Deliverable for I'mbesideyou Data Scientist Internship

Vishnu Tirth Bysani

EDA Analysis and Clustering

Contents

Objective

We have been given the dataset of 10 candidates containing the emotion scores, transcript scores, and corresponding transcripts extracted from their introduction videos. Using Prompt Engineering and Exploratory Data Analysis (EDA), we need to provide valuable and actionable insights from the data.

Main Steps followed in Exploratory Data Analysis:

1. Understanding the Data
2. Cleaning the Data
3. Relationship Analysis

After we have an idea of the data using Exploratory Data Analysis, we need to predict if the student can be recruited or not and we also need to analyze their communication skills. This can be done with the help of a Clustering algorithm.

Clustering is a technique in data analysis that groups similar data points together based on their characteristics. It's an unsupervised learning approach, meaning it doesn't rely on any predefined results. Instead, it aims to find natural groupings within a dataset. Clustering is used to discover patterns, segment data, and gain insights from complex datasets.

Exploratory Data Analysis (EDA)

The main objective is to gain a deep understanding of the dataset you're working with. It involves examining, visualizing, and summarizing the main characteristics of the data to uncover patterns, spot anomalies, and generate hypotheses. EDA is typically performed before applying any formal statistical methods or building predictive models.

Now let us perform Exploratory Data Analysis on the data provided.
For performing EDA, we first need to import some libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
```

NumPy is a powerful numerical computing library in Python. It provides support for arrays, matrices, and a large number of mathematical functions to operate on these data structures. It is a fundamental library for numerical and scientific computing in Python. It forms the basis for many other libraries and tools in the scientific Python ecosystem.

Pandas is a data manipulation library that provides data structures (Series and DataFrame) to efficiently handle and analyze structured data. It is widely used in data analysis and manipulation tasks, especially in fields like data science, machine learning, and quantitative finance.

Seaborn is a statistical data visualization library that builds on top of Matplotlib. It provides a high-level interface for creating informative and visually appealing statistical graphics. It is particularly well-suited for visualizing complex datasets with multiple variables. It is widely used in data analysis, statistical modeling, and data science projects

Emotion Data:

Understanding the Data

a. As, the emotion_data folder was in the same directory as the jupyter notebook used, this command loads the emotion.csv file of the 1st student into the emotion_df dataframe using pandas' read_csv command.

```
emotion_df=pd.read_csv("emotion_data/1/emotion.csv")
```

b. We need to merge the emotion data of all the students, so to do this, we can take the help of a for loop. The for loop starts from 2 and ends at 10 as we have already loaded the data of 1st student and we have 10 students in total. We can the load the data of each student in a temporary dataframe df1, and merge it with emotion_df using pd.merge. We are using outer merge. An outer merge returns all the rows when there is a match in one of the DataFrames. It combines the effect of left and right merges. As we have same columns in all the dataframes, outer merge will combine all the dataframes to give us our final emotion_df that contains the emotion data of the 10 students.

```
for i in range(2,11):
    df1=pd.read_csv("emotion_data/"+str(i)+"/emotion.csv")
    emotion_df=pd.merge(emotion_df,df1,how='outer')
```

c. The head command returns the first 5 rows of the dataframe.

```
emotion_df.head()
```

	movie_id	image_seq	angry	disgust	fear	happy	sad	surprise	neutral	dominant_emotion
0	93663f94-bf0a-4ce8-a29a-a5236cc7fe6a	0	4.31735	0.000594	2.879790	1.65035	2.779980	0.600814	87.77110	neutral
1	93663f94-bf0a-4ce8-a29a-a5236cc7fe6a	1	53.22530	2.981640	12.736800	1.52347	1.051320	27.216800	1.26462	angry
2	93663f94-bf0a-4ce8-a29a-a5236cc7fe6a	2	8.79651	0.029468	2.968160	16.83150	39.884600	0.279335	31.21050	sad
3	93663f94-bf0a-4ce8-a29a-a5236cc7fe6a	3	9.45303	0.106778	1.553080	20.93010	3.503870	0.909426	63.54370	neutral
4	93663f94-bf0a-4ce8-a29a-a5236cc7fe6a	4	56.00020	0.000004	0.162231	5.58358	0.197026	12.807600	25.24940	angry

d. Using the shape command, we know that emotion_df has 747 rows and 10 columns.

```
emotion_df.shape
```

```
(747, 10)
```

e. We can get details of the dataframe using describe command.

```
emotion_df.describe()
```

	image_seq	angry	disgust	fear	happy	sad	surprise	neutral
count	747.000000	747.000000	7.470000e+02	747.000000	7.470000e+02	747.000000	7.470000e+02	7.470000e+02
mean	44.631861	6.223748	1.586366e-01	19.003329	1.187686e+01	12.636010	5.490173e+00	4.461124e+01
std	27.270607	13.007503	1.101350e+00	25.615108	2.411922e+01	20.387900	1.537337e+01	3.966181e+01
min	0.000000	0.000015	1.680020e-14	0.000002	1.959540e-09	0.000022	4.278190e-07	7.548270e-07
25%	21.000000	0.193459	1.355525e-06	0.687261	8.623350e-02	0.448145	1.265175e-02	4.210900e+00
50%	44.000000	1.308920	1.866540e-04	5.731700	8.230810e-01	2.619840	2.016130e-01	3.377610e+01
75%	68.000000	5.617910	1.171565e-02	30.086950	8.921020e+00	13.845600	2.054355e+00	8.916495e+01
max	99.000000	95.056300	2.150890e+01	99.750200	9.998180e+01	98.333400	9.976910e+01	9.999140e+01

Cleaning the Data

a. Using the columns command, we can find out all the columns of the dataframe.

```
emotion_df.columns
```

```
Index(['movie_id', 'image_seq', 'angry', 'disgust', 'fear', 'happy', 'sad',
       'surprise', 'neutral', 'dominant_emotion'],
      dtype='object')
```

b. We do need the movie_id, and the image_seq as they have no relevance in our data analysis. Also the dominant emotion is not a factor we want to consider as we want to look at all the emotions of the student when analysing. So we can drop these columns.

```
emotion_df=emotion_df.drop(['movie_id','image_seq','dominant_emotion'],axis=1)
emotion_df
```

	angry	disgust	fear	happy	sad	surprise	neutral
0	4.317350	5.942640e-04	2.879790	1.650350e+00	2.779980	0.600814	87.77110
1	53.225300	2.981640e+00	12.736800	1.523470e+00	1.051320	27.216800	1.26462
2	8.796510	2.946810e-02	2.968160	1.683150e+01	39.884600	0.279335	31.21050
3	9.453030	1.067780e-01	1.553080	2.093010e+01	3.503870	0.909426	63.54370
4	56.000200	4.152410e-06	0.162231	5.583580e+00	0.197026	12.807600	25.24940
...	...	...	...	...	...	...	...
742	21.623500	3.223740e-01	55.701200	1.837300e+00	14.471100	2.007100	4.03747
743	0.483833	8.153230e-05	83.415300	2.197600e+00	12.474100	0.059187	1.36993
744	0.175224	4.728190e-10	13.272400	1.959540e-09	63.701500	0.000002	22.85090
745	0.326095	2.007640e-05	1.177400	3.822260e-02	33.006200	0.011101	65.44100
746	0.041253	4.232110e-08	0.005674	1.281550e-02	0.843036	0.035985	99.06120

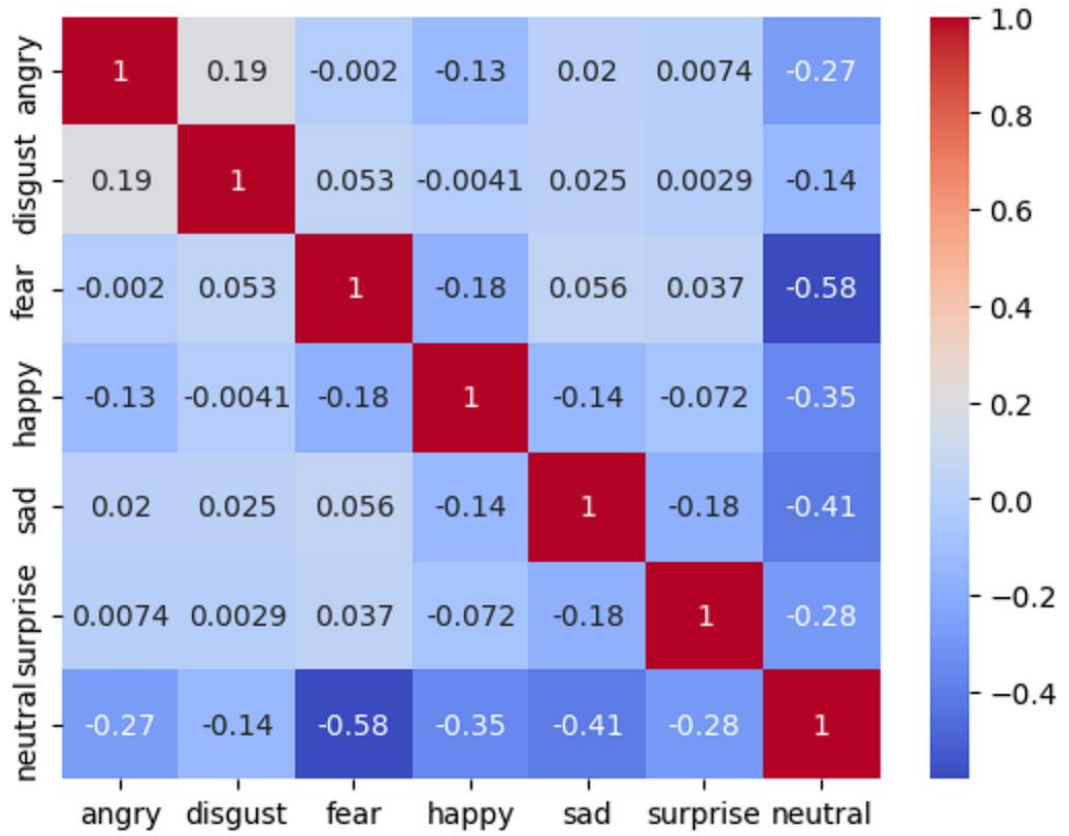
747 rows × 7 columns

Relationship Analysis

a. Using the sns library we can plot the heatmap of the correlation matrix, which shows how correlated each variable is with each other.

```
correlation_matrix = emotion_df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
```

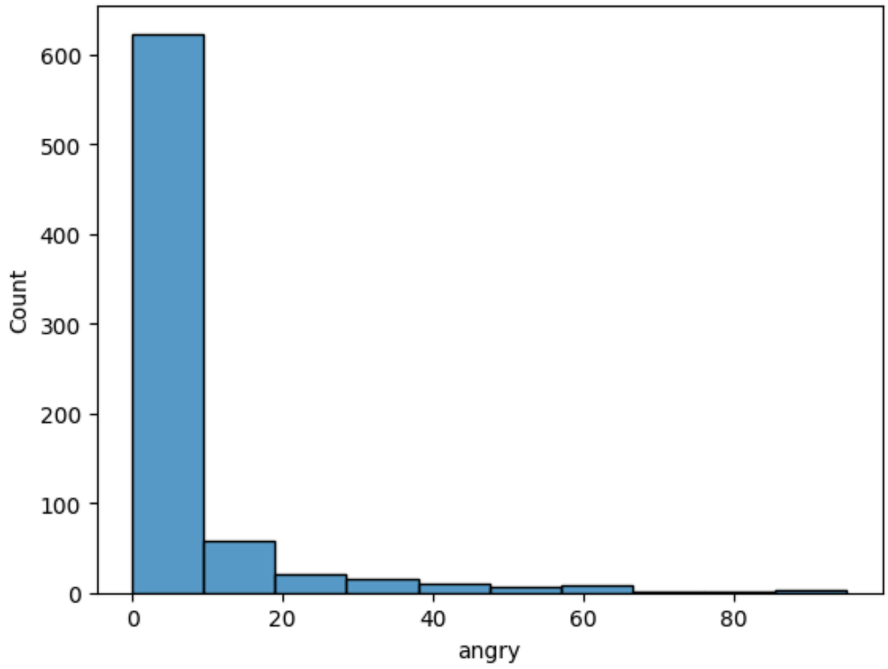
```
<Axes: >
```



b. Now, we can use a histogram to check the frequency of values occurring for each column.

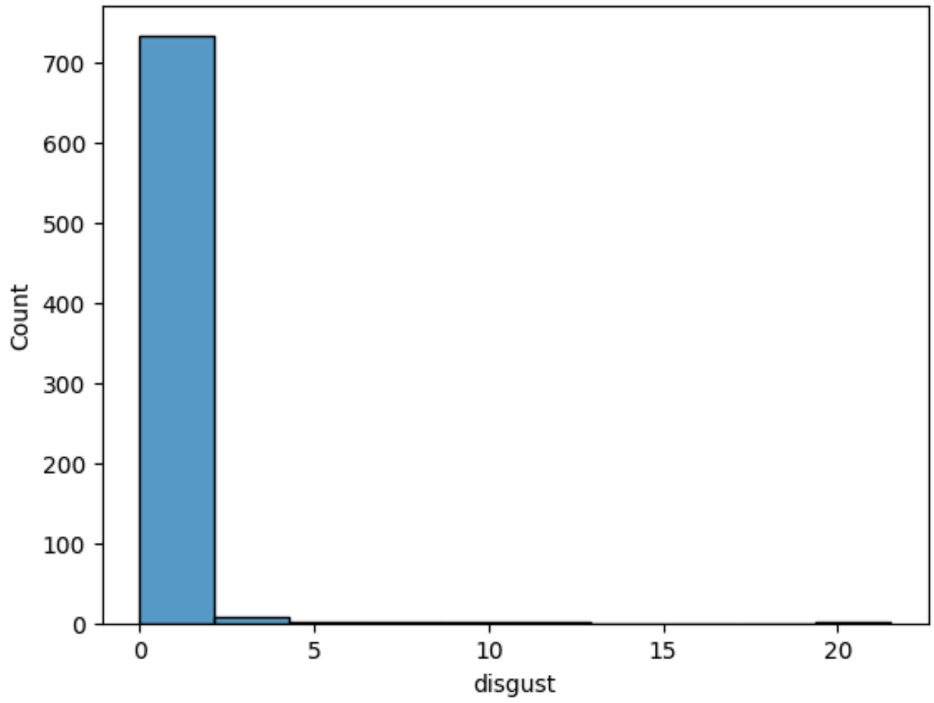
```
sns.histplot(emotion_df['angry'],bins=10)
```

```
<Axes: xlabel='angry', ylabel='Count'>
```



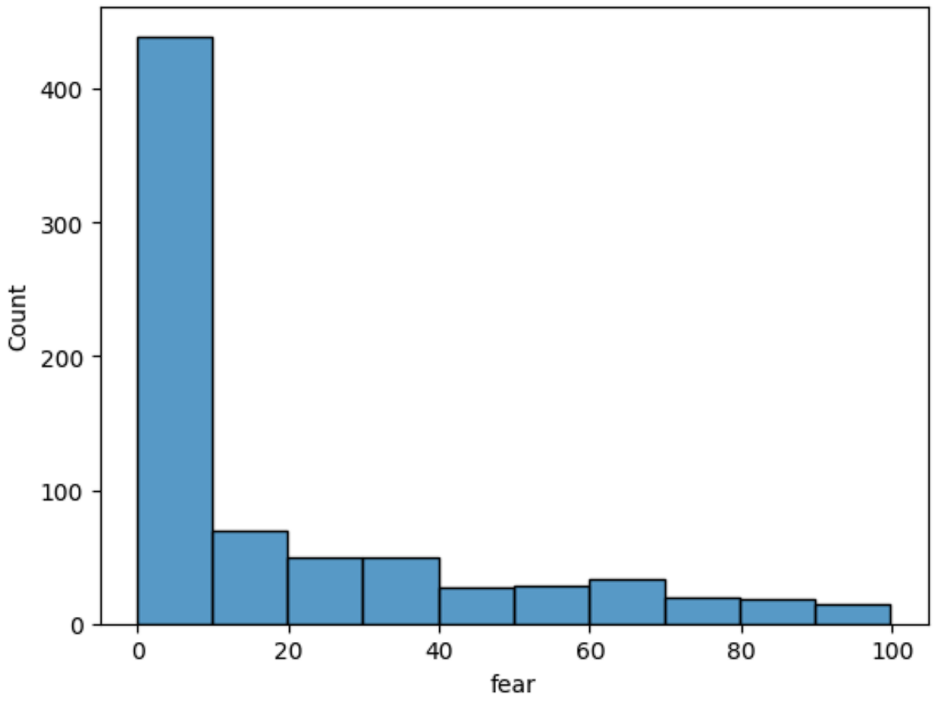

```
sns.histplot(emotion_df['disgust'],bins=10)
```

```
<Axes: xlabel='disgust', ylabel='Count'>
```



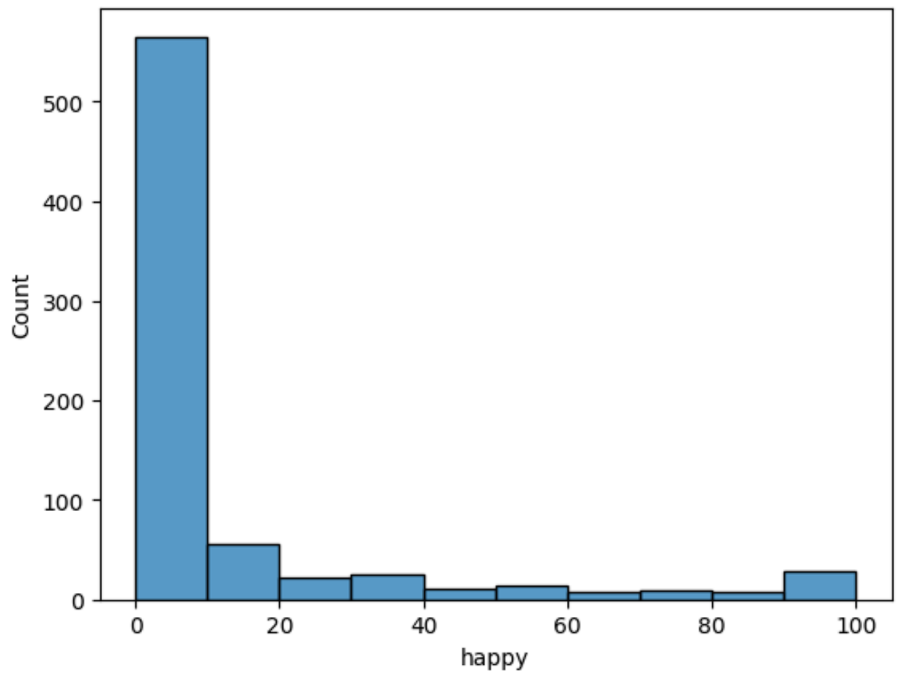
```
sns.histplot(emotion_df['fear'],bins=10)
```

```
<Axes: xlabel='fear', ylabel='Count'>
```



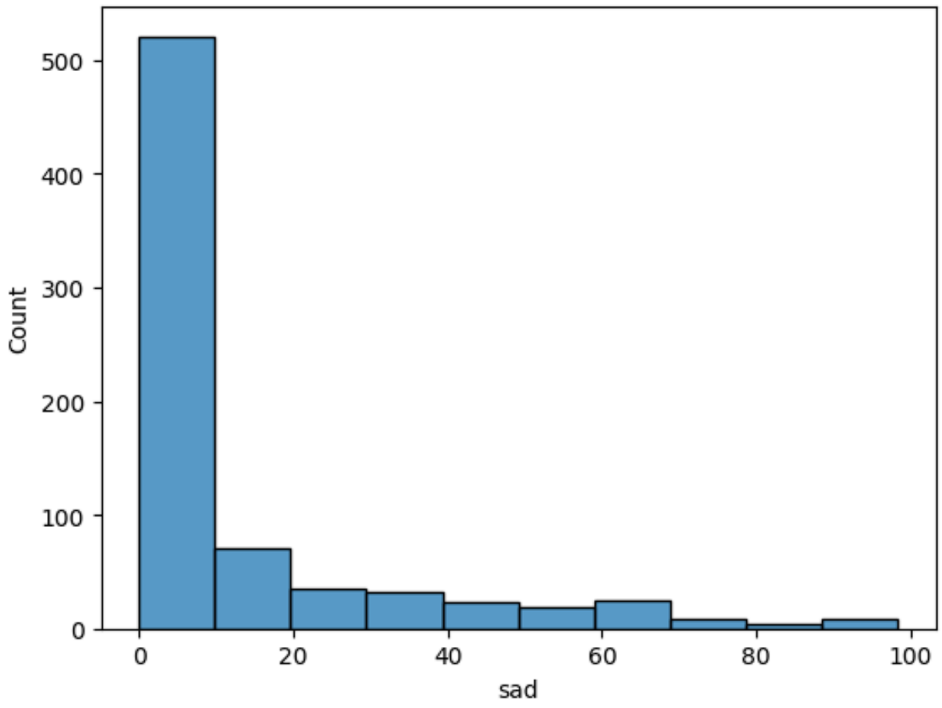

```
sns.histplot(emotion_df['happy'],bins=10)
```

```
<Axes: xlabel='happy', ylabel='Count'>
```



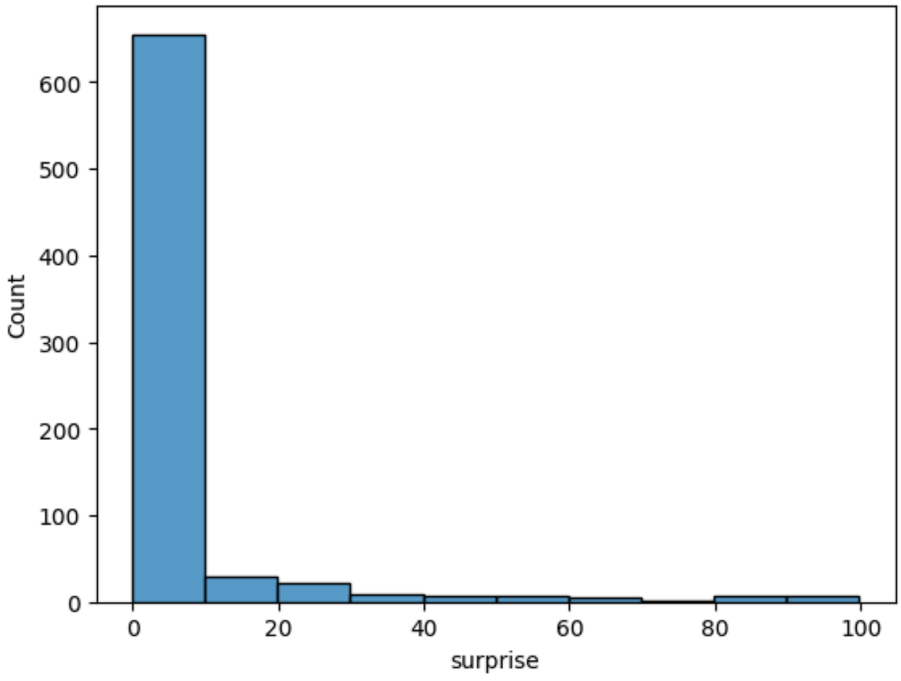
```
sns.histplot(emotion_df['sad'],bins=10)
```

```
<Axes: xlabel='sad', ylabel='Count'>
```



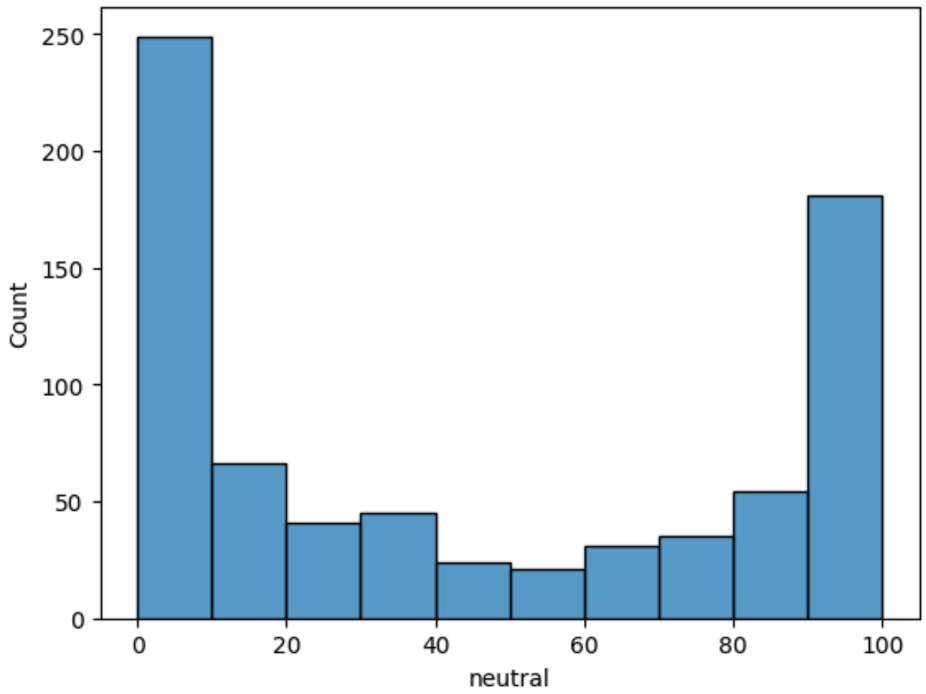

```
sns.histplot(emotion_df['surprise'],bins=10)
```

```
<Axes: xlabel='surprise', ylabel='Count'>
```



```
sns.histplot(emotion_df['neutral'],bins=10)
```

```
<Axes: xlabel='neutral', ylabel='Count'>
```



From the above data, we can infer that most of the students have neutral emotion scores. It is also noticed that the percentage of angry, disgust and sad values are the smallest which shows the students have good emotions throughout their transcript videos. This is a good sign and shows the students have a good body language which makes them a desirable candidate for recruitment. Proper emotional control and body language is essential for good communication.

Gaze Data:

Understanding the Data

a. This command loads the gaze.csv file of the 1st student into the gaze_df dataframe using pandas' read_csv command. The last row of gaze_df has been deleted, as for student 1 the number of rows of emotion.csv and gaze.csv differ by 1, so for sake of uniformity and as it will not affect the analysis and prediction much, the last row has been removed.

```
gaze_df=pd.read_csv("emotion_data/1/gaze.csv")
gaze_df=gaze_df.drop(gaze_df.index[-1])
```

b. We need to merge the gaze data for all the students, so to do this, we can take the help of a for loop. The for loop starts from 2 and ends at 10 as we have already loaded the data of 1st student and we have 10 students in total. We can then load the data of each student in a temporary dataframe df2, and outer-merge it with gaze_df using pd.merge. As we have same columns in all the dataframes, outer merge will combine all the dataframes to give us our final gaze_df that contains the gaze data of the 10 students.

```
for i in range(2,11):
    df2=pd.read_csv("emotion_data/"+str(i)+"/gaze.csv")
    gaze_df=pd.merge(gaze_df,df2,how='outer')
```

c. The head command returns the first 5 rows of the dataframe.

```
gaze_df.head()
```

	movie_id	image_seq	gaze	blink	eye_offset
0	93663f94-bf0a-4ce8-a29a-a5236cc7fe6a	1	1	0	6.2253
1	93663f94-bf0a-4ce8-a29a-a5236cc7fe6a	2	1	0	22.7274
2	93663f94-bf0a-4ce8-a29a-a5236cc7fe6a	3	1	0	2.5704
3	93663f94-bf0a-4ce8-a29a-a5236cc7fe6a	4	1	0	21.1097
4	93663f94-bf0a-4ce8-a29a-a5236cc7fe6a	5	1	0	1.8453

d. Using the shape command, we know that gaze_df has 747 rows and 5 columns.

```
gaze_df.shape
```

```
(747, 5)
```

e. We can get details of the dataframe using describe command.

```
gaze_df.describe()
```

	image_seq	gaze	blink	eye_offset
count	747.000000	747.000000	747.000000	747.000000
mean	45.614458	0.740295	0.096386	14.231156
std	27.243194	0.438767	0.295317	19.291602
min	1.000000	0.000000	0.000000	-35.804300
25%	22.000000	0.000000	0.000000	0.827200
50%	45.000000	1.000000	0.000000	8.668700
75%	69.000000	1.000000	0.000000	25.273250
max	100.000000	1.000000	1.000000	91.234700

Cleaning the Data

a. Using the columns command, we can find out all the columns of the dataframe.

```
gaze_df.columns
```

```
Index(['movie_id', 'image_seq', 'gaze', 'blink', 'eye_offset'], dtype='object')
```

b. We do not need the movie_id, and the image_seq as they have no relevance in our data analysis. Also, we cannot judge a student based on whether they are blinking or not. So we can drop these columns.

```
gaze_df=gaze_df.drop(['movie_id','image_seq','blink'],axis=1)
gaze_df
```

	gaze	eye_offset
0	1	6.2253
1	1	22.7274
2	1	2.5704
3	1	21.1097
4	1	1.8453
...	...	...
742	0	41.0800
743	1	-11.6497
744	1	8.1219
745	1	3.7459
746	1	-15.7229

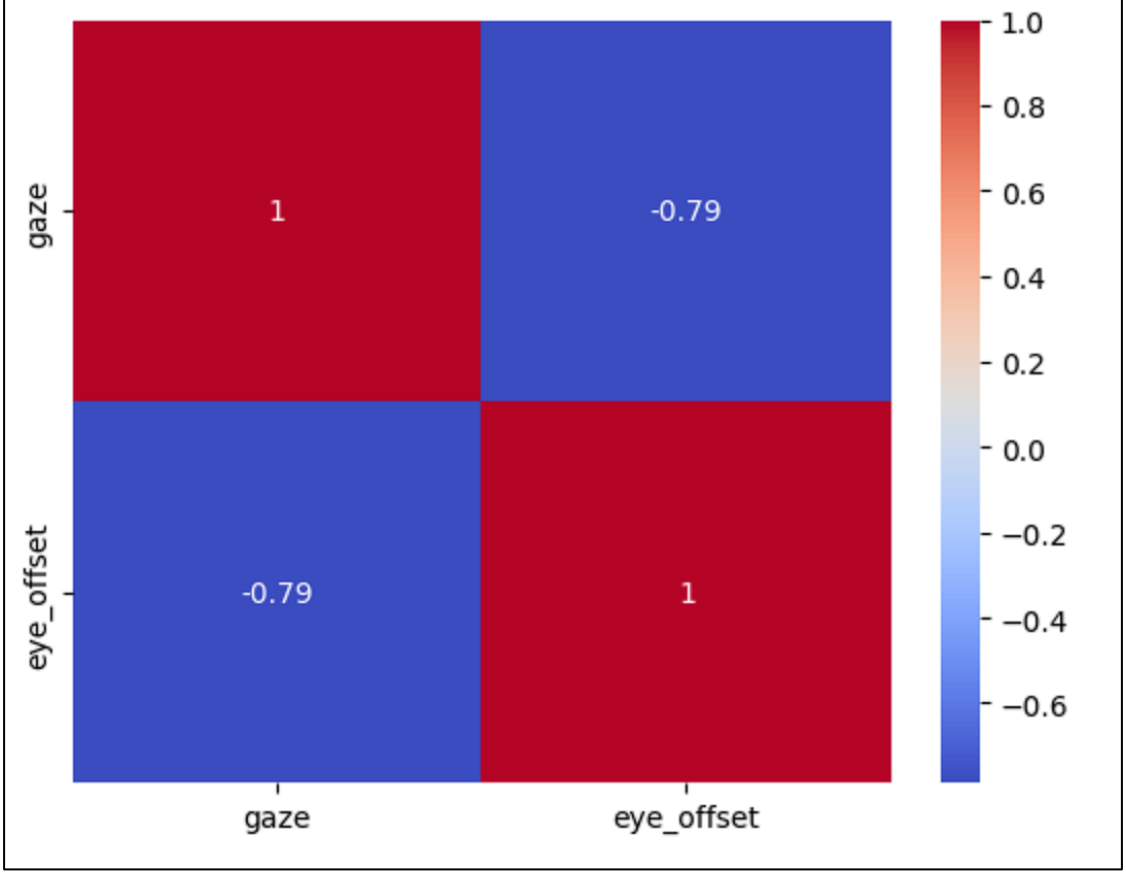
747 rows × 2 columns

Relationship Analysis

a. Using the sns library we can plot the heatmap of the correlation matrix, which shows how correlated each variable is with each other. We can clearly there is high correlation between gaze and eye_offset.

```
correlation_matrix = gaze_df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
```

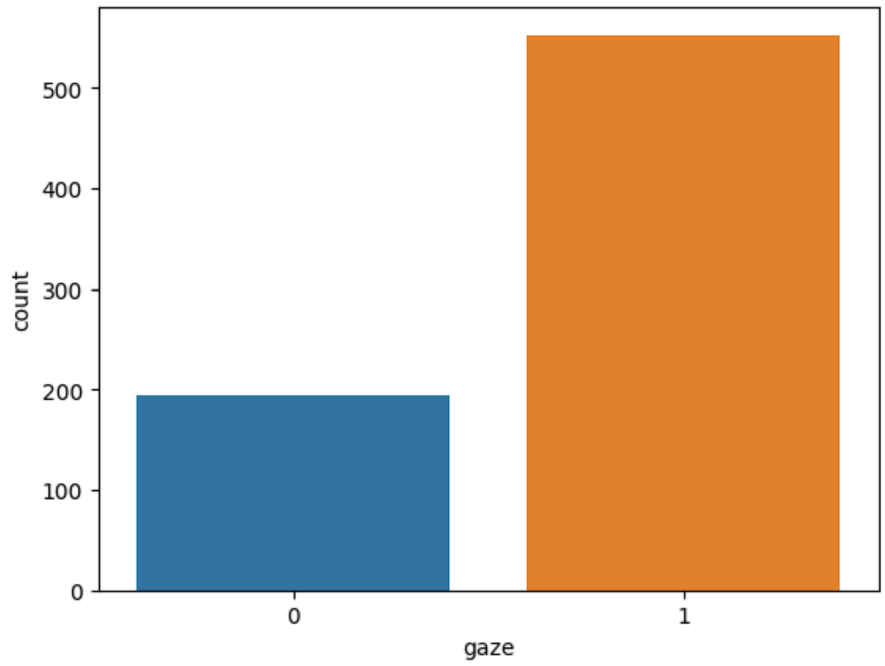
```
<Axes: >
```



b. Now, we can use a countplot and histogram to plot the frequency of values occurring for each column

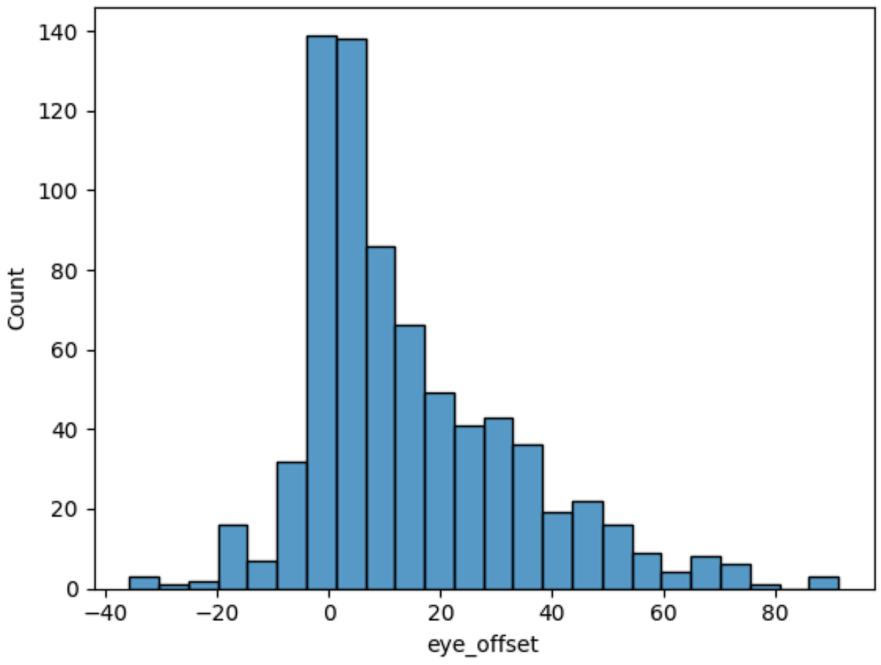
```
sns.countplot(x='gaze', data=gaze_df)
```

```
<Axes: xlabel='gaze', ylabel='count'>
```




```
sns.histplot(gaze_df['eye_offset'])
```

```
<Axes: xlabel='eye_offset', ylabel='Count'>
```



From the above data, we can infer that the candidates are looking at the camera often. And their offset is generally between -25 and 25.

c. Due to the high correlation between gaze and eye_offset, they must be rekated to each other. On closer analysis of the data we can notice that gaze is 1 whenever eye_offset is between -25 and 25.

```
gaze_df.loc[(gaze_df['eye_offset']>=-25) & (gaze_df['eye_offset']<=25)]['gaze'].value_counts()
```

```
1    553
Name: gaze, dtype: int64
```

```
gaze_df.loc[(gaze_df['eye_offset']<-25) | (gaze_df['eye_offset']>25)]['gaze'].value_counts()
```

```
0    194
Name: gaze, dtype: int64
```

So when we are performing clustering we can drop the eye_offset column as it is a redundant column.

```
gaze_df=gaze_df.drop(['eye_offset'],axis=1)
gaze_df.head()
```

	gaze
0	1
1	1
2	1
3	1
4	1

The frequency of gaze=1 is high, which shows that most of the students are looking directly at the screen. This is also an important measure of their communication skills as it shows how confident they are with their own ability and their prepared transcript.

Transcript Data:

Understanding the Data

a. As, the transcript_data folder was in the same directory as the jupyter notebook used, this command loads the 1.csv file into the transcript_df dataframe using pandas' read_csv command.

```
transcript_df=pd.read_csv("transcript_data/1.csv")
```

b. We need to merge the transcript data for all the students, so to do this, we can take the help of a for loop. We can then load the data of each student in a temporary dataframe df3, and outer-merge it with transcript_df using pd.merge. As we have same columns in all the dataframes, outer merge will combine all the dataframes to give us our final transcript_df that contains the transcript data of the 10 students.

```
for i in range(2,11):
    df3=pd.read_csv("transcript_data/"+str(i)+".csv")
    transcript_df=pd.merge(transcript_df,df3,how='outer')
```

c. The head command returns the first 5 rows of the dataframe.

```
transcript_df.head()
```

	id	seek	start	end	text	tokens	temperature	avg_logprob	compression_ratio	no_speech_prob	positive	negative	neutral	confident	hesitant	conc
0	0	0	0.00	5.56	Hello, I am Jeffrey Shepherd and I am current...	[50364, 2425, 11, 286, 669, 1587, 84, 2938, 49...	0.0	-0.447179	1.651235	0.63588	0.580265	0.152281	0.267454	0.846701	0.845698	0.6358
1	1	0	5.56	9.60	IIM Coikode. I have completed my B.Tech in Bi...	[50642, 286, 6324, 3066, 1035, 1429, 13, 286, ...	0.0	-0.447179	1.651235	0.63588	0.550327	0.189263	0.260410	0.679283	0.733701	0.5441
2	2	0	9.60	14.48	Technology Kolkata, followed by my M.Tech fro...	[50844, 15037, 26137, 74, 3274, 11, 6263, 538,...	0.0	-0.447179	1.651235	0.63588	0.639860	0.111150	0.248990	0.902729	0.834620	0.7158
3	3	0	14.48	18.48	of three years in the regulatory affairs doma...	[51088, 295, 1045, 924, 294, 264, 18260, 17478...	0.0	-0.447179	1.651235	0.63588	0.441894	0.399186	0.158919	0.774308	0.813044	0.5224
4	4	0	18.48	23.28	as a medical writer in Ciro Klein Farm, Mumba...	[51288, 382, 257, 4625, 9936, 294, 383, 5182, ...	0.0	-0.447179	1.651235	0.63588	0.236254	0.532010	0.231735	0.286049	0.561375	0.3343

d. Using the shape command, we know that transcript_df has 174 rows and 18 columns.

```
transcript_df.shape
```

```
(174, 18)
```

e. We can get details of the dataframe using describe command.

```
transcript_df.describe()
```

	id	seek	start	end	temperature	avg_logprob	compression_ratio	no_speech_prob	positive	negative	neutral	confident
count	174.000000	174.000000	174.000000	174.000000	174.0	174.000000	174.000000	174.000000	174.000000	174.000000	174.000000	174.000000
mean	8.804598	3259.954023	42.967356	48.048161	0.0	-0.305174	1.607418	0.312377	0.649040	0.181337	0.169623	0.633645
std	6.041933	2751.098118	27.410972	27.229853	0.0	0.060732	0.147240	0.268419	0.226489	0.174729	0.100368	0.251203
min	0.000000	0.000000	0.000000	3.000000	0.0	-0.661634	0.733333	0.003822	0.010479	0.002975	0.004540	0.001164
25%	4.000000	0.000000	19.270000	24.400000	0.0	-0.331398	1.582979	0.082161	0.520933	0.067987	0.087058	0.482443
50%	8.000000	2872.000000	42.560000	48.090000	0.0	-0.291910	1.626052	0.233486	0.654415	0.139484	0.171088	0.700413
75%	13.000000	5440.000000	63.930000	69.520000	0.0	-0.268956	1.667732	0.543254	0.841149	0.242137	0.240698	0.821050
max	27.000000	8692.000000	97.840000	98.920000	0.0	-0.196253	1.789298	0.910383	0.992485	0.971378	0.498208	0.993897

Cleaning the Data

a. Using the columns command, we can find out all the columns of the dataframe.

```
transcript_df.columns
```

```
Index(['id', 'seek', 'start', 'end', 'text', 'tokens', 'temperature',
       'avg_logprob', 'compression_ratio', 'no_speech_prob', 'positive',
       'negative', 'neutral', 'confident', 'hesitant', 'concise',
       'enthusiastic', 'speech_speed'],
      dtype='object')
```

b. We do need the id, seek, start, end, text, tokens, temperature, avg_logprob and compression_ratio as they have no relevance in our data analysis. So we can drop these columns.

```
transcript_df=transcript_df.iloc[:,9:]
# Or transcript_df.drop(['id','seek','start','end','text','tokens','temperature','avg_logprob','compression_ratio'])
transcript_df
```

	no_speech_prob	positive	negative	neutral	confident	hesitant	concise	enthusiastic	speech_speed
0	0.635880	0.580265	0.152281	0.267454	0.846701	0.845698	0.635805	0.647783	2.517986
1	0.635880	0.550327	0.189263	0.260410	0.679283	0.733701	0.544145	0.417390	3.217822
2	0.635880	0.639860	0.111150	0.248990	0.902729	0.834620	0.715861	0.700062	2.868852
3	0.635880	0.441894	0.399186	0.158919	0.774308	0.813044	0.522462	0.279916	3.750000
4	0.635880	0.236254	0.532010	0.231735	0.286049	0.561375	0.334381	0.197305	3.541667
...	...	...	...	...	...	...	...	...	...
169	0.036832	0.737435	0.063301	0.199264	0.821343	0.204142	0.422417	0.254029	3.169014
170	0.036832	0.594038	0.206492	0.199470	0.455449	0.631635	0.221028	0.127612	2.884615
171	0.036832	0.587039	0.207191	0.205771	0.127398	0.436416	0.020206	0.275292	2.866242
172	0.101272	0.542674	0.259974	0.197352	0.539320	0.450221	0.284381	0.104140	2.571429
173	0.101272	0.963019	0.013739	0.023242	0.807010	0.303308	0.646336	0.479207	3.409091

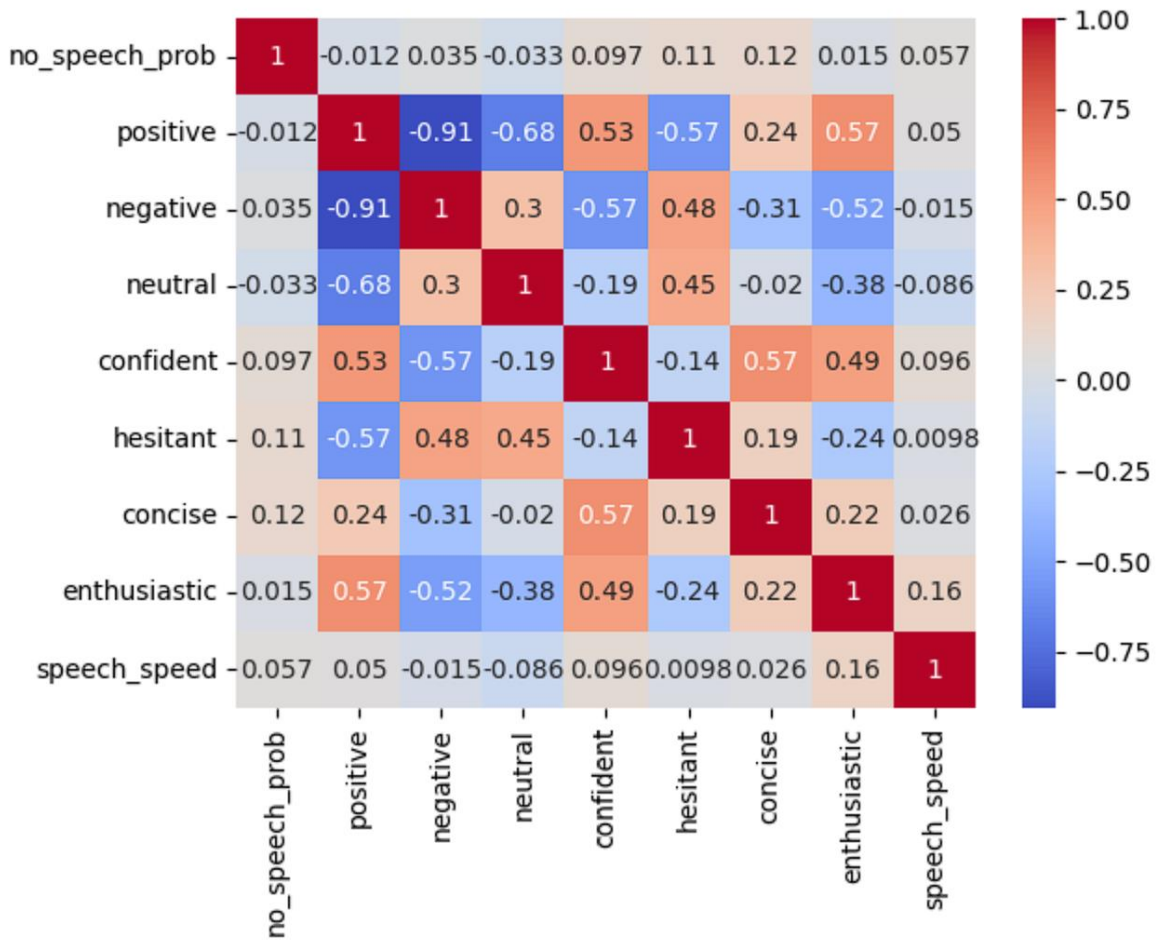
174 rows × 9 columns

Relationship Analysis

a. Using the sns library we can plot the heatmap of the correlation matrix, which shows how correlated each variable is with each other.

```
correlation_matrix = transcript_df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
```

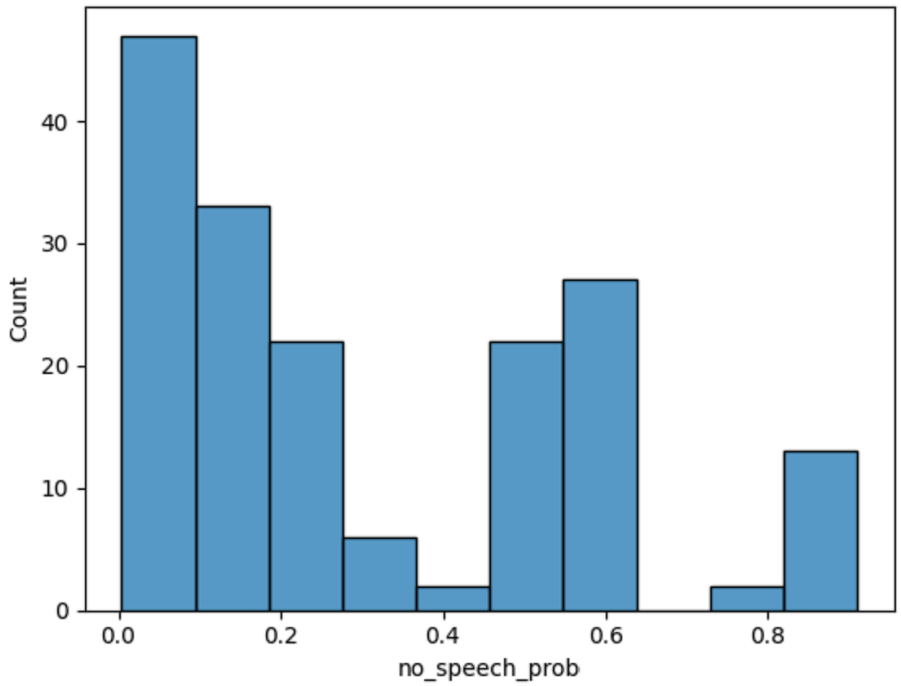
```
<Axes: >
```



b. Now, we can us plot histograms to check the frequency of values occurring for each column.

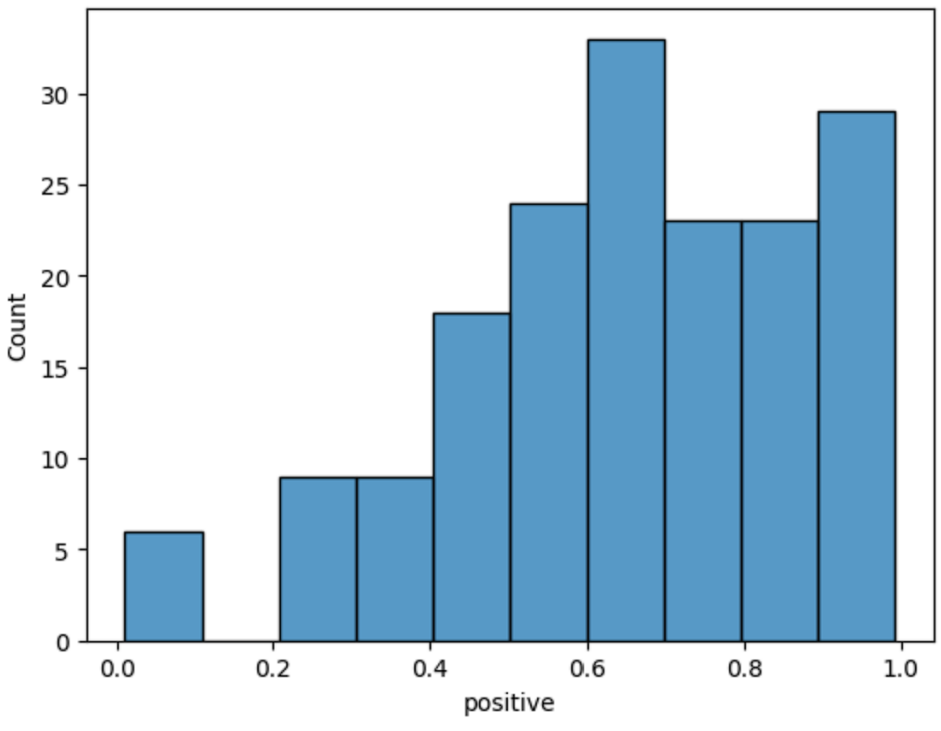
```
sns.histplot(transcript_df['no_speech_prob'],bins=10)
```

```
<Axes: xlabel='no_speech_prob', ylabel='Count'>
```



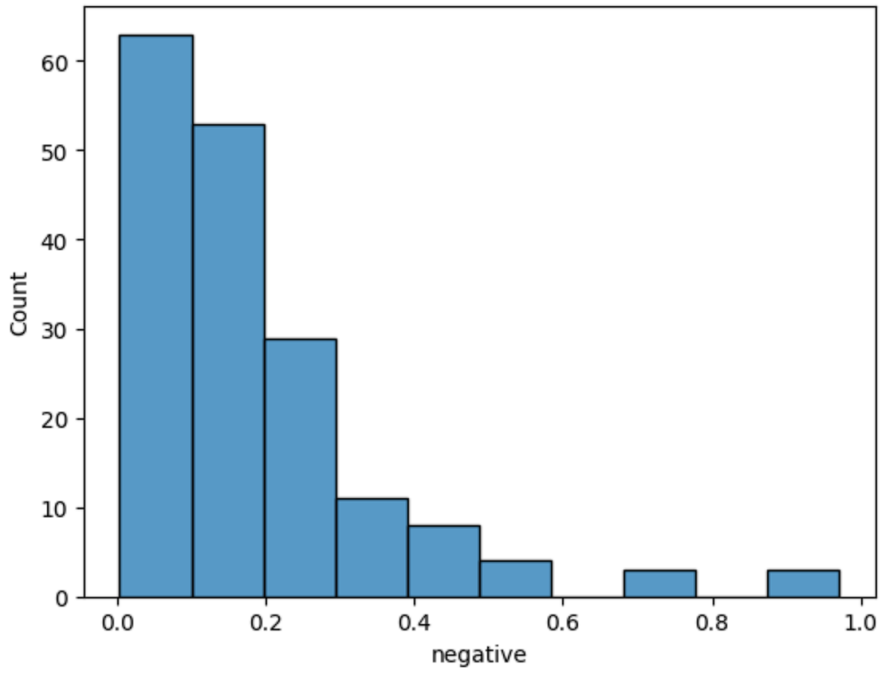

```
sns.histplot(transcript_df['positive'],bins=10)
```

```
<Axes: xlabel='positive', ylabel='Count'>
```



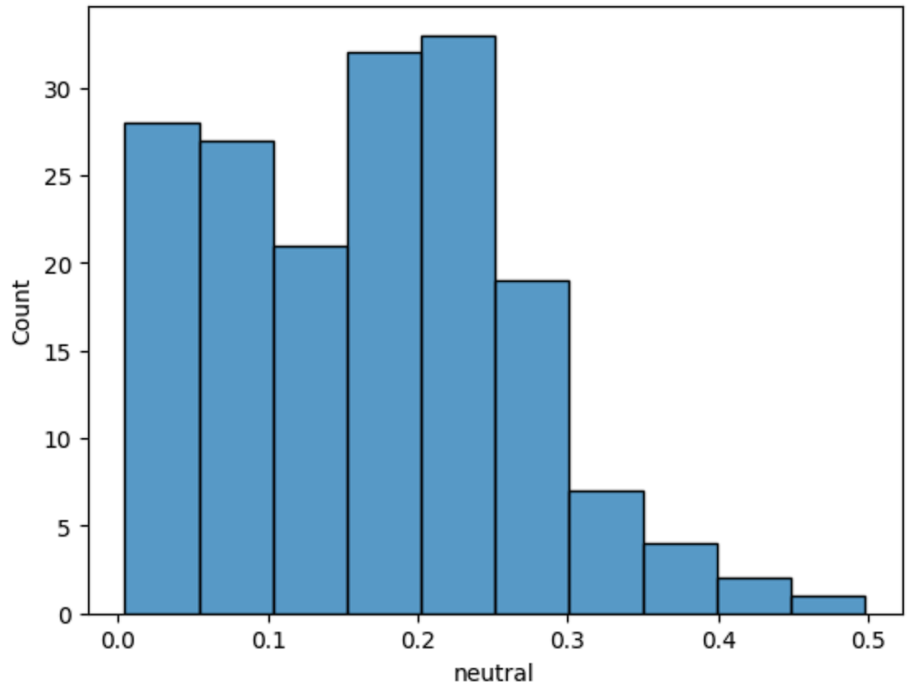
```
sns.histplot(transcript_df['negative'],bins=10)
```

```
<Axes: xlabel='negative', ylabel='Count'>
```



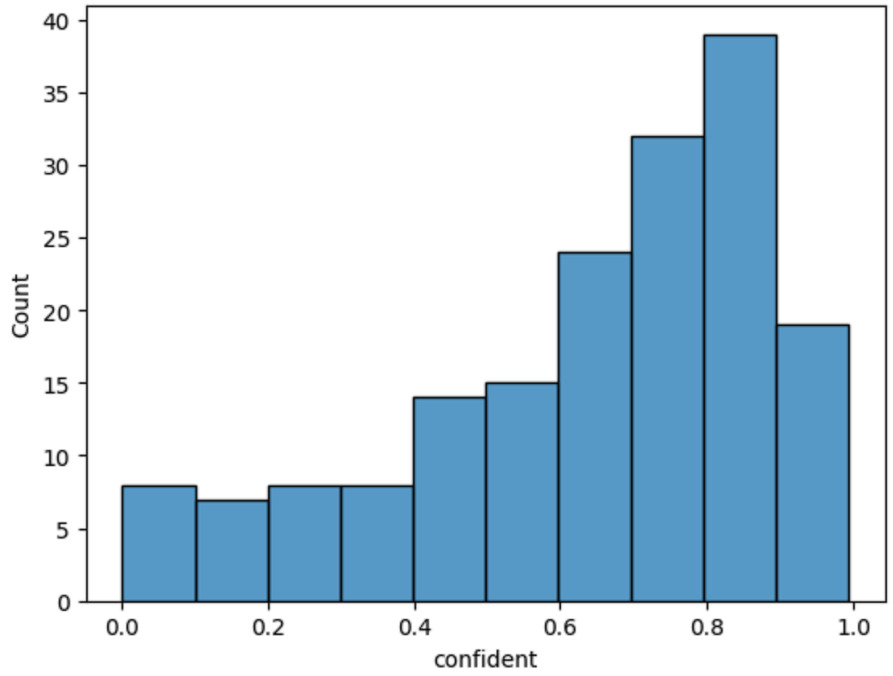

```
sns.histplot(transcript_df['neutral'],bins=10)
```

```
<Axes: xlabel='neutral', ylabel='Count'>
```



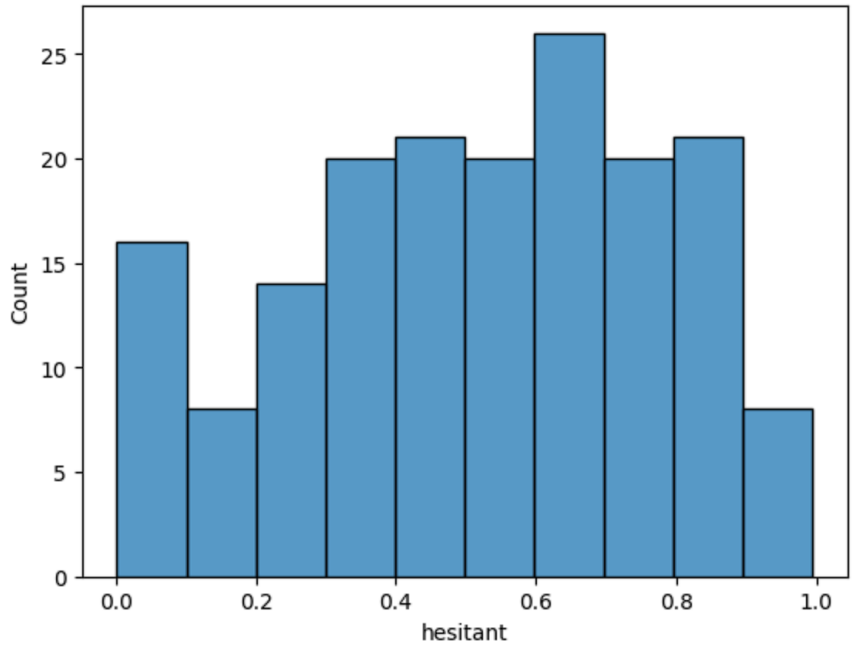
```
sns.histplot(transcript_df['confident'],bins=10)
```

```
<Axes: xlabel='confident', ylabel='Count'>
```



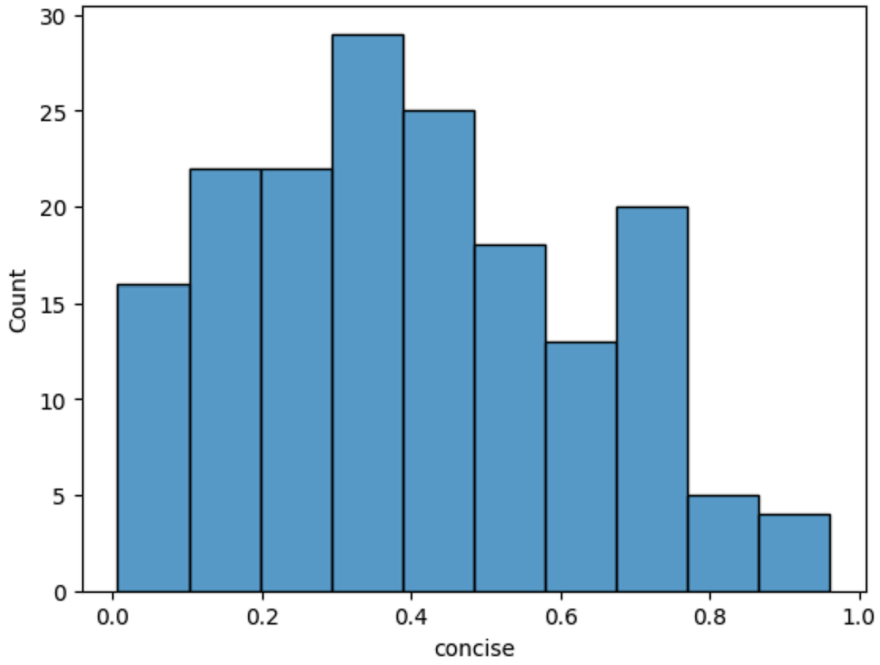

```
sns.histplot(transcript_df['hesitant'],bins=10)
```

```
<Axes: xlabel='hesitant', ylabel='Count'>
```



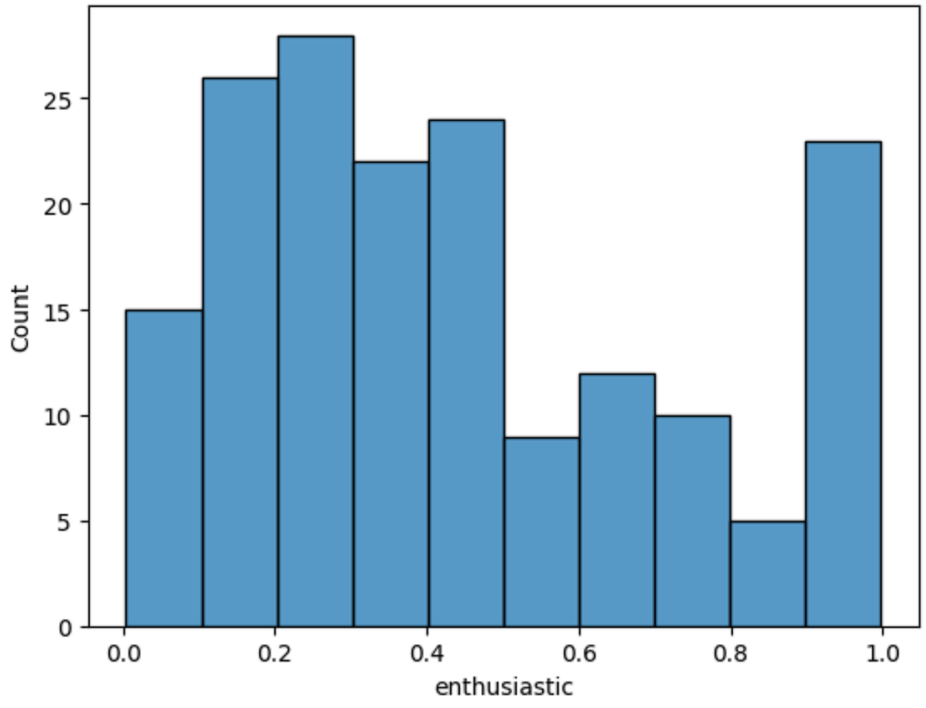
```
sns.histplot(transcript_df['concise'],bins=10)
```

```
<Axes: xlabel='concise', ylabel='Count'>
```



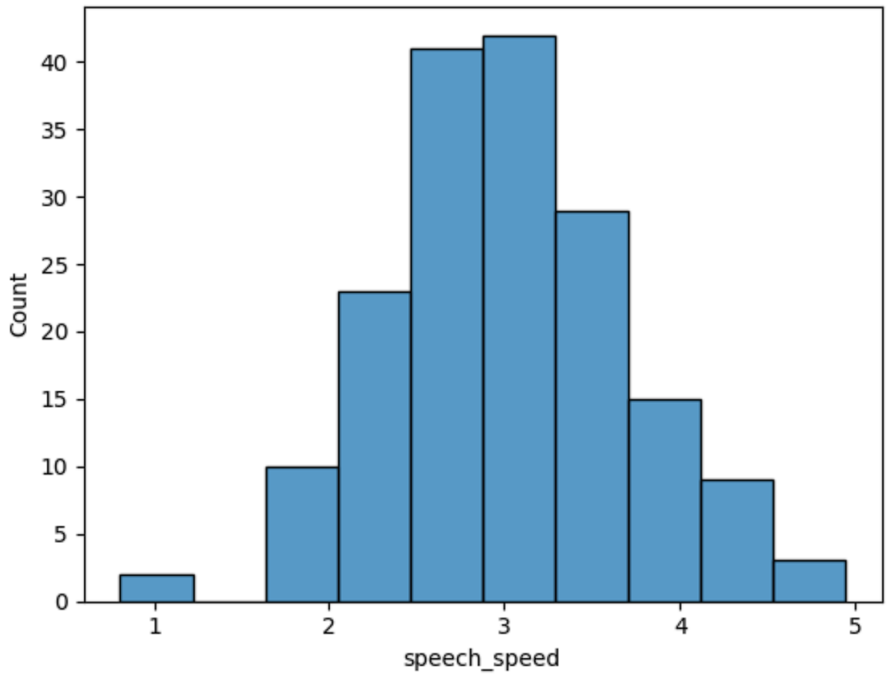

```
sns.histplot(transcript_df['enthusiastic'],bins=10)
```

```
<Axes: xlabel='enthusiastic', ylabel='Count'>
```



```
sns.histplot(transcript_df['speech_speed'],bins=10)
```

```
<Axes: xlabel='speech_speed', ylabel='Count'>
```



No speech is mostly low indicating there are very less pauses in the videos of the candidates. This is a good sign as it shows that the students are confident with the content they are speaking. Students are also highly positive, confident and enthusiastic. They also have an average level of neutrality, conciseness and speech speed. Looking at this data we can see that it describes the communication skills of the students in great detail. It is an important measure of the student's personality and overall soft skills. It tells us about the scripting and speaking skills of the students. This is very important as it shows how well a student can write reports and emails which are an important part of a job.

Prediction using Clustering

Now, that we have all the data let us try to predict if we need to take the student or not using an unsupervised learning method called clustering.

Unsupervised learning is a type of machine learning paradigm where the model learns patterns and relationships in data without being provided with explicit labels or target outputs. In other words, it involves training a model on a dataset that doesn't have predefined target variables.

Clustering algorithms group similar data points together based on their feature similarity. The aim is to discover natural groupings or clusters within the data.

As in our datasets, emotion and gaze have same number of rows, they can be merged together. So we are going to predict the values for student with two dataframes: one containing emotion and gaze, while the other containing transcript scores. As each student has multiple rows defining their emotions, gaze and transcript scores at different points of time in the video, we can take the average of the values that we get from clustering. Clustering will only give us 2 values: 0 and 1, as we have only 2 clusters. After taking the average of the values received for each student, if the value is closer to 0, then the student is not suitable for recruitment, whereas if the value is closer to 1 then the student is suitable for recruitment.

We will apply clustering separately for dataframe containing emotion and gaze, and the dataframe containing transcript scores.

1. The clustering algorithm has been written in the jupyter notebook file named Clustering. Now first we will need to use the pd.read_csv command to read the emotion, gaze and transcript data, and store it in emotion_df, gaze_df and transcript_df. The last row of gaze_df has been deleted, as for student 1 the number of rows of emotion.csv and gaze.csv differ by 1, so for sake of uniformity and as it will not affect the analysis and prediction much, the last row has been removed. We will also store the number of rows of each student's emotion and transcript scores to help us take average later.

```
# Loading the data
emotion_df=pd.read_csv("emotion_data/1/emotion.csv") # Loading emotion score of first student
gaze_df=pd.read_csv("emotion_data/1/gaze.csv") # Loading gaze score of first student
transcript_df=pd.read_csv("transcript_data/1.csv") # Loading transcript score of first student
gaze_df=gaze_df.drop(gaze_df.index[-1]) # Due to one row difference between emotion and gaze for 1st student and to keep unniformity
lengths1=[len(emotion_df.index)] # To keep track of number of rows of each student for emotion and gaze
lengths2=[len(transcript_df.index)] # To keep track of number of rows of each student for transcript score
```

2. Now using a for loop we can easily merge all 10 students' data into the dataframes.

```
for i in range(2,11):
    df1=pd.read_csv("emotion_data/"+str(i)+"/emotion.csv")
    df2=pd.read_csv("emotion_data/"+str(i)+"/gaze.csv")
    df3=pd.read_csv("transcript_data/"+str(i)+".csv")
    # Merging the dataframes
    emotion_df=pd.merge(emotion_df,df1,how='outer')
    gaze_df=pd.merge(gaze_df,df2,how='outer')
    transcript_df=pd.merge(transcript_df,df3,how='outer')
    lengths1.append(len(df1.index)) # Storing the number of rows of each student's emotion and gaze
    lengths2.append(len(df3.index)) # Storing the number of rows of each student's transcript score
```

3. As already seen in the EDA, we can remove the movie_id, image_seq and dominant_emotion columns from emotion_df, the movie_id, image_seq, blink and eye_offset columns from gaze_df, and ,the id, seek, start, end, text, tokens, temperature, avg_logprob and compression_ratio columns from the transcript_df dataframe.

```
# Dropping the unwanted and redundant columns
emotion_df=emotion_df.drop(['movie_id','image_seq','dominant_emotion'],axis=1)
gaze_df=gaze_df.drop(['movie_id','image_seq','blink','eye_offset'],axis=1)
transcript_df=transcript_df.iloc[:,9:]
```

4. Now, let us concatenate the emotion_df and gaze_df and assign it to result1_df, and assign transcript_data to result2_df.

```
result1_df=pd.concat([emotion_df,gaze_df],axis=1) # result1 has the concatenated dataframe of emotion_df and gaze_df
result2_df=transcript_df #result2 has the dataframe transcript_df
```

result1_df

	angry	disgust	fear	happy	sad	surprise	neutral	gaze
0	4.317350	5.942640e-04	2.879790	1.650350e+00	2.779980	0.600814	87.77110	1
1	53.225300	2.981640e+00	12.736800	1.523470e+00	1.051320	27.216800	1.26462	1
2	8.796510	2.946810e-02	2.968160	1.683150e+01	39.884600	0.279335	31.21050	1
3	9.453030	1.067780e-01	1.553080	2.093010e+01	3.503870	0.909426	63.54370	1
4	56.000200	4.152410e-06	0.162231	5.583580e+00	0.197026	12.807600	25.24940	1
...	...	...	...	...	...	...	...	...
742	21.623500	3.223740e-01	55.701200	1.837300e+00	14.471100	2.007100	4.03747	0
743	0.483833	8.153230e-05	83.415300	2.197600e+00	12.474100	0.059187	1.36993	1
744	0.175224	4.728190e-10	13.272400	1.959540e-09	63.701500	0.000002	22.85090	1
745	0.326095	2.007640e-05	1.177400	3.822260e-02	33.006200	0.011101	65.44100	1
746	0.041253	4.232110e-08	0.005674	1.281550e-02	0.843036	0.035985	99.06120	1

747 rows × 8 columns

result2_df

	no_speech_prob	positive	negative	neutral	confident	hesitant	concise	enthusiastic	speech_speed
0	0.635880	0.580265	0.152281	0.267454	0.846701	0.845698	0.635805	0.647783	2.517986
1	0.635880	0.550327	0.189263	0.260410	0.679283	0.733701	0.544145	0.417390	3.217822
2	0.635880	0.639860	0.111150	0.248990	0.902729	0.834620	0.715861	0.700062	2.868852
3	0.635880	0.441894	0.399186	0.158919	0.774308	0.813044	0.522462	0.279916	3.750000
4	0.635880	0.236254	0.532010	0.231735	0.286049	0.561375	0.334381	0.197305	3.541667
...	...	...	...	...	...	...	...	...	...
169	0.036832	0.737435	0.063301	0.199264	0.821343	0.204142	0.422417	0.254029	3.169014
170	0.036832	0.594038	0.206492	0.199470	0.455449	0.631635	0.221028	0.127612	2.884615
171	0.036832	0.587039	0.207191	0.205771	0.127398	0.436416	0.020206	0.275292	2.866242
172	0.101272	0.542674	0.259974	0.197352	0.539320	0.450221	0.284381	0.104140	2.571429
173	0.101272	0.963019	0.013739	0.023242	0.807010	0.303308	0.646336	0.479207	3.409091

174 rows × 9 columns

5. We now need to convert the dataframes to a numpy array to proceed further.

```
# to_numpy convert a dataframe to a numpy array
result1=result1_df.to_numpy()
result2=result2_df.to_numpy()
```

6. Now to perform clustering, we first need to normalise/scale the data with respect to the columns. Min-Max Scaling helps us to scale features to a certain range and also maintains the relative relationships between data points. All values will be between 0 and 1 after Min-Max Scaling.

   The Formula for Min-Max Scaling is:

   $$X\_scaled = (X - X\_min) / (X\_max - X\_min)$$

```
from sklearn.preprocessing import MinMaxScaler # This is required for Min-Max Scaling

scaler = MinMaxScaler()
model=scaler.fit(result1)
scaled_result1=model.transform(result1) # This is the scaled result_1

scaler = MinMaxScaler()
model=scaler.fit(result2)
scaled_result2=model.transform(result2)  # This is the scaled result_1
```

7. I am creating 2 temporary arrays that shows the values for each column that an ideal student would have. ideal1 corresponds to ideal case of scaled_result1 and ideal2 corresponds to ideal case of scaled_result2.

```
ideal1 = [0,0,0,1,0,0.5,1,1] # Array with ideal values of emotion and gaze
ideal2 = [0,1,0,1,1,0,1,1,0.5] # Array with ideal values of transcript scores
```

8. Now there are multiple types of clustering techniques out of which I have tried out three: K-Means (Centroid Model), Gaussian Mixture Model (Distribution Model), and Agglomerative Clustering (Connectivity Model). To find out their performance we can use the Davies-Bouldin Index.

   Davies-Bouldin Index: It is a metric used to evaluate the clustering performance of a dataset. It quantifies the "average similarity" between each cluster's data points and the centroid of that cluster, while also considering the "average dissimilarity" to the centroids of other clusters. In simple terms, it measures how well-separated the clusters are.

```
from sklearn.metrics import davies_bouldin_score # To analyse the perfomance of the clustering algorithms
```

9. **K-Means:**

   K-Means partitions data into K clusters where each data point belongs to the cluster with the nearest mean. It iteratively assigns points to clusters and updates cluster centroids until they converge.

a. First we need to import K-Means function from scikit learn.

```
from sklearn.cluster import KMeans
```

b. In K-Means, the parameters: random_state has been initialised to 0 so that centroids chosen remain the same every time the code is run, and, n_init is auto, so the algorithm automatically decides the number of times to run with different initialisations and choose the one with lowest inertia. Inertia is a measure of how far the points within a cluster are from the centroid of that cluster.

   We are also going to check the prediction for the ideal case, to make sure that it belongs to cluster 1.

```
# random_state has been initialised to 0 so that centroids chosen remain the same everytime the code is run
# n_init is the number of times the algorithm will be run with different initialisations and choose the one with lowest inertia
kmeans = KMeans(n_clusters=2, random_state=0, n_init="auto")
kmeans.fit(scaled_result1) # Computes K-Means Clustering for scaled_result1
kmeans.predict([ideal1]) # To check if ideal case is giving 0 or 1
```

```
array([1])
```

c. Ideal value belongs to cluster 1. Now we can calculate the Davies-Bouldin Index for this prediction. We can also find the prediction scores of each student obtained by using K-Means and take their average with the help of the lengths1 array which contains the number of rows of data of each candidate, and store it in ans1_kmeans.

```
predictions=kmeans.labels_ # These are the cluster labels for each row, by K-Means
db_index_kmeans_1 = davies_bouldin_score(scaled_result1, predictions) # Stores value of DB Index when using K-Means for scaled_result1
ans1_kmeans=[] # The final prediction scores of each student using emotion and gaze and K-Means is stored in this list
i=0
for l in lengths1: # Using lengths1 to help in taking average of all rows that contribute to each student
    ans1_kmeans.append(np.average(predictions[i:i+l]))
    i+=l
ans1_kmeans
```

```
[0.3563218390804598,
 0.28735632183908044,
 0.31,
 0.9696969696969697,
 1.0,
 1.0,
 0.11494252873563218,
 0.7311827956989247,
 0.32558139534883723,
 0.12222222222222222]
```

d. Similarly doing the above for scaled_result2.

```
# Similarly doing for scaled_result2
kmeans = KMeans(n_clusters=2, random_state=0, n_init="auto")
kmeans.fit(scaled_result2)
kmeans.predict([ideal2])
```

```
array([0])
```

e. Ideal value is in cluster 0. For the sake of uniformity, I want to make sure ideal value and values close to it to always belong in cluster 1. Since we have only 2 clusters, simply doing predictions=1-predictions, will swap the clusters.

```
# Since in this case ideal values are belonging to cluster-0, we need to swap the clusters
# This is to make sure that ideal and good values always remain in cluster 1 for uniformity
predictions=kmeans.labels_
predictions
```

```
array([0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0,
       1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0,
       1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0,
       0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1,
       1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0])
```

```
predictions=1-predictions # Simple way to swap the clusters if only 2 clusters are present
predictions # Now ideal case will belong to cluster 1
```

```
array([1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
       0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1,
       0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1,
       0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,
       0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1])
```

f. Now, we can calculate DB Index, and calculate the average of predicted values, with the help of lengths2 array, and store them in ans2_kmeans.

```
db_index_kmeans_2 = davies_bouldin_score(scaled_result2, predictions) # DB Index when using K-Means for scaled_result2
ans2_kmeans=[] # The final prediction scores of each student using transcript scores and K-Means is stored in this list
i=0
for l in lengths2: # Using lengths2 to help in taking average of all rows that contribute to each student
    ans2_kmeans.append(np.average(predictions[i:i+l]))
    i+=l
ans2_kmeans
```

```
[0.6666666666666666,
 0.5263157894736842,
 0.35714285714285715,
 0.5263157894736842,
 0.47058823529411764,
 0.4444444444444444,
 0.46153846153846156,
 0.5625,
 0.4444444444444444,
 0.35294117647058826]
```

10. **Gaussian Mixture Model:**

Gaussian Mixture Model models the distribution of data points as a mixture of multiple Gaussian distributions. It maximizes the likelihood of the observed data with respect to a mixture of Gaussian distributions.

a. First we need to import Gaussian Mixture Model function from scikit learn.

```
from sklearn.mixture import GaussianMixture
```

b. N_components parameter tells the algorithm how many clusters to create. Here also we check the predictions with ideal array to make sure it belongs to cluster 1.

```
# n_components tells the algorithm how many clusters to make
gaussian_model = GaussianMixture(n_components=2)
gaussian_model.fit(scaled_result1) # train the data
gaussian_result = gaussian_model.predict(scaled_result1) # assign each data point to a cluster
gaussian_model.predict([ideal1]) # To check if ideal case is giving 0 or 1
```

```
array([1], dtype=int64)
```

c. Ideal value belongs to cluster 1. Now we can calculate the Davies-Bouldin Index for this prediction. We can also find the prediction scores of each student obtained by using GMM and take their average with help of the lengths1 array which stored the number of rows for each candidate, and store it in ans1_gaussian.

```
db_index_gaussian_1 = davies_bouldin_score(scaled_result1, gaussian_result) # DB Index when using GMM for scaled_result1
ans1_gaussian=[]  # The final prediction scores of each student using emotion, gaze and GMM is stored in this list
i=0
for l in lengths1:
    ans1_gaussian.append(np.average(gaussian_result[i:i+l]))
    i+=l
ans1_gaussian
```

```
[0.6206896551724138,
 0.6091954022988506,
 0.45,
 0.7878787878787878,
 1.0,
 1.0,
 0.7816091954022989,
 0.946236559139785,
 0.9651162790697675,
 0.7333333333333333]
```

d. Similarly doing for scaled_result2.

```
# Similarly doing for scaled_result2
gaussian_model = GaussianMixture(n_components=2)
gaussian_model.fit(scaled_result2) # train the data
gaussian_result = gaussian_model.predict(scaled_result2) # assign each data point to a cluster
gaussian_model.predict([ideal2])
```

```
array([1], dtype=int64)
```

g. Ideal array belongs to cluster 1. Now, we can calculate DB Index, and calculate the average of predicted values, with the help of lengths2 array, and store them in ans2_gaussian.

```
db_index_gaussian_2 = davies_bouldin_score(scaled_result2, gaussian_result) # DB Index when using GMM for scaled_result2
ans2_gaussian=[] # The final prediction scores of each student using transcript score and GMM is stored in this list
i=0
for l in lengths2:
    ans2_gaussian.append(np.average(gaussian_result[i:i+l]))
    i+=l
ans2_gaussian
```

```
[0.16666666666666666,
 0.3157894736842105,
 0.5714285714285714,
 0.42105263157894735,
 0.47058823529411764,
 0.3333333333333333,
 0.38461538461538464,
 0.5,
 0.4444444444444444,
 0.5882352941176471]
```

11. Agglomerative Clustering:

Agglomerative Clustering hierarchically clusters data by successively merging or agglomerating points and clusters. It begins with each data point as a single cluster and merges clusters iteratively based on a distance metric.

a. First we need to import Agglomerative Clustering function from scikit learn.

```
from sklearn.cluster import AgglomerativeClustering
```

b. N_clusters parameter tells the algorithm how many clusters to create. Here also we check the predictions with ideal array to make sure it belongs to cluster 1, but agglomerative clustering only predicts with an aggregate data, not individual data. So to do this I have appended the scaled_result1 to the ideal row and stored it in scaled_result1_with_ideal. The first row of scaled_result1 contains the ideal array, so the first value of the agglomeration_result is the cluster of the ideal array.

```
# n_clusters tells the algorithm how many clusters are there
agglomerative_model = AgglomerativeClustering(n_clusters=2)
scaled_result1_with_ideal=np.row_stack([ideal1,scaled_result1]) # Including the ideal case to check whcih cluster it belongs
agglomerative_result = agglomerative_model.fit_predict(scaled_result1_with_ideal) # Performing prediction
agglomerative_result[0]
```

```
0
```

c. Ideal case belongs to cluster 0. So, we predict clusters for scaled_result1 and then to swap the clusters we do agglomeration_result=1-agglomeration_result.

```
# As ideal case belongs to cluster 0, clusters have to be swapped after prediction
agglomerative_result = agglomerative_model.fit_predict(scaled_result1) # Predicting for the scaled_result1
agglomerative_result=1-agglomerative_result # Now ideal case will belong to cluster 1
```

d. Now we can calculate the Davies-Bouldin Index for this prediction. We can also find the prediction scores of each student obtained by using Agglomerative Clustering and take their average with help of the lengths1 array which stored the number of rows for each candidate, and store it in ans1_agglomerative.

```
db_index_agglomerative_1 = davies_bouldin_score(scaled_result1, agglomerative_result) # DB Index when using Agglomerative for scaled_result1
ans1_agglomerative=[]  # The final prediction scores of each student using emotion, gaze and Agglomerative is stored in this list
i=0
for l in lengths1:
    ans1_agglomerative.append(np.average(agglomerative_result[i:i+l]))
    i+=l
ans1_agglomerative
```

```
[0.6206896551724138,
 0.6091954022988506,
 0.45,
 0.7878787878787878,
 1.0,
 1.0,
 0.7816091954022989,
 0.946236559139785,
 0.9651162790697675,
 0.7333333333333333]
```

e. Similarly doing the above for scaled_result2.

```
# Similarly doing for scaled_result2
agglomerative_model = AgglomerativeClustering(n_clusters=2)
scaled_result2_with_ideal=np.row_stack([ideal2,scaled_result2]) # Including the ideal case to check whcih cluster it belongs
agglomerative_result = agglomerative_model.fit_predict(scaled_result2_with_ideal) # Performing prediction
agglomerative_result[0]
```

```
0
```

f. As ideal case belongs to cluster 0, we need to swap the clusters.

```
# As ideal case belongs to cluster 0, clusters have to be swapped after prediction
agglomerative_result = agglomerative_model.fit_predict(scaled_result2) # Predicting for the scaled_result1
agglomerative_result=1-agglomerative_result # Now ideal case will belong to cluster 1
```

g. Now, we can calculate DB Index, and calculate the average of predicted values, with the help of lengths2 array, and store them in ans2_agglomerative.

```
db_index_agglomerative_2 = davies_bouldin_score(scaled_result2, agglomerative_result)  # DB Index when using Agglomerative for scaled_result2
ans2_agglomerative=[] # The final prediction scores of each student using transcript score and Agglomerative is stored in this list
i=0
for l in lengths2:
    ans2_agglomerative.append(np.average(agglomerative_result[i:i+l]))
    i+=l
ans2_agglomerative
```

```
[0.8888888888888888,
 0.8947368421052632,
 0.6785714285714286,
 0.6842105263157895,
 0.7058823529411765,
 0.8333333333333334,
 0.7692307692307693,
 0.5625,
 0.8888888888888888,
 0.7647058823529411]
```

12. Now to choose the best clustering algorithm, we will look at the Davis-Bouldin Index for each algorithm. The lower the DB index, the better the clustering. Lower DB Index indicates that the clusters are well-separated and that the data points within each cluster are close to the centroid of that cluster, while also being far from the centroids of other clusters.

```
db_index_kmeans_1
```

```
1.2723514577449737
```

```
db_index_kmeans_2
```

```
1.7255051146526268
```

```
db_index_gaussian_1
```

```
1.1430053187993996
```

```
db_index_gaussian_2
```

```
1.8186697617003467
```

```
db_index_agglomerative_1
```

```
1.1430053187993996
```

```
db_index_agglomerative_2
```

```
1.5727742568753424
```

It is observed that agglomerative has lowest DB index for both datasets. So it is the best performing clustering algorithm for our datasets.

13. Now we can aggregate the results of ans1_agglomerative and ans2_agglomerative into a final_ans. In this array, the first element shows how suitable the student is for being recruited based on their emotion and gaze scores. The second element shows how suitable the student is for being recruited based on their transcript score.

```
final_ans = [[x,y] for x, y in zip(ans1_agglomerative, ans2_agglomerative)] # Stores the final predicted recruitability of all 10 students
# First element shows the recruitability of student based on their emotion and gaze
# Second element shows the recruitability of student based on their transcript score
# If value is closer to 1, then student is more suitable for recruitment
final_ans
```

```
[[0.6206896551724138, 0.8888888888888888],
 [0.6091954022988506, 0.8947368421052632],
 [0.45, 0.6785714285714286],
 [0.7878787878787878, 0.6842105263157895],
 [1.0, 0.7058823529411765],
 [1.0, 0.8333333333333334],
 [0.7816091954022989, 0.7692307692307693],
 [0.946236559139785, 0.5625],
 [0.9651162790697675, 0.8888888888888888],
 [0.7333333333333333, 0.7647058823529411]]
```

Conclusion:

If the values are closer to 1, then the student is more suitable for recruitment. If the values are closer to 0, then the student is less suitable for recruitment. Using the values obtained in final_ans, the recruiter can see the values obtained after applying clustering on emotion, gaze and transcript scores, and choose the best students to be recruited.

A higher value in the ans1_agglomerative shows that the student has good emotions and is mostly looking directly at the screen (gaze value is 1). A higher value in the ans2_agglomerative shows that the student has good scripting and speaking skills, and also has a decent speech speed, which allows people to understand what is being conveyed easily.

A student having an overall value close to 1 indicates that he/she has positive emotions, a gaze value close to 1 and a good transcript score. A student showing positive emotions and good speaking skills should be a good recruit as he/she can convey their thoughts well and please the people they interact, with their amicable nature (due to their good emotions). Having good emotions and proper emotional control also signifies that the student has a good body language. They will also speak directly to the person in front showing a sense of confidence (due to a good gaze value). A student's personality and overall soft skills is an important factor to be considered for recruitment. Thus, from this prediction we are analysing the communication skills of a candidate and seeing if they are fit to be recruited. A student must have a good mix of all qualities to be a suitable candidate for recruitment.