DATA MINING

LAB ASSIGNMENT

NAME: MAHNOOR ASIF

REG NO: FA22-BCS-150

SEC: BCS-5D

DATE: 24 NOV, 2024

Table of Content:

TASK 1:

VISUALIZATIONS:

Pair plot and scatter plot :

Purpose of the Visualization:

Pairplot:

- Pairplot is a grid like structure of the scatter plots.
- On the diagonal of the grid, it typically shows histograms or density plots of individual features to give an overview of their distributions.
- Pairplot shows relationship between pair of features in the dataset.
- Its usefull when we have dataset with many columns and we have to find relationship between them.
- It provides relationships among multiple numerical features and their distributions.

Scatter Plot:

- A scatter plot focuses on the relationship between two specific numerical features.
- The x-axis and y-axis represent the two features, while points in the plot represent individual data instances

Need in Machine Learning:

- Pairplots and scatter plots help identify whether features are correlated
- They also reveal trends, such as whether a relationship between two features is linear, nonlinear, or non-existent.
- Overlapping classes might may lead to classification errors, while distinct clusters indicate better model performance.
- Pairplots also highlights unusual data points (outliers), which can affect model performance. Removing or addressing them can improve results.
- Scatter plots are particularly useful for visually identifying outliers when comparing two features.

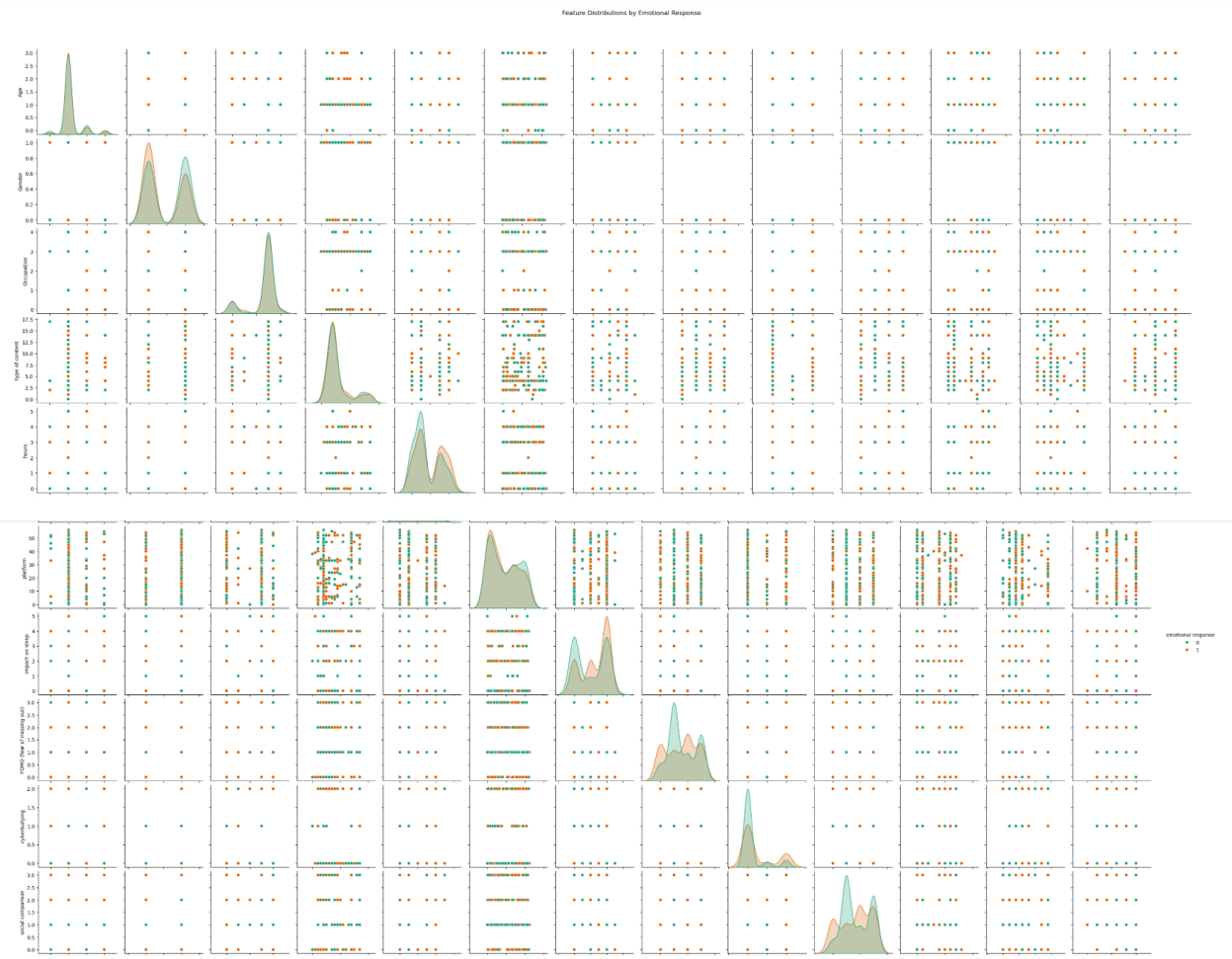
In naïve bayes:

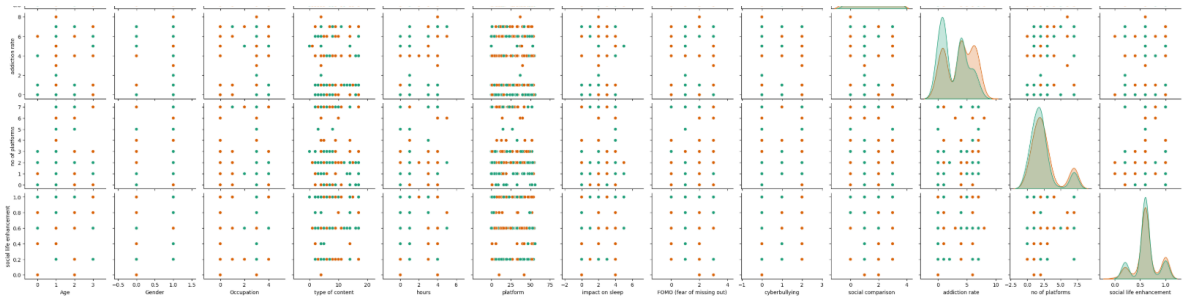
- Naive Bayes works by assuming that all features are **independent** of each other when we know the class. the value of one feature doesn't affect another feature if we know which class the data belongs to.
- When features are dependent, the model might wrongly treat them as separate, combining their effects inappropriately and leading to inaccurate predictions. This can make the model less reliable, as it overestimates the importance of those related features.

Code:

```
[242]: # Plot Pairplot to visualize feature distributions based on 'emotional response'
sns.pairplot(data, hue='emotional response', palette='Dark2')
plt.suptitle("Feature Distributions by Emotional Response", y=1.02)
plt.show()
output_file_path = 'pairplot and scatterplot.png'  # Specify your desired output path
data.to_csv(output_file_path, index=False)
```

output:





<Figure size 640x480 with 0 Axes>

Correlation Heatmap :

Purpose of the Visualization:

A correlation heatmap visually represents the relationships between multiple variables in a dataset, displaying the degree of correlation between each pair of variables. The color intensity shows the strength and direction of the correlation, with positive correlations shown in one color (e.g., blue) and negative correlations in another (e.g., red).

- A value close to +1 indicates a strong positive correlation,
- A value near -1 indicates a strong negative correlation
- 0 indicates no correlation.

Need in Machine Learning:

- the heatmap plays a key role in data exploration by highlighting relationships between variables.
- In machine learning, understanding the relationships between features is crucial for several reasons.
- A correlation heatmap helps us identify patterns and relationships in the data, which can guide important decisions. First, it aids in feature selection by showing us which features are highly correlated, allowing us to eliminate redundant ones and reduce the dimensionality of the dataset. This makes the model more efficient and easier to interpret. Second, a heatmap helps avoid multicollinearity, which occurs when features are highly correlated with each other.
- This insight can lead to better feature engineering, where we can create new features or apply transformations that help improve the model.

In naïve bayes:

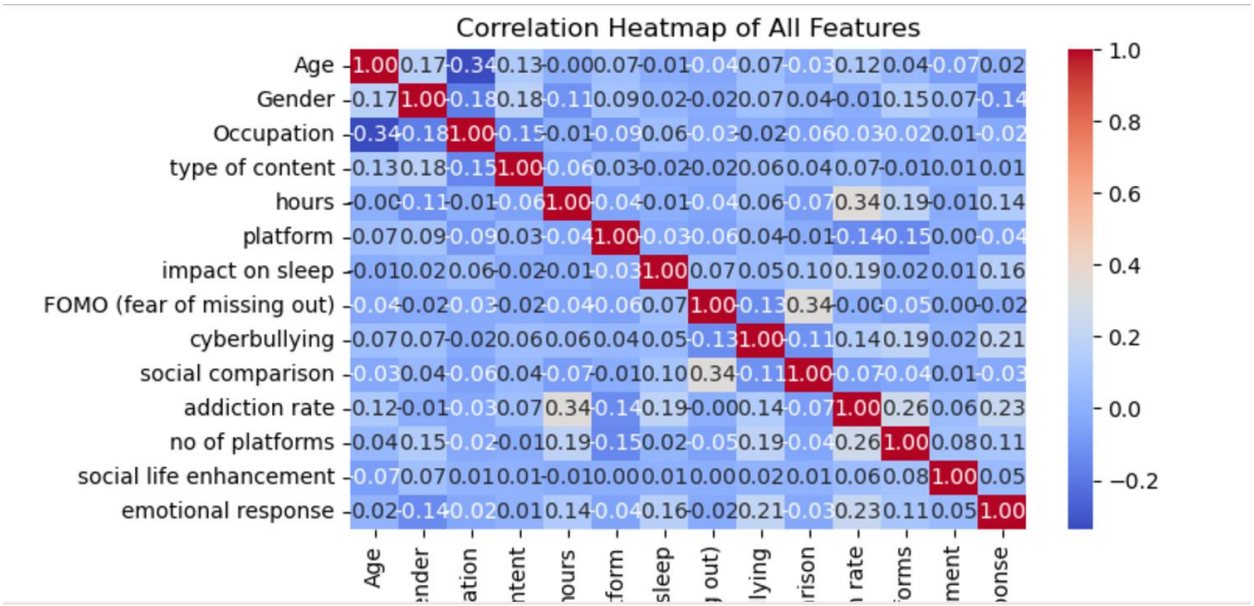
- For a Naive Bayes model, which assumes that features don't affect each other, a correlation heatmap shows if features are related.
- If features are too closely related, it breaks the model's assumption and can impact model accuracy.

Code:

```
[265]: import seaborn as sns
       import matplotlib.pyplot as plt
       correlation_matrix = data_encoded.corr()

       # Plot the correlation heatmap
       plt.figure(figsize=(7, 4))  # Adjust the figure size as needed
       sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
       plt.title("Correlation Heatmap of All Features")
       plt.show()
```

Output:



Key Observations from the Heatmap:

Strong Positive Correlations:

- FOMO (fear of missing out) and social life enhancement show a moderate positive correlation (0.34), indicating that people who experience higher FOMO might also report enhanced social life.
- Addiction rate and FOMO also have a positive correlation (0.34), suggesting that people who feel a higher sense of FOMO might also report higher addiction rates to social media.

Weak or No Correlation:

Most of the features, like gender, occupation, and type of content, show low correlations with each other, with values close to 0, meaning they are largely independent of one another in this dataset.

Negative Correlations:

Impact on sleep and hours spent on social media show a small negative correlation (-0.04), indicating that more hours spent on social media slightly correlates with reduced sleep impact.

Diagonal of the Heatmap:

The diagonal always shows a value of 1.0, which is the correlation of a feature with itself.

Box Plot:

Purpose of the Visualization:

A box plot is a simple graph that shows how the values in your data are spread out. It displays the smallest value, the middle value (median), and the largest value, along with any outliers that are far from the rest of the data.

Need in Machine Learning:

In machine learning, box plots help us understand how the data is spread. They show if there are any outliers (values that are much higher or lower than most of the data). They also show where most of the data is centered. This helps us know if we need to clean the data, change it, or remove some features to make the model work better.

In Naive Bayes:

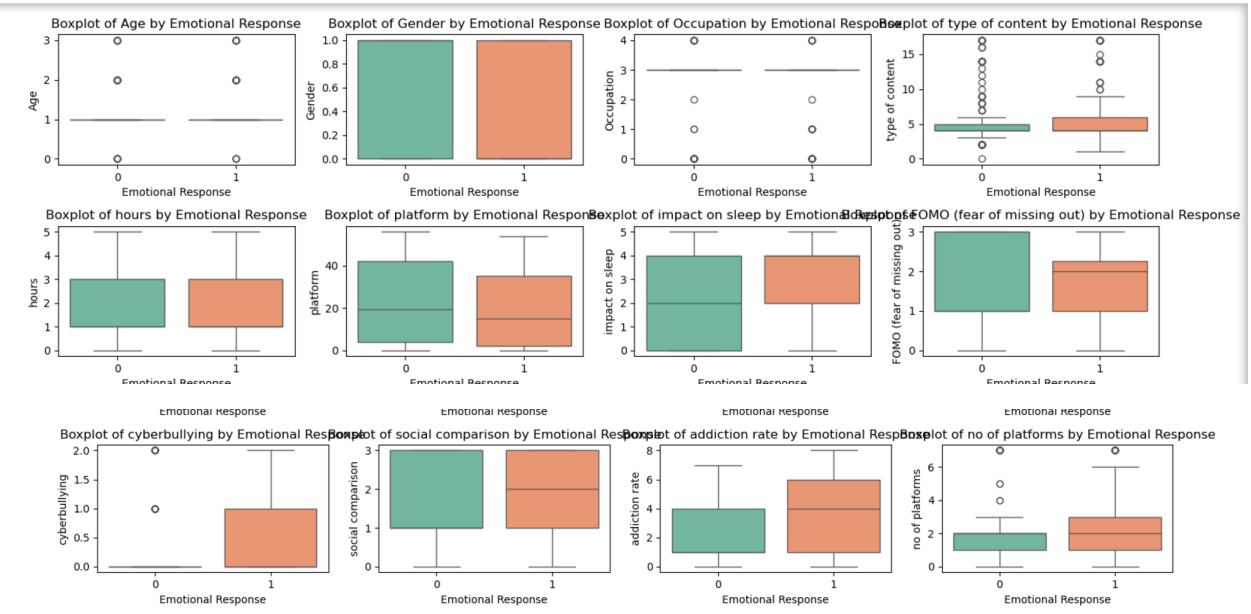
For a Naive Bayes model, which expects the features to follow a certain pattern (like a normal distribution), a box plot can show if the data fits that pattern. If the data is too spread out or has outliers, the model might not work well. The box plot helps us spot these issues and decide if we need to adjust the data or model to get better results.

Code:

```
[277]: import matplotlib.pyplot as plt
import seaborn as sns
all_columns = data.columns
plt.figure(figsize=(15, 10))

# Loop through all columns (features) and create boxplots
for idx, feature in enumerate(all_columns):
    plt.subplot(4, 4, idx + 1)  # Adjust the grid layout to fit all features (adjust rows and columns)
    sns.boxplot(x='emotional response', y=feature, data=data, palette='Set2')
    plt.title(f"Boxplot of {feature} by Emotional Response")
    plt.xlabel("Emotional Response")
    plt.ylabel(feature)
plt.tight_layout()
plt.savefig('Boxplots_all_features_by_Emotional_Response.png')
plt.show()
```

output:



Key Observations from the Boxplot:

- **Age by Emotional Response:**

  There's not much variation in age between the two categories of emotional response, as the boxplots are quite narrow. This might mean that age has little influence on emotional response.

- **Gender by Emotional Response:**

  The two boxplots look almost identical. This suggests gender does not significantly differentiate emotional responses.

- **Occupation and Type of Content by Emotional Response:**
  For Occupation, the data points seem sparse, with outliers present. This might mean no clear pattern exists for occupation's influence.

  For Type of Content, there's a wider spread in one group (likely indicating more diverse preferences) compared to the other.

- **Hours on Social Media by Emotional Response:**

  Both categories spend roughly similar amounts of time on social media, but the medians and spreads suggest slight variations.

- **Platforms Used by Emotional Response:**

  The wide range in values could indicate that people with a stronger emotional response engage on more platforms overall.

- **Impact on Sleep by Emotional Response:**

  This plot shows a difference, where one group might experience greater sleep impact than the other.

- **FOMO by Emotional Response:**

  People in one group (likely emotionally affected) seem to report higher levels of FOMO compared to the other.

- **Cyberbullying by Emotional Response:**

  Emotional response appears more prevalent in individuals who have experienced cyberbullying, as seen by the higher values for one group.

- **Social Comparison by Emotional Response:**

  There's a noticeable difference in social comparison levels, with one group engaging more in comparisons.

- **Addiction Rate and Number of Platforms by Emotional Response:**

  Both metrics show a clear distinction, where individuals with an emotional response engage with more platforms and report higher addiction levels.

Count Plot:

Purpose of the Visualization:

A count plot is a simple bar chart that shows how many times each category appears in your data. It's an easy way to see patterns or check if some categories have more or fewer data points than others.

Need in Machine Learning:

Count plots are helpful when working with machine learning because they show how balanced your data is. Here's why they matter:

Spotting Class Imbalance: If one category (like "Yes" vs. "No") has way more data points, your model might become biased toward the larger group. Count plots help you catch this early.

In Naive Bayes:

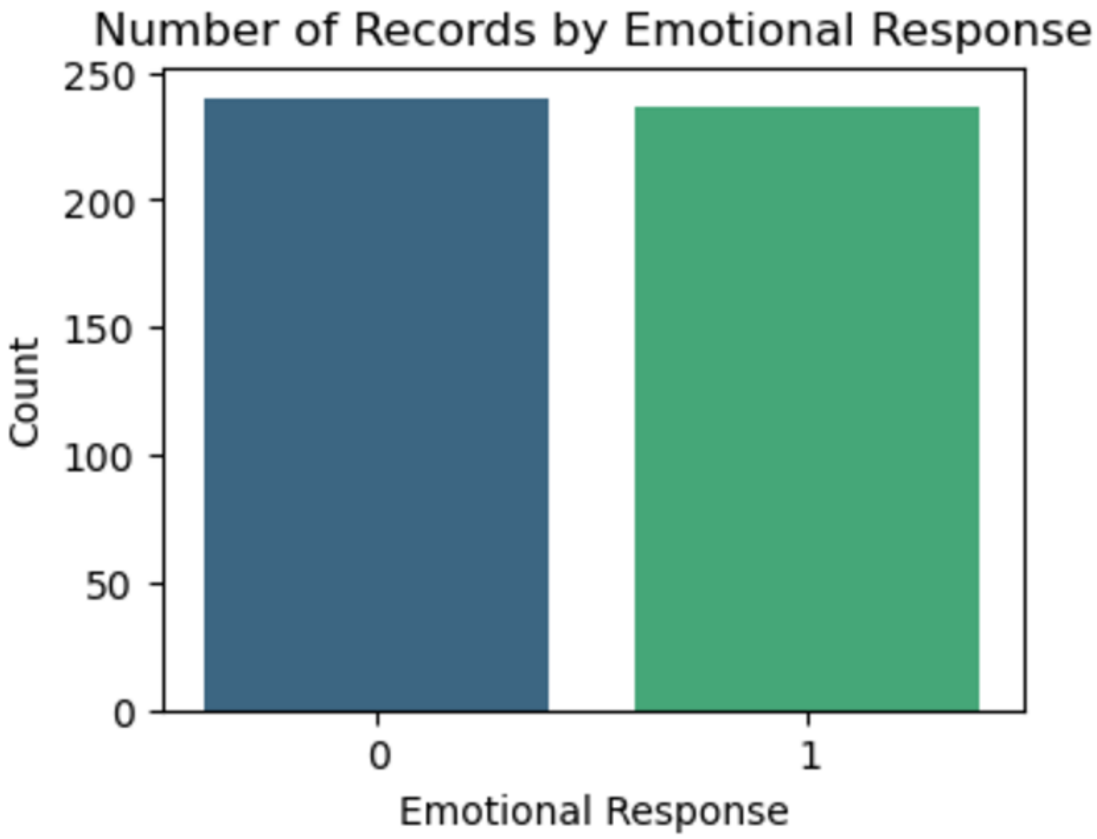
Naive Bayes works by looking at how often things happen (frequencies), so count plots can help in:

- make sure your data is evenly spread or at least realistic for the model.
- If your predictions don't match the actual data distribution, a count plot can show where it's going wrong.
- The model's predictions are based on frequencies. By looking at the count plot, you can better understand why the model makes certain decisions.

Code:

```
[237]: # Countplot for 'emotional response' distribution
plt.figure(figsize=(4, 3))
sns.countplot(data=data, x='emotional response', palette='viridis')
plt.title("Number of Records by Emotional Response")
plt.xlabel("Emotional Response")
plt.ylabel("Count")
plt.show()
```

Output:



Key Observations from the countplot:

Emotional response (0 and 1 or “yes” and “no”) are almost equal in height, indicating that the dataset has a nearly balanced distribution of the Emotional Response variable. This is good for machine learning models, as balanced classes typically result in better performance.

Density Plot:

Purpose of the Visualization:

A density plot shows the distribution of data over a continuous range. It's like a smooth curve that represents how often certain values occur. The plot helps us see patterns like where most data points are located, if the data is spread out or tightly packed, and if there are any peaks where the data is more concentrated.

Need in Machine Learning:

In machine learning, the density plot helps us:

- Understand how features are distributed, which is important when choosing the right model or preprocessing steps.
- See if the data is balanced or if it has any skew (like if most data is to one side).
- Detect unusual patterns, such as outliers or gaps in the data.

In Naive Bayes:

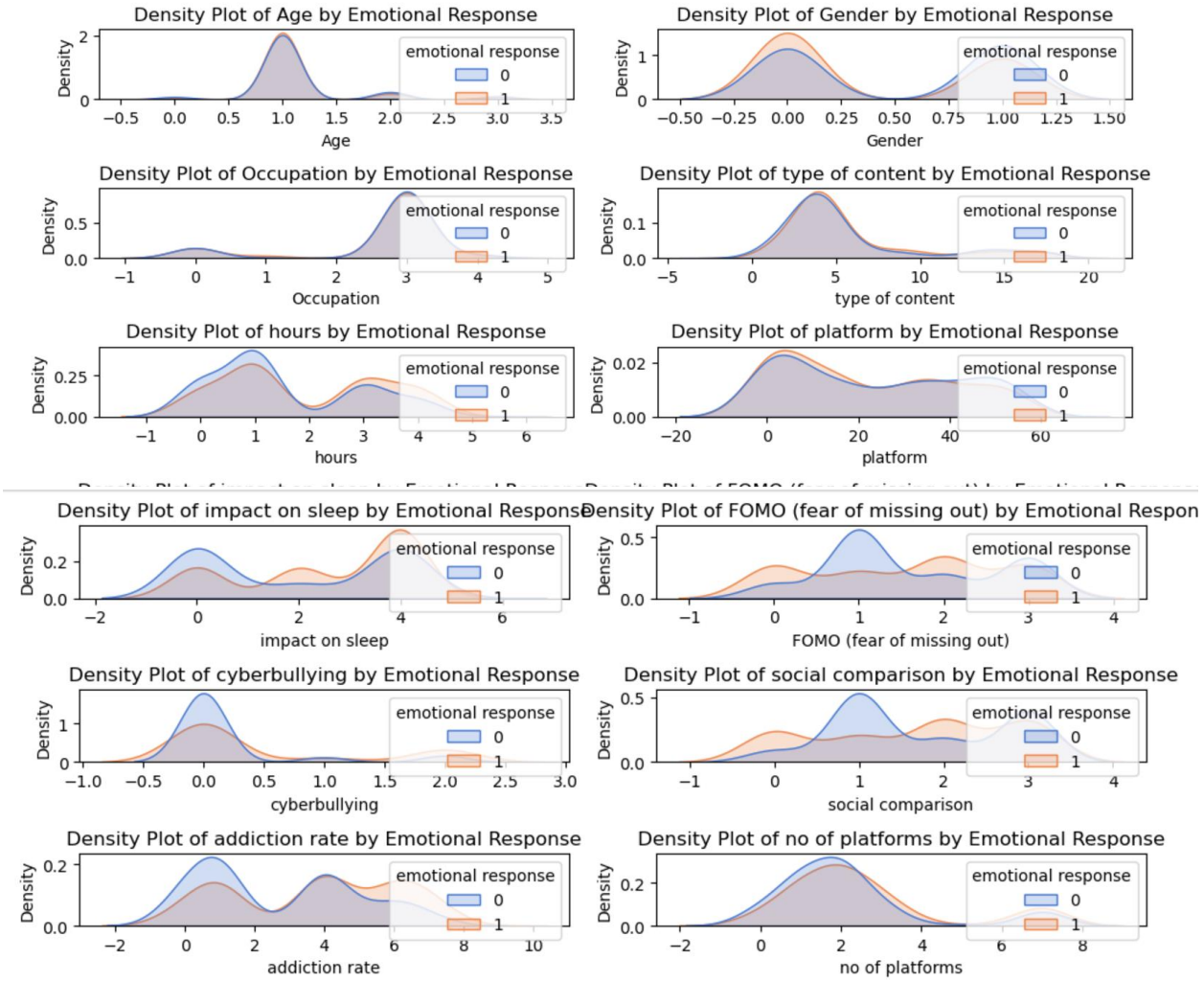
- Check if the features in each class (like emotional response 0 and 1) follow a similar distribution or if they are different. If the distributions are very different, the model will likely perform better with those features.
- Spot if a feature is skewed or has gaps, which might suggest we need to adjust the data (like log transformation).

Code:

```
[291]: import matplotlib.pyplot as plt
import seaborn as sns
features = [col for col in data.columns if col != 'emotional response']
plt.figure(figsize=(10, 10))  # Adjust to fit your number of plots
for idx, feature in enumerate(features):

    plt.subplot((len(features) + 1) // 2, 2, idx + 1)  # Dynamically arrange subplots based on the number of features
    sns.kdeplot(data=data, x=feature, hue='emotional response', fill=True, common_norm=False, palette='muted')
    plt.title(f"Density Plot of {feature} by Emotional Response")
    plt.xlabel(feature)
    plt.ylabel('Density')
plt.savefig('density plots.png')
plt.tight_layout()  # To avoid overlap between plots
plt.show()
```

Output:


Density Plot of Age by Emotional Response
Density
Age
emotional response
0
1
Density Plot of Gender by Emotional Response
Gender
Density Plot of Occupation by Emotional Response
Occupation
Density Plot of type of content by Emotional Response
type of content
Density Plot of hours by Emotional Response
hours
Density Plot of platform by Emotional Response
platform
Density Plot of impact on sleep by Emotional Response
impact on sleep
Density Plot of FOMO (fear of missing out) by Emotional Respon
FOMO (fear of missing out)
Density Plot of cyberbullying by Emotional Response
cyberbullying
Density Plot of social comparison by Emotional Response
social comparison
Density Plot of addiction rate by Emotional Response
addiction rate
Density Plot of no of platforms by Emotional Response
no of platforms

TASK 2:

Naïve Bayes Implementation:

Step # 1: importing libraries

step 1: importing libraries

```
[5]: import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.naive_bayes import GaussianNB
     from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
     from sklearn.preprocessing import MinMaxScaler, LabelEncoder
     import seaborn as sns
     import matplotlib.pyplot as plt
```

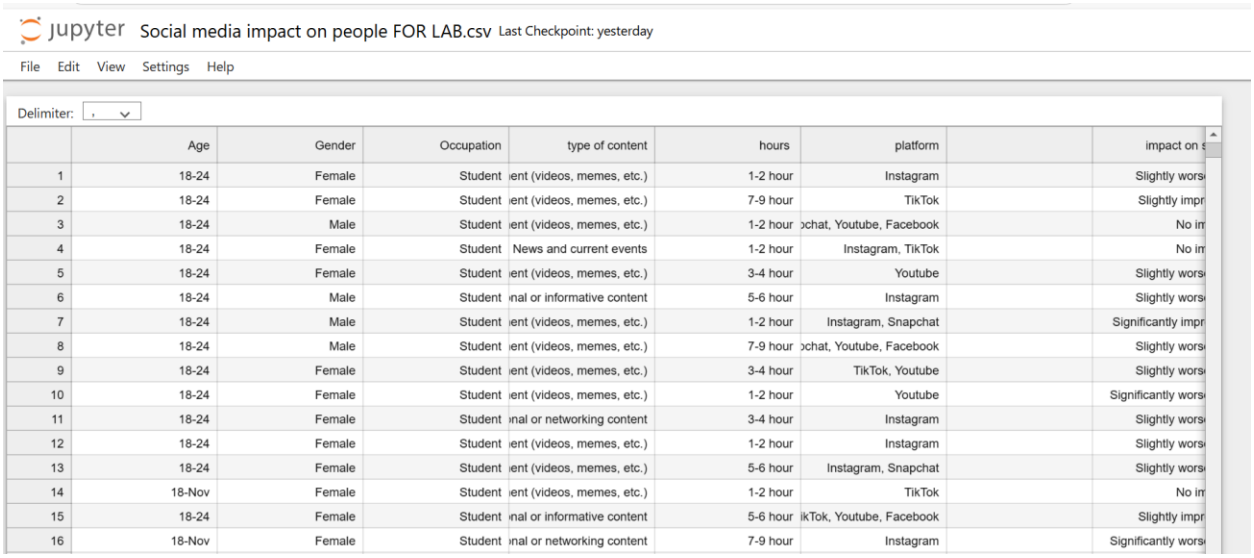
Explanation:

- **import pandas as pd:**
  Used to handle and manipulate data in the form of dataframes (tables).
- **from sklearn.model_selection import train_test_split**
  Splits the dataset into training and testing sets to evaluate the model on unseen data.
- **from sklearn.naive_bayes import GaussianNB**
  A classifier for building a Naive Bayes model, suitable for normally distributed (Gaussian) data.
- **from sklearn.metrics import accuracy_score, classification_report, confusion_matrix**
  accuracy_score: Measures the percentage of correct predictions.
  classification_report: Provides precision, recall, and F1-score for each class.
  confusion_matrix: Displays actual vs. predicted values in a matrix.
- **from sklearn.preprocessing import MinMaxScaler, LabelEncoder**
  Scales numerical features to a fixed range (typically 0 to 1) for consistency.
  Converts categorical labels (e.g., "yes" and "no") into numeric values (e.g., 1 and 0).
- **import seaborn as sns:**
  A visualization library for creating attractive statistical plots.
- **import matplotlib.pyplot as plt**

A library for generating static, animated, and interactive plots for visualizing data.

step # 2: loading and preprocessing dataset:

Data before preprocessing:

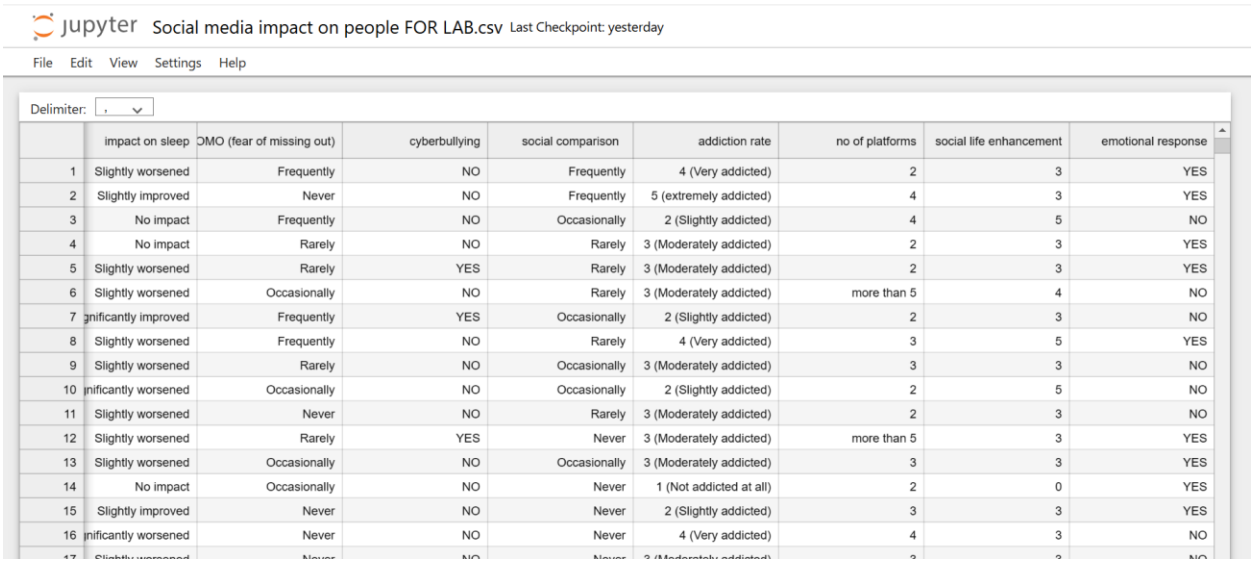


jupyter Social media impact on people FOR LAB.csv Last Checkpoint: yesterday

File Edit View Settings Help

Delimiter: ,

	Age	Gender	Occupation	type of content	hours	platform		impact on s
1	18-24	Female	Student	ient (videos, memes, etc.)	1-2 hour	Instagram		Slightly wors
2	18-24	Female	Student	ient (videos, memes, etc.)	7-9 hour	TikTok		Slightly impr
3	18-24	Male	Student	ient (videos, memes, etc.)	1-2 hour	ɔchat, Youtube, Facebook		No im
4	18-24	Female	Student	News and current events	1-2 hour	Instagram, TikTok		No im
5	18-24	Female	Student	ient (videos, memes, etc.)	3-4 hour	Youtube		Slightly wors
6	18-24	Male	Student	inal or informative content	5-6 hour	Instagram		Slightly wors
7	18-24	Male	Student	ient (videos, memes, etc.)	1-2 hour	Instagram, Snapchat		Significantly impr
8	18-24	Male	Student	ient (videos, memes, etc.)	7-9 hour	ɔchat, Youtube, Facebook		Slightly wors
9	18-24	Female	Student	ient (videos, memes, etc.)	3-4 hour	TikTok, Youtube		Slightly wors
10	18-24	Female	Student	ient (videos, memes, etc.)	1-2 hour	Youtube		Significantly wors
11	18-24	Female	Student	inal or networking content	3-4 hour	Instagram		Slightly wors
12	18-24	Female	Student	ient (videos, memes, etc.)	1-2 hour	Instagram		Slightly wors
13	18-24	Female	Student	ient (videos, memes, etc.)	5-6 hour	Instagram, Snapchat		Slightly wors
14	18-Nov	Female	Student	ient (videos, memes, etc.)	1-2 hour	TikTok		No im
15	18-24	Female	Student	inal or informative content	5-6 hour	ikTok, Youtube, Facebook		Slightly impr
16	18-Nov	Female	Student	inal or networking content	7-9 hour	Instagram		Significantly wors

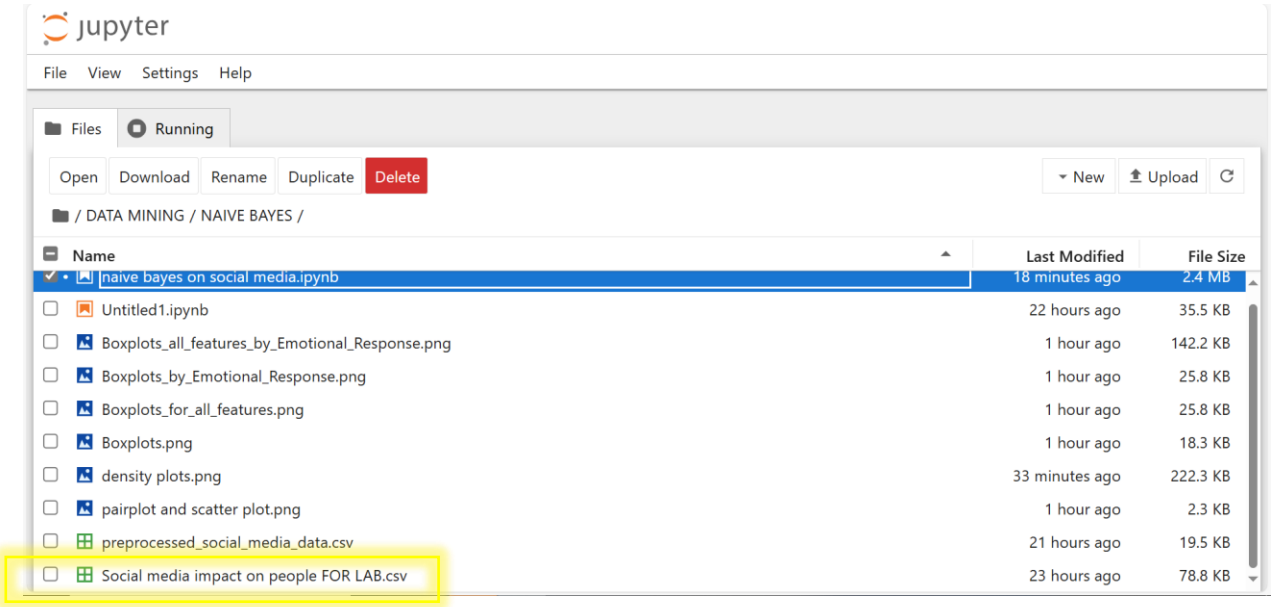


jupyter Social media impact on people FOR LAB.csv Last Checkpoint: yesterday

File Edit View Settings Help

Delimiter: ,

	impact on sleep	ɔMO (fear of missing out)	cyberbullying	social comparison	addiction rate	no of platforms	social life enhancement	emotional response
1	Slightly worsened	Frequently	NO	Frequently	4 (Very addicted)	2	3	YES
2	Slightly improved	Never	NO	Frequently	5 (extremely addicted)	4	3	YES
3	No impact	Frequently	NO	Occasionally	2 (Slightly addicted)	4	5	NO
4	No impact	Rarely	NO	Rarely	3 (Moderately addicted)	2	3	YES
5	Slightly worsened	Rarely	YES	Rarely	3 (Moderately addicted)	2	3	YES
6	Slightly worsened	Occasionally	NO	Rarely	3 (Moderately addicted)	more than 5	4	NO
7	gnificantly improved	Frequently	YES	Occasionally	2 (Slightly addicted)	2	3	NO
8	Slightly worsened	Frequently	NO	Rarely	4 (Very addicted)	3	5	YES
9	Slightly worsened	Rarely	NO	Occasionally	3 (Moderately addicted)	3	3	NO
10	inificantly worsened	Occasionally	NO	Occasionally	2 (Slightly addicted)	2	5	NO
11	Slightly worsened	Never	NO	Rarely	3 (Moderately addicted)	2	3	NO
12	Slightly worsened	Rarely	YES	Never	3 (Moderately addicted)	more than 5	3	YES
13	Slightly worsened	Occasionally	NO	Occasionally	3 (Moderately addicted)	3	3	YES
14	No impact	Occasionally	NO	Never	1 (Not addicted at all)	2	0	YES
15	Slightly improved	Never	NO	Never	2 (Slightly addicted)	3	3	YES
16	inificantly worsened	Never	NO	Never	4 (Very addicted)	4	3	NO



- the dataset is loaded in jupyter notebook in rapid miner in folder naïve bayes.

Complete code:

step 2: loading dataset and preprocessing

[170]:
```
# Import necessary libraries
import pandas as pd
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.impute import SimpleImputer

# Load the dataset
file_path = 'Social media impact on people FOR LAB.csv'  # Replace with your file path
data = pd.read_csv(file_path)

# Clean column names to avoid issues
data.columns = data.columns.str.strip()

# Step 1: Inspect the 'Age' column for any unexpected values
print("Unique values in 'Age' column:", data['Age'].unique())
data['Age'] = data['Age'].replace('34-55', '34-54')
```

```
# Step 3: Replace '18-nov' with '11-18' in the 'Age' column
data['Age'] = data['Age'].replace('18-Nov', '11-18')


# Step 4: Remove duplicate rows
data = data.drop_duplicates()

# Step 5: Fill missing values in 'no of platforms' (or any other column) with the most frequent value
if 'no of platforms' in data.columns:
    data['no of platforms'] = data['no of platforms'].fillna(data['no of platforms'].mode()[0])

# Step 6: Remove completely empty columns (like 'Unnamed: 6')
data = data.dropna(axis=1, how='all')
print("Final unique values in 'emotional response':", data['emotional response'].unique())
data['emotional response'] = data['emotional response'].replace('yes','YES')
data['emotional response'] = data['emotional response'].replace('No','NO')

# Step 7: Label encode categorical columns (Age, Gender, Emotional Response, etc.)
categorical_columns = data.select_dtypes(include=['object']).columns  # Identify categorical columns
label_encoders = {}  # Store label encoders for each column (optional, for decoding later)
```

```
for col in categorical_columns:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le  # Save the encoder (optional)

# Step 8: Handle missing values in the entire dataset using SimpleImputer for numerical columns
# Identify numerical columns
numerical_columns = data.select_dtypes(include=['int64', 'float64']).columns

# Initialize the imputer (mean strategy to fill missing values for numerical columns)
imputer = SimpleImputer(strategy='mean')
data[numerical_columns] = imputer.fit_transform(data[numerical_columns])

# Step 9: Normalize numerical columns
scaler = MinMaxScaler()
data[numerical_columns] = scaler.fit_transform(data[numerical_columns])
```

```
# Display dataset preview
print("Preprocessed Data Preview:")
display(data.head())
# Save the preprocessed data to a new CSV file
output_file_path = 'preprocessed_social_media_data.csv'  # Specify your desired output path
data.to_csv(output_file_path, index=False)

print(f"Preprocessed data saved to: {output_file_path}")
```

Step by step explaination of code:

Importing necessary libraries:

```
[170]: # Import necessary libraries
       import pandas as pd
       from sklearn.preprocessing import LabelEncoder, MinMaxScaler
       from sklearn.impute import SimpleImputer
```

Explaination:

- **pandas (pd):** Used to load and manipulate the dataset (dataframes).
- **LabelEncoder:** Used to convert categorical variables (like strings) into numeric values.
- **MinMaxScaler:** Used to scale numerical data to a fixed range (usually 0 to 1).
- **SimpleImputer:** Used to fill in missing (NaN) values in the dataset, either with the mean, median, or most frequent value.

Load the dataset:

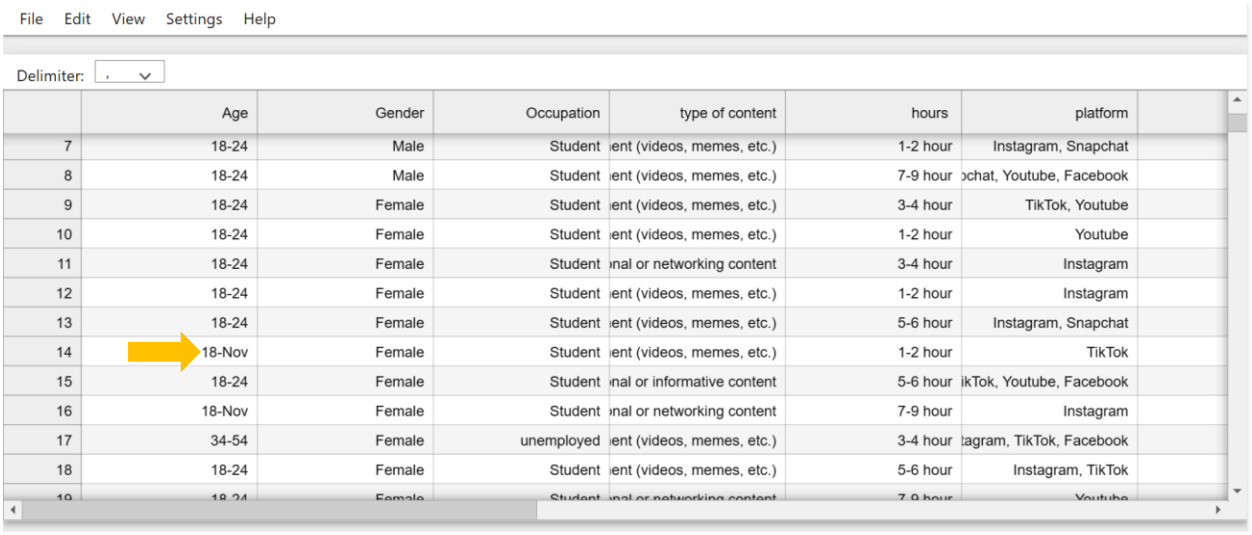
```
# Load the dataset
file_path = 'Social media impact on people FOR LAB.csv'  # Replace with your file path
data = pd.read_csv(file_path)
```

Inspect and clean values in the 'Age' column

```
# Step 1: Inspect the 'Age' column for any unexpected values
print("Unique values in 'Age' column:", data['Age'].unique())
data['Age'] = data['Age'].replace('34-55', '34-54')

# Step 3: Replace '18-nov' with '11-18' in the 'Age' column
data['Age'] = data['Age'].replace('18-Nov', '11-18')
```

- When dataset was loaded , I noticed that the age group **11-18** was converted to date format which need to be replaced to required age group so replace was used for that purpose.



File Edit View Settings Help

Delimiter: ,

	Age	Gender	Occupation	type of content	hours	platform
7	18-24	Male	Student	ent (videos, memes, etc.)	1-2 hour	Instagram, Snapchat
8	18-24	Male	Student	ent (videos, memes, etc.)	7-9 hour	chat, Youtube, Facebook
9	18-24	Female	Student	ent (videos, memes, etc.)	3-4 hour	TikTok, Youtube
10	18-24	Female	Student	ent (videos, memes, etc.)	1-2 hour	Youtube
11	18-24	Female	Student	nal or networking content	3-4 hour	Instagram
12	18-24	Female	Student	ent (videos, memes, etc.)	1-2 hour	Instagram
13	18-24	Female	Student	ent (videos, memes, etc.)	5-6 hour	Instagram, Snapchat
14	18-Nov	Female	Student	ent (videos, memes, etc.)	1-2 hour	TikTok
15	18-24	Female	Student	nal or informative content	5-6 hour	ikTok, Youtube, Facebook
16	18-Nov	Female	Student	nal or networking content	7-9 hour	Instagram
17	34-54	Female	unemployed	ent (videos, memes, etc.)	3-4 hour	agram, TikTok, Facebook
18	18-24	Female	Student	ent (videos, memes, etc.)	5-6 hour	Instagram, TikTok
19	18-24	Female	Student	nal or networking content	7-9 hour	Youtube

- With .unique() , it listed me all values in age group column where I noticed the age group **34-55** was wrong which should be replaced to **34-54.**

```
Unique values in 'Age' column: ['18-24' '18-Nov' '34-54' '24-34' '34-55']
```

- **data = data.drop_duplicates() :** Removes any duplicate rows in the dataset to ensure that each row is unique.

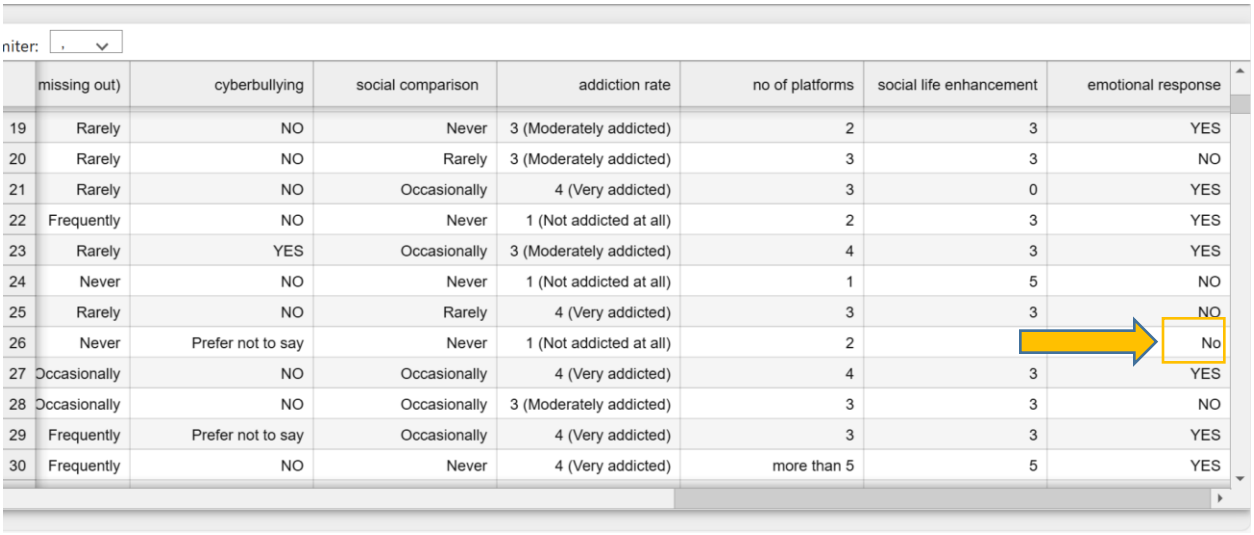
```
# Step 3: Fill missing values in 'no of platforms' (or any other column) with the most frequent value
if 'no of platforms' in data.columns:
    data['no of platforms'] = data['no of platforms'].fillna(data['no of platforms'].mode()[0])
```

- **this** part of code is used to replace the missing values with **mode()** which is the most frequently occuring value in the column.

```
# Step 6: Remove completely empty columns (like 'Unnamed: 6')
data = data.dropna(axis=1, how='all')
print("Final unique values in 'emotional response':", data['emotional response'].unique())
data['emotional response'] = data['emotional response'].replace('yes','YES')
data['emotional response'] = data['emotional response'].replace('No','NO')
```

- **data = data.dropna(axis=1, how='all'):** is used to remove columns which are fully empty.
- Again replace operators are used bcz I noticed 4 labels in emotional response column instead of 2 probably spelling errors.

```
unique values in 'emotional response': ['YES' 'NO' 'No' 'yes']
```



	missing out)	cyberbullying	social comparison	addiction rate	no of platforms	social life enhancement	emotional response
19	Rarely	NO	Never	3 (Moderately addicted)	2	3	YES
20	Rarely	NO	Rarely	3 (Moderately addicted)	3	3	NO
21	Rarely	NO	Occasionally	4 (Very addicted)	3	0	YES
22	Frequently	NO	Never	1 (Not addicted at all)	2	3	YES
23	Rarely	YES	Occasionally	3 (Moderately addicted)	4	3	YES
24	Never	NO	Never	1 (Not addicted at all)	1	5	NO
25	Rarely	NO	Rarely	4 (Very addicted)	3	3	NO
26	Never	Prefer not to say	Never	1 (Not addicted at all)	2		No
27	Occasionally	NO	Occasionally	4 (Very addicted)	4	3	YES
28	Occasionally	NO	Occasionally	3 (Moderately addicted)	3	3	NO
29	Frequently	Prefer not to say	Occasionally	4 (Very addicted)	3	3	YES
30	Frequently	NO	Never	4 (Very addicted)	more than 5	5	YES

Encoding categorical columns using label encoding:

```
# Step 7: Label encode categorical columns (Age, Gender, Emotional Response, etc.)
categorical_columns = data.select_dtypes(include=['object']).columns  # Identify categorical columns
label_encoders = {}  # Store label encoders for each column (optional, for decoding later)

for col in categorical_columns:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le  # Save the encoder (optional)
```

Explaination:

- Label encoding is a crucial step before applying any model, we need top convert categorical columns to numerical sp that data stays consistent.
- **data.select_dtypes(include=['object']).columns:** Identifies columns with categorical (text) data (i.e., columns with data type object).
- **LabelEncoder():** Initializes a LabelEncoder to convert categorical values into numeric labels.
- **data[col] = le.fit_transform(data[col]):** Applies the label encoding to each categorical column.
- **label_encoders[col] = le:** stores the encoder value for each column.

Normalize numerical features:

```
# Step 9: Normalize numerical columns
scaler = MinMaxScaler()
data[numerical_columns] = scaler.fit_transform(data[numerical_columns])

# Display dataset preview
print("Preprocessed Data Preview:")
display(data.head())
# Save the preprocessed data to a new CSV file
output_file_path = 'preprocessed_social_media_data.csv'  # Specify your desired output path
data.to_csv(output_file_path, index=False)

print(f"Preprocessed data saved to: {output_file_path}")
```

Explaination:

- normalization help transforming numerical features into a shorter scale which helps model perform better.
- **MinMaxScaler**(): Initializes a MinMaxScaler to scale numerical values to a range (0 to 1).
- **data[numerical_columns] = scaler.fit_transform(data[numerical_columns]):** Applies the scaler to normalize the identified numerical columns.
- Then the preprocessed data is saved so that we can view the preprocessed data.

Output:

```
Unique values in 'Age' column: ['18-24' '18-Nov' '34-54' '24-34' '34-55']
Final unique values in 'emotional response': ['YES' 'NO' 'No' 'yes']
Preprocessed Data Preview:
```

	Age	Gender	Occupation	type of content	hours	platform	impact on sleep	FOMO (fear of missing out)	cyberbullying	social comparison	addiction rate	no of platforms	social life enhancement	emotional response
0	1	0	3	4	0	1	4	0	0	0	6	1	0.6	1
1	1	0	3	4	4	42	3	1	0	0	7	3	0.6	1
2	1	1	3	4	0	27	0	0	0	2	1	3	1.0	0
3	1	0	3	14	0	10	0	3	0	3	4	1	0.6	1
4	1	0	3	4	1	52	4	3	2	3	4	1	0.6	1

```
Preprocessed data saved to: preprocessed_social_media_data.csv
```

Data after preprocessing:

jupyter preprocessed_social_media_data.csv Last Checkpoint: 23 hours ago

File Edit View Settings Help

Delimiter: ,

	Age	Gender	Occupation	type of content	hours	platform	impact on sleep	OMO (fear of missing o
1	1	0	3	4	0	1	4	
2	1	0	3	4	4	42	3	
3	1	1	3	4	0	27	0	
4	1	0	3	14	0	10	0	
5	1	0	3	4	1	52	4	
6	1	1	3	2	3	1	4	
7	1	1	3	4	0	4	1	
8	1	1	3	4	4	27	4	
9	1	0	3	4	1	47	4	
10	1	0	3	4	0	52	2	
11	1	0	3	17	1	1	4	
12	1	0	3	4	0	1	4	
13	1	0	3	4	3	4	4	
14	0	0	3	4	0	42	0	
15	1	0	3	2	3	20	3	
16	0	0	3	17	4	1	2	
17	3	0	4	4	1	12	4	
18	1	0	3	4	3	10	4	

jupyter preprocessed_social_media_data.csv Last Checkpoint: 23 hours ago

File Edit View Settings Help

Delimiter: ,

	impact on sleep	OMO (fear of missing out)	cyberbullying	social comparison	addiction rate	no of platforms	social life enhancement	emotional response
1	4	0	0	0	6	1	0.6000000000000001	1
2	3	1	0	0	7	3	0.6000000000000001	1
3	0	0	0	2	1	3	1.0	0
4	0	3	0	3	4	1	0.6000000000000001	1
5	4	3	2	3	4	1	0.6000000000000001	1
6	4	2	0	3	4	7	0.8	0
7	1	0	2	2	1	1	0.6000000000000001	0
8	4	0	0	3	6	2	1.0	1
9	4	3	0	2	4	2	0.6000000000000001	0
10	2	2	0	2	1	1	1.0	0
11	4	1	0	3	4	1	0.6000000000000001	0
12	4	3	2	1	4	7	0.6000000000000001	1
13	4	2	0	2	4	2	0.6000000000000001	1
14	0	2	0	1	0	1	0.0	1
15	3	1	0	1	1	2	0.6000000000000001	1
16	2	1	0	1	6	3	0.6000000000000001	0
17	4	1	0	1	4	2	0.6000000000000001	0
18	4	2	0	0	7	2	0.6000000000000001	1

step # 3: Splitting the dataset:

step 3:splitting data

```
[172]: # Define features (X) and labels (y)
X = data.drop(columns=['emotional response'])  # Features (all columns except 'emotional response')
y = data['emotional response']  # Labels (emotional response column)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Print the shape of the training and testing datasets
print("\nDataset split complete:")
print(f"Training samples: {X_train.shape[0]}, Testing samples: {X_test.shape[0]}")
```

Explaination:

- **X = data.drop(columns=['emotional response']):**
  This part of code is used to exclude the emotional response column and include remaining . and the result is stored in variable X.
- **y = data['emotional response']:**
  This defines the labels y, which is the column 'emotional response'. This column will be the target variable the model is trying to predict.
- **X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42):**
  This part of code is used to split data data into 70% training and 30% testing data
- **random_state=42:**
  This ensures the split is reproducible. By using the same random_state, you can get the same split of the data each time the code runs.

Output:

```
Dataset split complete:
Training samples: 333, Testing samples: 143
```

step # 4:Training the model:

step 4: training model

```
[174]: # Initialize the Naive Bayes classifier
       nb_model = GaussianNB()

       # Train the model
       nb_model.fit(X_train, y_train)

       # Print confirmation
       print("\nNaive Bayes model trained successfully.")
```

```
Naive Bayes model trained successfully.
```

Explanation:

- **fit(X_train, y_train):** This method trains the Naive Bayes model using the training data **(X_train for features and y_train for labels).** The model learns the relationship between the features and the labels during this step
- After training, the model can make predictions on new, unseen data**.**

step # 5:Evaluating model:

step 5: evaluating model

```
[196]: # Make predictions on the test set
       y_pred = nb_model.predict(X_test)

       # Evaluate model accuracy
       accuracy = accuracy_score(y_test, y_pred)
       print("\nModel Evaluation:")
       print(f"Accuracy: {accuracy:.2f}")
       print("Classification Report:\n", classification_report(y_test, y_pred))
```

Explanation:

- y_pred = nb_model.predict(X_test):
  this part of code uses the Naïve Bayes model **nb.model** to make predictions on the test data**.(X_test).**

- The input features (X_test) are passed to the model, and it outputs predicted labels (y_pred), which represent the model's guess for the target variable (in this case, emotional response) based on the input features.
- accuracy = accuracy_score(y_test, y_pred):
- **accuracy_score(y_test, y_pred):** This function compares the predicted labels **(y_pred)** with the true labels **(y_test)** from the test set. It calculates the accuracy of the model, which is the percentage of correct predictions out of all predictions made.
- **classification_report(y_test, y_pred):**
  This generates a detailed classification report that includes:
  Precision: The proportion of positive predictions that were actually correct.
  Recall: The proportion of actual positives that were correctly predicted.
  F1-Score: A balance between precision and recall, which is especially useful when dealing with imbalanced datasets.

Output:

```
Model Evaluation:
Accuracy: 0.71
Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.68      0.71        72
           1       0.70      0.75      0.72        71

    accuracy                           0.71       143
   macro avg       0.71      0.71      0.71       143
weighted avg       0.71      0.71      0.71       143
```

Accuracy:

- Accuracy represents the overall percentage of correct predictions the model made
- In this case the Accuracy is **71%.**

Classification report:

Class 0 (NO in emotional response):

- **Precision: 0.73:** Precision means correctly predicted positive instances (class 0) out of all instances predicted as class 0. In other words, 73% of the times the model predicted class 0, it was correct.
- **Recall: 0.68:** Recall is the proportion of actual instances of class 0 that were correctly predicted as class 0. So, 68% of the actual class 0 instances were correctly identified by the model.

- **F1-Score: 0.71:** The F1-score is the harmonic mean of precision and recall, providing a balance between the two. In this case, the F1-score is 0.71, which represents a good balance between precision and recall for class 0.
- Support: 72: Support refers to the number of actual instances of class 0 in the test set (72 instances).

Class 1 (YES in emotional response):

- **Precision: 0.70:** Precision for class 1 is 70%, meaning 70% of the predictions for class 1 were correct.
- **Recall: 0.75:** Recall for class 1 is 75%, meaning 75% of the actual class 1 instances were correctly predicted by the model.
- **F1-Score: 0.72:** The F1-score for class 1 is 0.72, reflecting a good balance between precision and recall for this class as well.
- **Support: 71:** Support refers to the number of actual instances of class 1 in the test set (71 instances).

TASK 3:

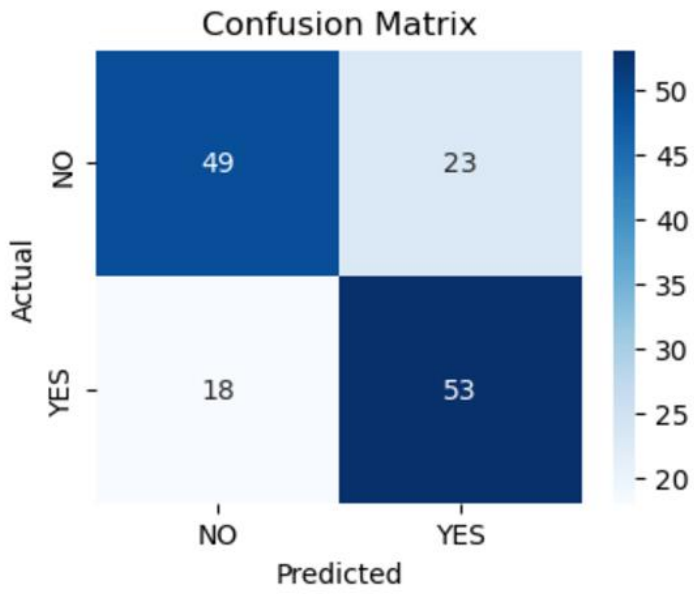
Visualize insights:

Confusion Matrix:

step 6 :confusion matrix

[178]:
```
# Plot Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(4, 3))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoders['emotional response'].classes_,
            yticklabels=label_encoders['emotional response'].classes_)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

- **confusion_matrix(y_test, y_pred):** This function compares the true labels (y_test) with the predicted labels (y_pred) and returns a confusion matrix. A confusion matrix shows the number of correct and incorrect predictions for each class**:**
- **True Positives (TP**): Correctly predicted positive cases.
- **True Negatives (TN):** Correctly predicted negative cases**.**
- **False Positives (FP):** Predicted positive when the true value was negative.
- **False Negatives (FN):** Predicted negative when the true value was positive**.**



KEY INSIGHTS:

TN: 49 NO cases are correctly predicted as NO

FN: 23 NO cases predicted as YES

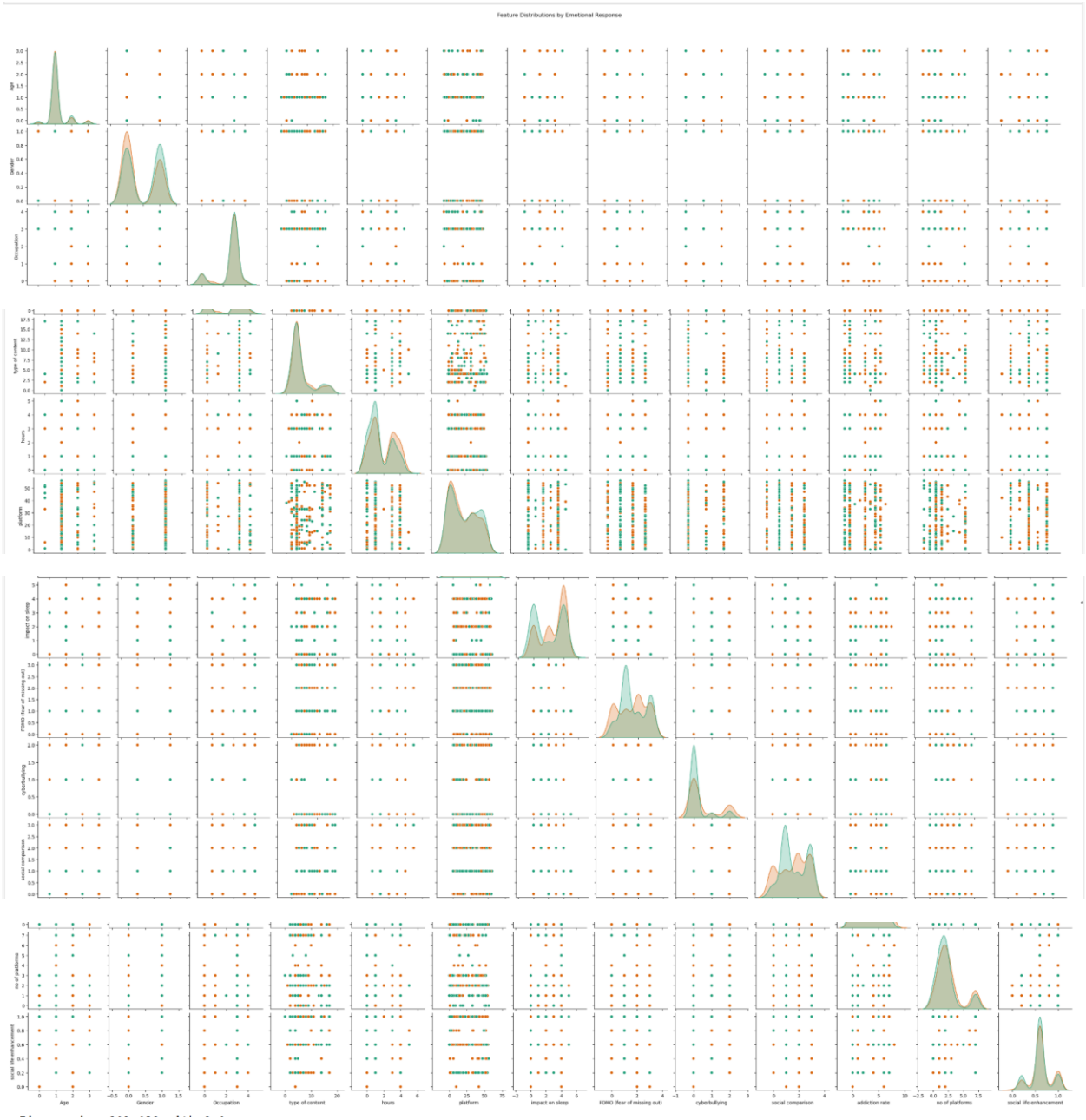
TP: 53 YES cases correctly predicted as YES

FP: 18 YES cases predicted as NO (False Negatives):

Feature Distributions:

Feature Distributions by emotional response:

```
[251]: # Plot Pairplot to visualize feature distributions based on 'emotional response'
sns.pairplot(data, hue='emotional response', palette='Dark2')
plt.suptitle("Feature Distributions by Emotional Response", y=1.02)
plt.show()
plt.savefig('pairplot and scatter plot.png')
```


Feature Distributions by Emotional Response
Age
Gender
Occupation
type of content
hours
platform
impact on sleep
FOMO (fear of missing out)
cyberbullying
social comparison
addiction rate
no of platforms
social life enhancement
emotional response
0
1

KEY INSIGHTS:

PairPlot:

This pairplot visualizes relationships between different features (attributes) in the dataset, split by the emotional response classes:

Green (0): No emotional response.

Orange (1): Emotional response.

1. Age :

Diagonal Plot: Both classes (0 and 1) have a similar age distribution. Emotional response doesn't depend strongly on age.

Scatterplots:

No clear separation between classes when combined with other attributes like gender or platform.

Slight clustering appears when combined with FOMO or impact on sleep, but overall, age is weakly predictive.

2. Gender

Diagonal Plot: Clear separation between the two classes.

One gender seems slightly more likely to have emotional responses (orange), but the distinction isn't strong.

Scatterplots:

Gender alone isn't impactful unless combined with features like FOMO, impact on sleep, or social comparison.

3. Occupation

Diagonal Plot: Emotional responses occur across various occupations. No strong signal here.

Scatterplots:

When combined with hours on social media, certain occupations show a stronger likelihood of emotional responses.

4. Type of Content

Diagonal Plot:

A significant difference is visible in content type preference between classes. Some content types are linked strongly to emotional responses.

Scatterplots:

Emotional responses increase when paired with certain platforms or hours spent.

Strong clustering for emotional responses (orange) around specific content types.

5. Hours (Time Spent on Social Media)

Diagonal Plot:

Both classes show overlap, but emotional responses slightly increase with higher hours spent on social media.

Scatterplots:

When combined with addiction rate or impact on sleep, higher hours contribute more to emotional responses.

6. Platform

Diagonal Plot: Platforms used show minimal differences between emotional response classes.

Scatterplots:

Emotional response patterns are more visible when platforms are combined with features like content type and addiction rate.

7. Impact on Sleep

Diagonal Plot:

Strong separation between classes. People with greater sleep disruption (higher values) are more likely to show emotional responses.

Scatterplots:

Sleep disruption correlates with features like FOMO and addiction rate to predict emotional responses well.

8. FOMO (Fear of Missing Out)

Diagonal Plot:

Clear separation—higher FOMO is strongly linked with emotional responses.

Scatterplots:

FOMO shows strong clustering with addiction rate and impact on sleep, making it a key indicator of emotional response.

9. Cyberbullying

Diagonal Plot:

Some separation, but emotional responses aren't as tightly linked to cyberbullying.

Scatterplots:

Moderate clustering with social comparison and addiction rate, but the feature's predictive power is moderate.

10. Social Comparison

Diagonal Plot:

Moderate separation—higher levels of social comparison are somewhat linked to emotional responses.

Scatterplots:

Stronger when combined with features like FOMO or impact on sleep.

11. Addiction Rate

Diagonal Plot:

Strong separation higher addiction rates strongly correlate with emotional responses.

Scatterplots:

Strong clustering with features like sleep disruption, FOMO, and hours spent, making it one of the most impactful features.

12. Number of Platforms Used

Diagonal Plot:

Minimal separation—number of platforms used doesn't strongly affect emotional responses.

Scatterplots:

Weak impact even when paired with features like content type or addiction rate.

13. Social Life Enhancement

Diagonal Plot:

Negligible separation. Using social media for enhancing social life doesn't predict emotional responses well.

Scatterplots:

Almost no impact in combination with any other feature.

IN SUMMARY:

FEATURES who are highly influencing emotional response:

FOMO (Fear of Missing Out)

Addiction Rate

Impact on Sleep

Type of Content

Features with Some Influence:

Cyberbullying

Social Comparison

Features having no influence:

Age

Gender

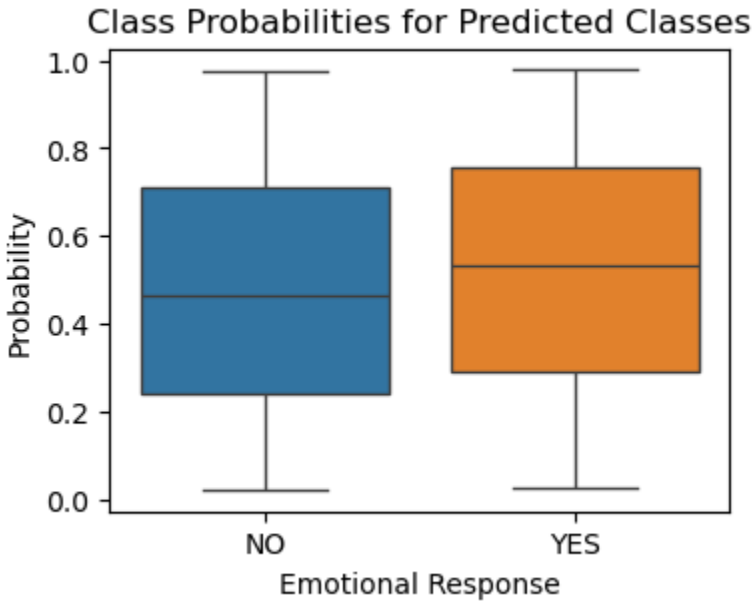
Platforms

Social Life Enhancement

Class probabilities:

```
[247]: # Predict probabilities for the test set
probs = nb_model.predict_proba(X_test)
# Create a DataFrame for visualization of probabilities
probs_df = pd.DataFrame(probs, columns=classes, index=y_test.index)
probs_df['True Label'] = y_test.values  # Add the true labels for reference

# Visualize probabilities
plt.figure(figsize=(4, 3))
sns.boxplot(data=probs_df.iloc[:, :-1])
plt.title("Class Probabilities for Predicted Classes")
plt.xlabel("Emotional Response")
plt.ylabel("Probability")
plt.show()
```



KEY INSIGHTS:

- The chart illustrates the distribution of predicted probabilities for two classes: "NO" and "YES".
- The blue box shows the predicted probabilities for "NO" responses, and the orange box shows probabilities for "YES" responses.
- The horizontal line in each box represents the median (middle value) of the model's confidence for each class.
- Both "NO" and "YES" have a wide range of probabilities, from very low (close to 0) to very high (close to 1).
- The probabilities for "NO" and "YES" overlap quite a bit, meaning the model often isn't fully confident when deciding between the two.

Conclusion:

This document deals with the exploring the naïve bayes algorithm from data preparation to evaluating the model. Different visualization techniques (pairplot, scatterplot, boxplot, countplot, density plot etc) are applied on dataset to find insights from the dataset Key findings include that features such as FOMO, Addiction Rate, and Impact on Sleep have more influence on emotional response while others don't have much. The model was trained on training data and is evaluated on testing data. The model achieved an overall accuracy of 71% means the model is successful in predicting 71% of correct values. Visualization like confusion matrix, pairplot for feature distribution and class probabilities are helpful in visualizing the model behavior on unseen testing data.