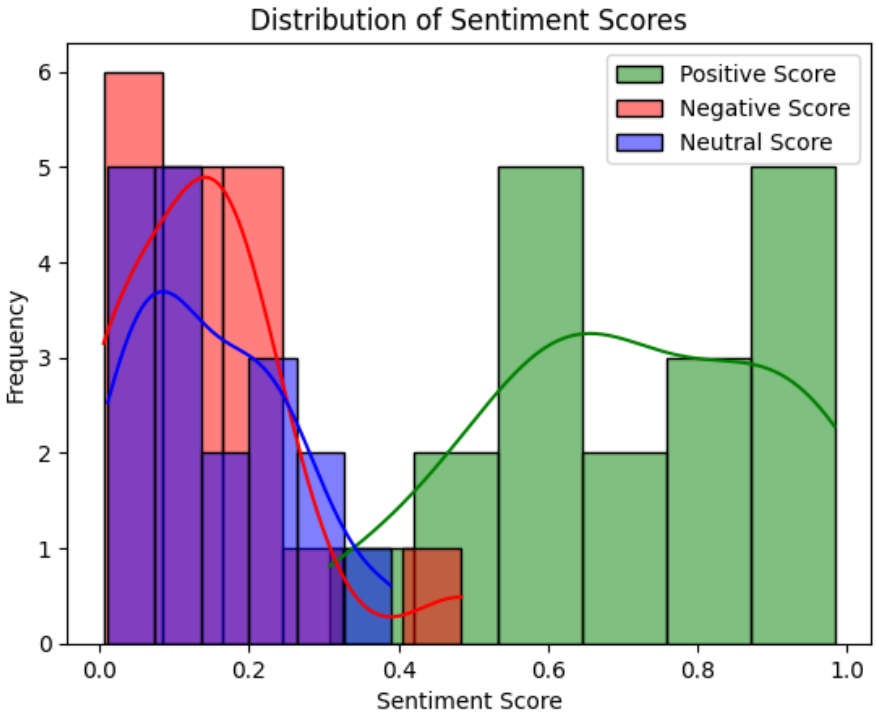


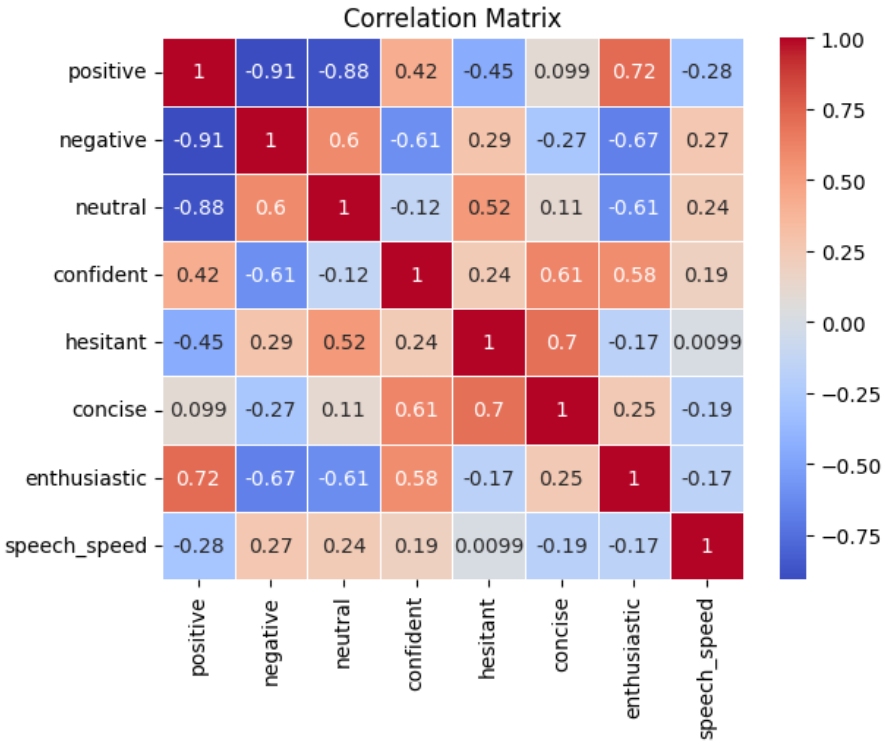

```
eda5 = EDA(pd.read_csv('5.csv'))
```



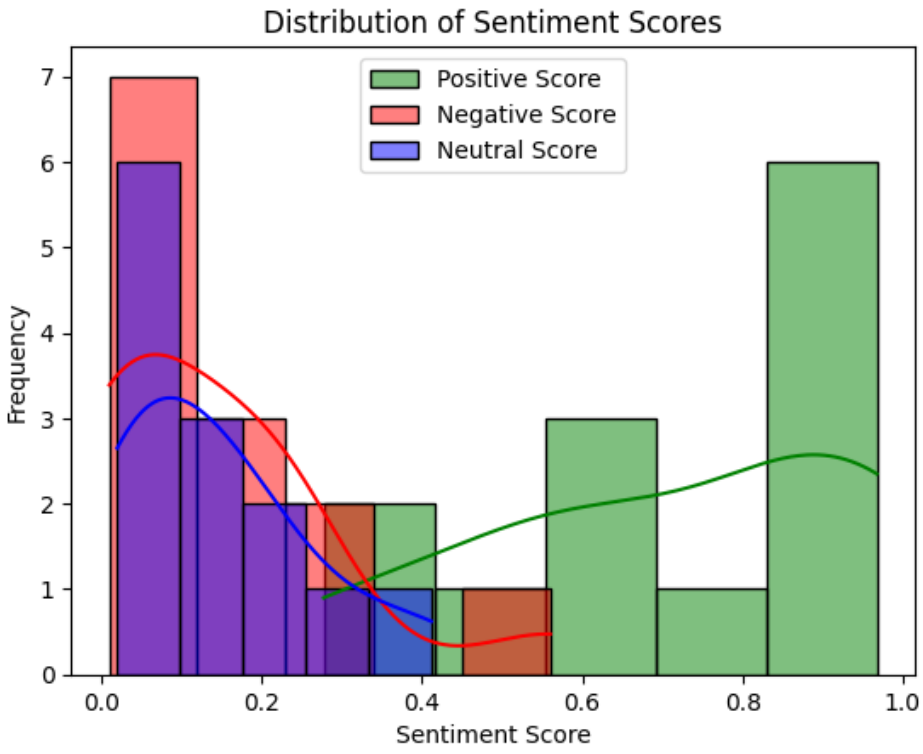


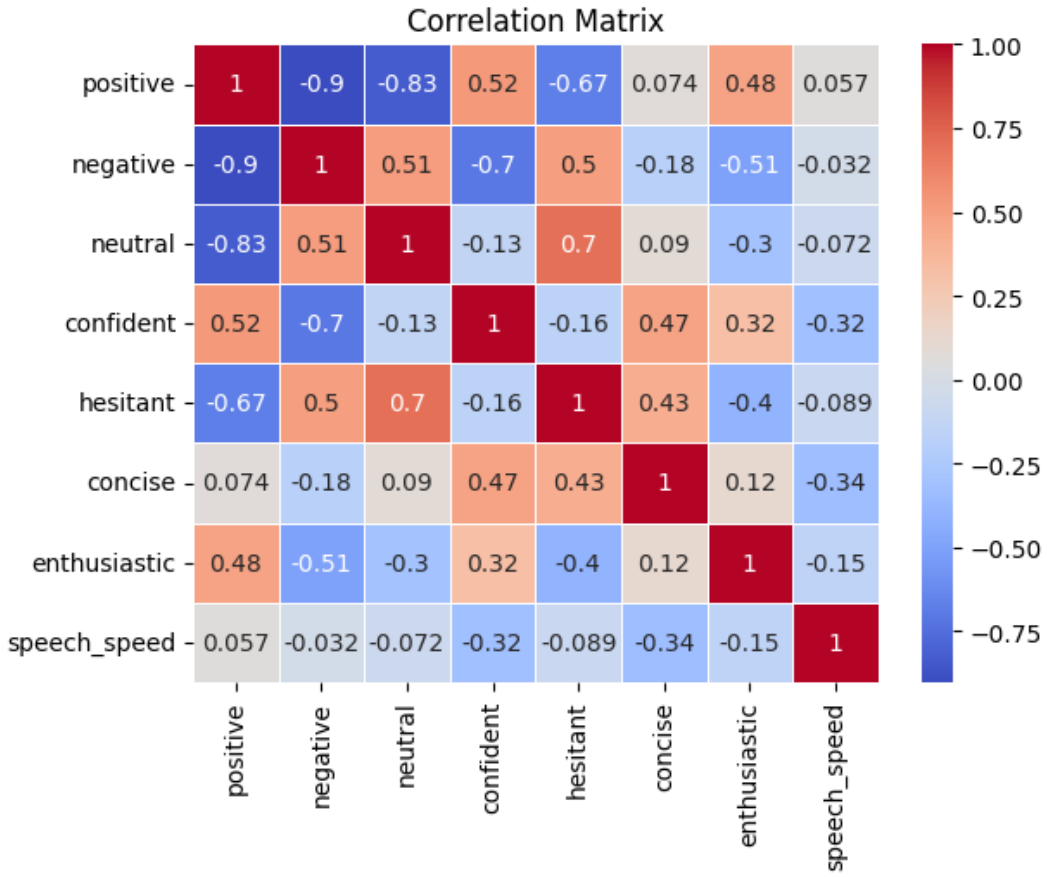

```
eda6 = EDA(pd.read_csv('6.csv'))
```



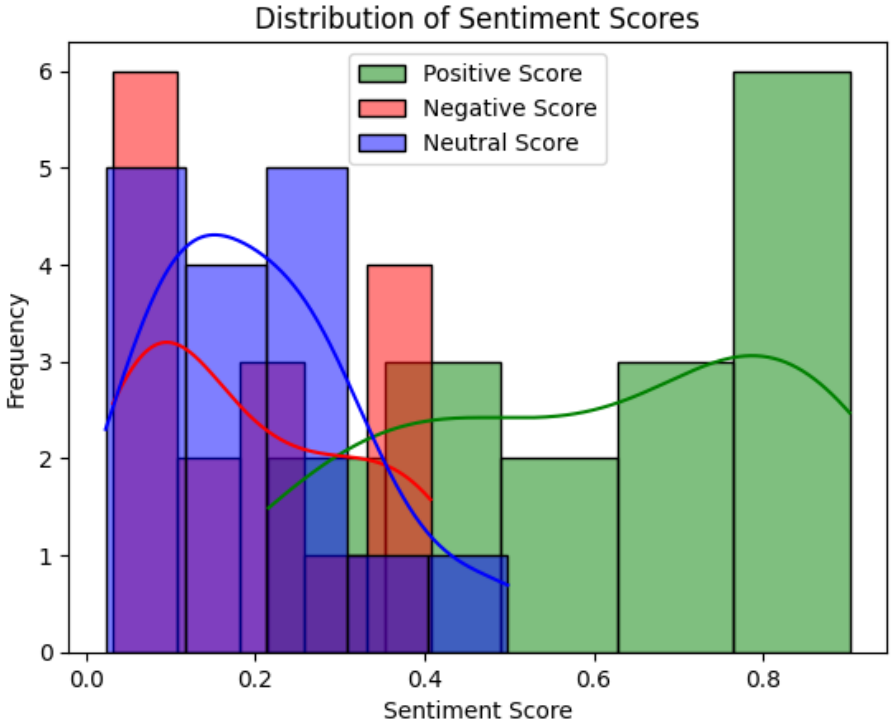


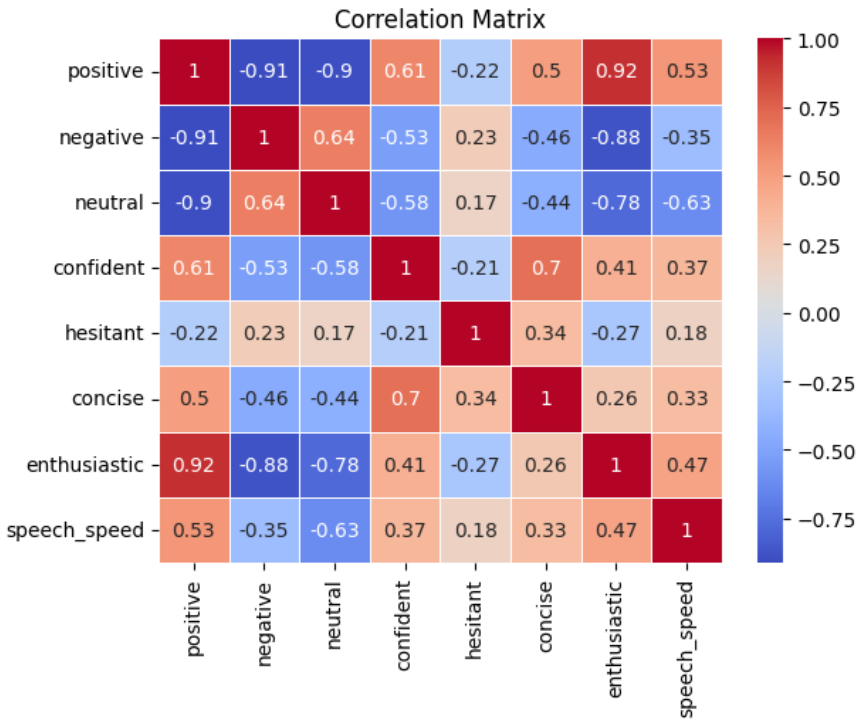

```
eda7 = EDA(pd.read_csv('7.csv'))
```



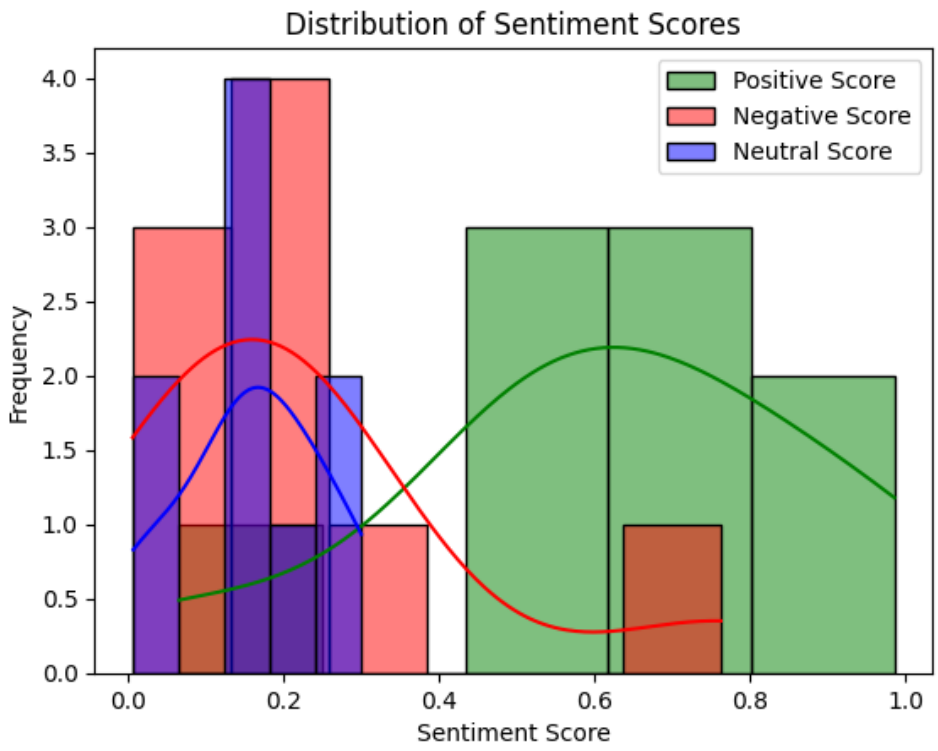


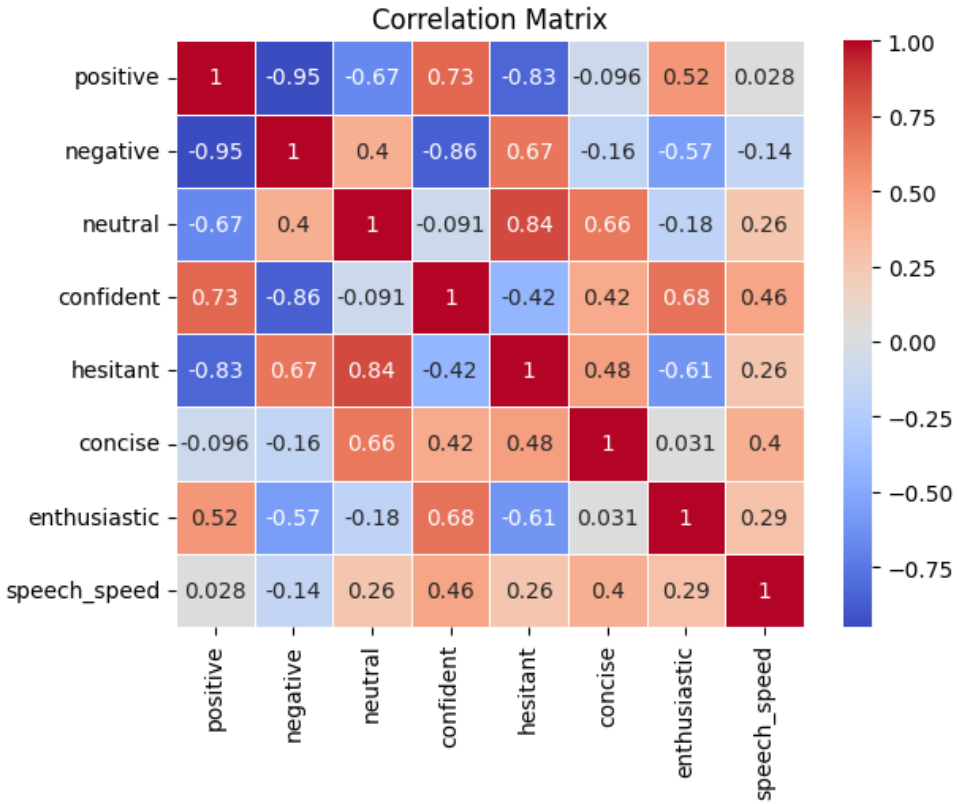

```
eda8 = EDA(pd.read_csv('8.csv'))
```



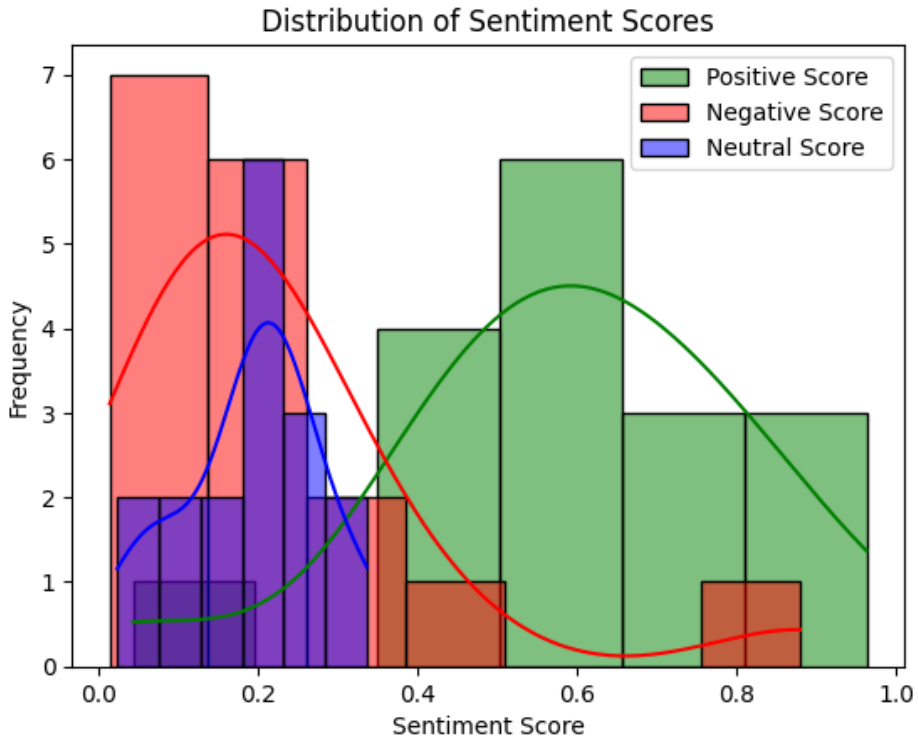


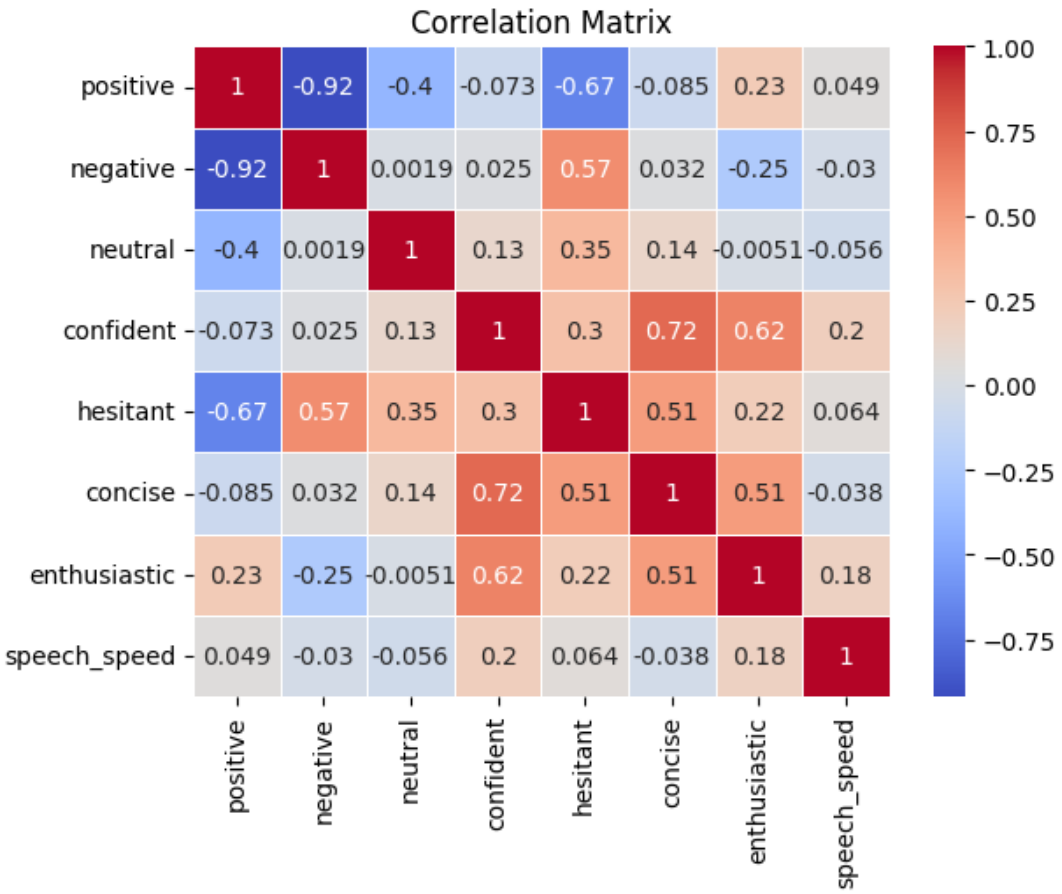

```
eda9 = EDA(pd.read_csv('9.csv'))
```






```
eda10 = EDA(pd.read_csv('10.csv'))
```





One approach to shortlist candidate could be that we can give arbitrary weight to each emotion column and calculate a hypothetical score based on that assumed weights for each person.

```
# We can also try this i.e. giving arbitrary scores to every csv
file by defining our own weights for each column
def transcript_score(df):
  weights = {
    'positive': 0.2,
    'negative': -0.2,
    'neutral': 0.1,
    'confident': 0.2,
    'hesitant': -0.1,
    'concise': 0.1,
    'enthusiastic': 0.1,
    'speech_speed': 0.1
```

```
  }

  df['hypothetical_score'] = np.dot(df[list(weights.keys())].values,
list(weights.values()))

  # Normalizinging the hypothetical score between 0 and 1
  df['hypothetical_score'] = (df['hypothetical_score'] -
df['hypothetical_score'].min()) / \
                             (df['hypothetical_score'].max() -
df['hypothetical_score'].min())

  # Outputing the hypothetical score for this person
  hypothetical_score = df['hypothetical_score'].mean()
  print(f'Overall Hypothetical Score considering all scores:
{hypothetical_score:.2f}')

score1 = transcript_score(pd.read_csv('1.csv'))
score2 = transcript_score(pd.read_csv('2.csv'))
score3 = transcript_score(pd.read_csv('3.csv'))
score4 = transcript_score(pd.read_csv('4.csv'))
score5 = transcript_score(pd.read_csv('5.csv'))
score6 = transcript_score(pd.read_csv('6.csv'))
score7 = transcript_score(pd.read_csv('7.csv'))
score8 = transcript_score(pd.read_csv('8.csv'))
score9 = transcript_score(pd.read_csv('9.csv'))
score10 = transcript_score(pd.read_csv('10.csv'))
```

```
Overall Hypothetical Score considering all scores: 0.52
Overall Hypothetical Score considering all scores: 0.44
Overall Hypothetical Score considering all scores: 0.58
Overall Hypothetical Score considering all scores: 0.66
Overall Hypothetical Score considering all scores: 0.63
Overall Hypothetical Score considering all scores: 0.56
Overall Hypothetical Score considering all scores: 0.63
Overall Hypothetical Score considering all scores: 0.52
Overall Hypothetical Score considering all scores: 0.60
Overall Hypothetical Score considering all scores: 0.55
```

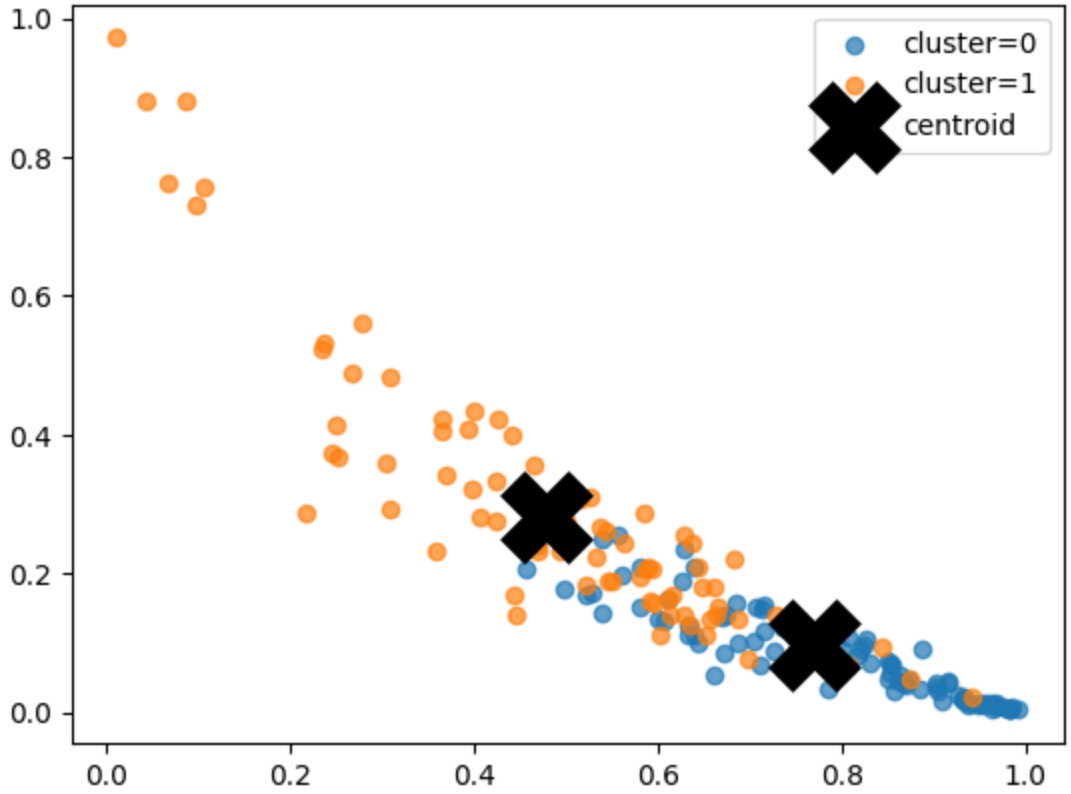
Second approach is using **Self-Organizing Maps (SOMs)** to cluster data into two separate classes and using tSNE.
SOMs are excellent for cluster analysis and data visualization. They can help discover natural groupings or patterns in data. Each neuron in the SOM grid represents a cluster, making it easy to see how data points are grouped in the lower-dimensional space.

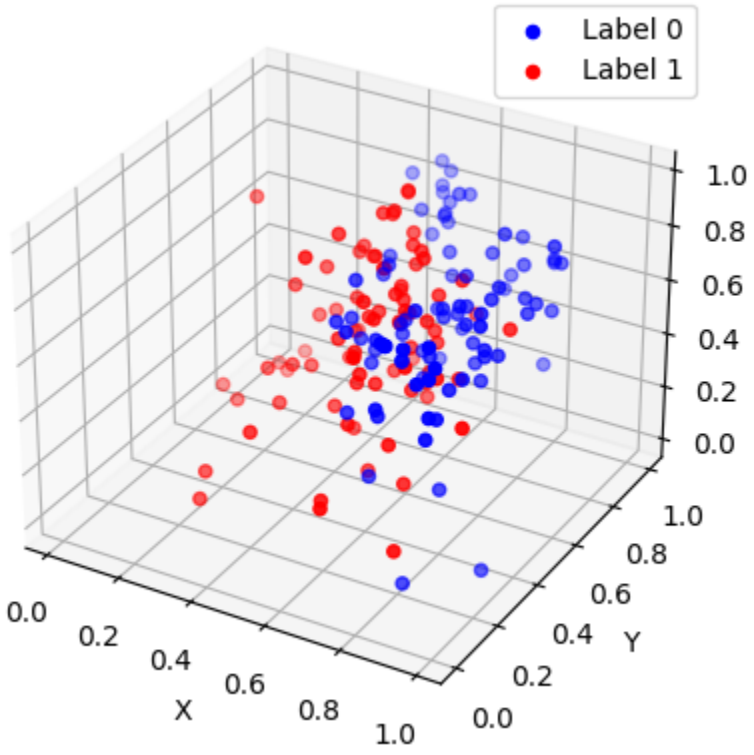
```
# Initialization and training
def som(df):
    emotions = [
        "positive",
        "negative",
        "neutral",
        "confident",
        "hesitant",
        "concise",
        "enthusiastic",
    ]
    new_df = df[emotions].to_numpy()
    som_shape = (1, 2)
    som = MiniSom(
        som_shape[0],
        som_shape[1],
        new_df.shape[1],
        sigma=0.5,
        learning_rate=0.5,
        neighborhood_function="gaussian",
        random_seed=10,
    )

    som.train_batch(new_df, 500, verbose=True)

    # each neuron represents a cluster
    winner_coordinates = np.array([som.winner(x) for x in new_df]).T
    # with np.ravel_multi_index we convert the bidimensional
    # coordinates to a monodimensional index
    cluster_index = np.ravel_multi_index(winner_coordinates, som_shape)
    # plotting the clusters using the first 2 dimentions of the data
```

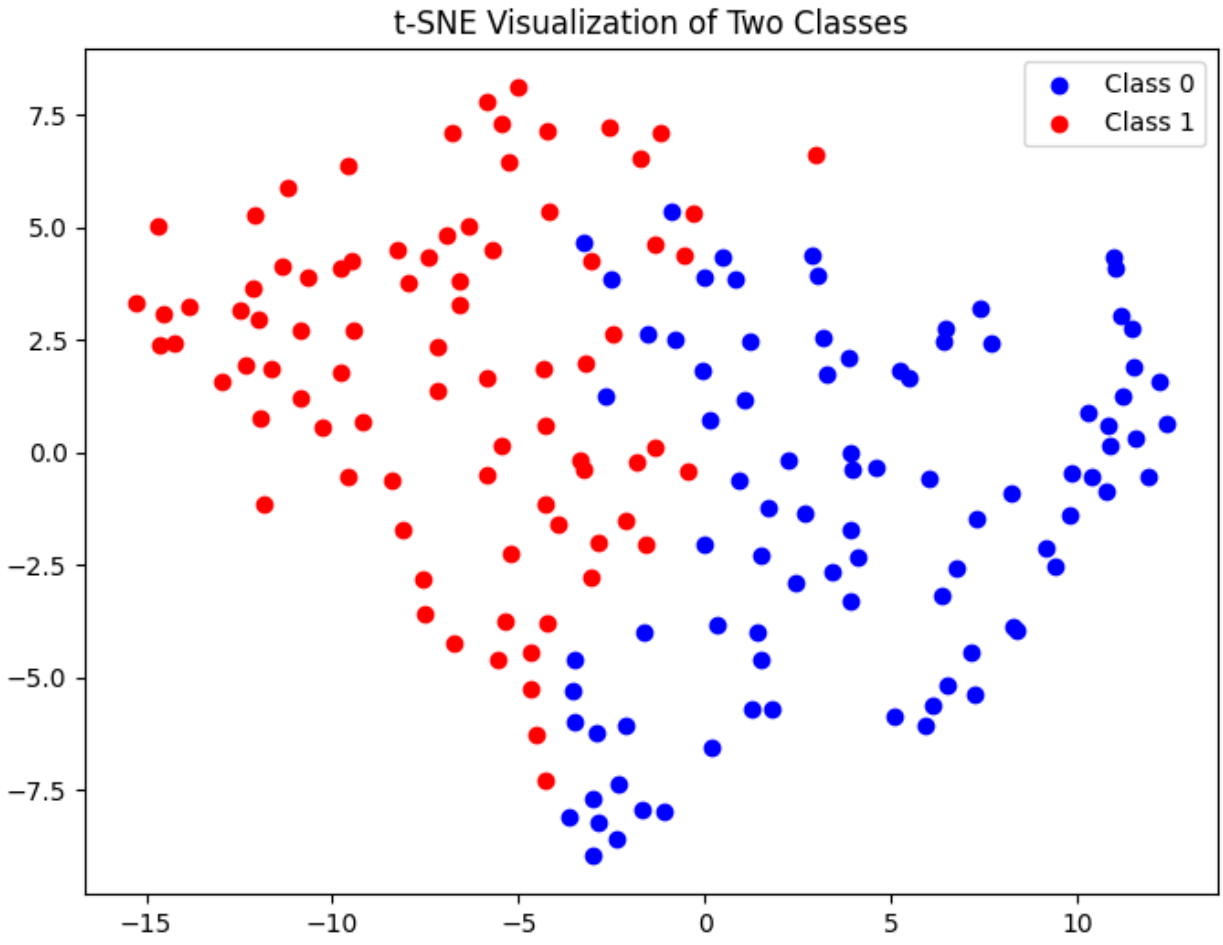
```
    for c in np.unique(cluster_index):
        plt.scatter(
            new_df[cluster_index == c, 0],
            new_df[cluster_index == c, 1],
            label="cluster=" + str(c),
            alpha=0.7,
        )
    # plotting centroids
    for centroid in som.get_weights():
        plt.scatter(
            centroid[:, 0],
            centroid[:, 1],
            marker="x",
            s=80,
            linewidths=35,
            color="k",
            label="centroid",
        )
    plt.legend()
    # Sample 3D coordinates and color labels (replace with your data)
    coordinates = new_df[:, [0, 4, 3]]  # 100 random 3D points
    colors = cluster_index  # Random color labels (0 or 1)

    # Create a 3D scatter plot
    fig = plt.figure()
    ax = fig.add_subplot(111, projection="3d")

    # Separate the points based on color labels and plot them
    ax.scatter(
        coordinates[colors == 0, 0],
        coordinates[colors == 0, 1],
        coordinates[colors == 0, 2],
        c="b",
        label="Label 0",
    )
    ax.scatter(
        coordinates[colors == 1, 0],
        coordinates[colors == 1, 1],
        coordinates[colors == 1, 2],
        c="r",
```

```
        label="Label 1",
    )

    # Set labels and legend
    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    ax.set_zlabel("Z")
    ax.legend()

    # Show the plot
    plt.show()
    # Creating random features and labels for binary classification
    X = new_df
    y = cluster_index

    # Split the dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
    )

    # Build a simple feedforward neural network
    model = keras.Sequential(
        [
            keras.layers.Dense(32, activation="relu", input_shape=(7,)),
            keras.layers.Dense(16, activation="relu"),
            keras.layers.Dense(
                1, activation="sigmoid"
            ),  # Output layer for binary classification
        ]
    )

    # Compile the model
    model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["accuracy"])

    # Train the model
    epochs = 50
    batch_size = 32
```

```
    model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size,
verbose=2)

    # Evaluate the model on the test set
    y_pred = model.predict(X_test)
    y_pred_binary = (y_pred > 0.5).astype(int)

    accuracy = accuracy_score(y_test, y_pred_binary)
    print(f"Accuracy on the test set: {accuracy:.2f}")

    # Generate a synthetic dataset with two classes (replace with your own
data)
    np.random.seed(42)

    # Create a t-SNE model
    tsne = TSNE(n_components=2, perplexity=30, random_state=42)

    # Fit and transform the data
    X_tsne = tsne.fit_transform(X)

    # Plot the t-SNE visualization
    plt.figure(figsize=(8, 6))
    plt.scatter(X_tsne[y == 0, 0], X_tsne[y == 0, 1], label="Class 0",
c="blue")
    plt.scatter(X_tsne[y == 1, 0], X_tsne[y == 1, 1], label="Class 1",
c="red")
    plt.title("t-SNE Visualization of Two Classes")
    plt.legend()

    plt.show()

som(combined_data)
```


cluster=0
cluster=1
centroid
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Label 0
Label 1
X
Y
0.0
0.2
0.4
0.6
0.8
1.0



Now using these techniques, we can form clusters of the people who are deserving for the role in the company.