

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

## Depression Anxiety Stress Scales Responses

### (1) Identify Columns

#### Part One (Q) :

| Column | Identifications   |
|--------|---|
| Q1     | I found myself getting upset by quite trivial things.   |
| Q2     | I was aware of dryness of my mouth.   |
| Q3     | I couldn't seem to experience any positive feeling at all.  |
| Q4     | I experienced breathing difficulty (eg, excessively rapid breathing, breathlessness in the absence of physical exertion).           |
| Q5     | I just couldn't seem to get going.  |
| Q6     | I tended to over-react to situations.   |
| Q7     | I had a feeling of shakiness (eg, legs going to give way).  |
| Q8     | I found it difficult to relax.  |
| Q9     | I found myself in situations that made me so anxious I was most relieved when they ended.   |
| Q10    | I felt that I had nothing to look forward to.   |
| Q11    | I found myself getting upset rather easily.   |
| Q12    | I felt that I was using a lot of nervous energy.  |
| Q13    | I felt sad and depressed.   |
| Q14    | I found myself getting impatient when I was delayed in any way (eg, elevators, traffic lights, being kept waiting).                 |
| Q15    | I had a feeling of faintness.   |
| Q16    | I felt that I had lost interest in just about everything.   |
| Q17    | I felt I wasn't worth much as a person.   |
| Q18    | I felt that I was rather touchy.  |
| Q19    | I perspired noticeably (eg, hands sweaty) in the absence of high temperatures or physical exertion.                                 |
| Q20    | I felt scared without any good reason.  |
| Q21    | I felt that life wasn't worthwhile.   |
| Q22    | I found it hard to wind down.   |
| Q23    | I had difficulty in swallowing.   |
| Q24    | I couldn't seem to get any enjoyment out of the things I did.   |
| Q25    | I was aware of the action of my heart in the absence of physical exertion (eg, sense of heart rate increase, heart missing a beat). |
| Q26    | I felt down-hearted and blue.   |
| Q27    | I found that I was very irritable.  |
| Q28    | I felt I was close to panic.  |
| Q29    | I found it hard to calm down after something upset me.  |
| Q30    | I feared that I would be "thrown" by some trivial but unfamiliar task.  |
| Q31    | I was unable to become enthusiastic about anything.   |
| Q32    | I found it difficult to tolerate interruptions to what I was doing.   |

|     |  |
|-----|--|
| Q33 | I was in a state of nervous tension.   |
| Q34 | I felt I was pretty worthless.   |
| Q35 | I was intolerant of anything that kept me from getting on with what I was doing. |
| Q36 | I felt terrified.  |
| Q37 | I could see nothing in the future to be hopeful about.                           |
| Q38 | I felt that life was meaningless.  |
| Q39 | I found myself getting agitated.   |
| Q40 | I was worried about situations in which I might panic and make a fool of myself. |
| Q41 | I experienced trembling (eg, in the hands).                                      |
| Q42 | I found it difficult to work up the initiative to do things.                     |

*Identify Some Symbols*

| Symbols | Identifications  |
|---------|--|
| A       | This response is stored in variable (e.g. Q1A).                    |
| E       | The time taken in milliseconds to answer that question (e.g. Q1E). |
| I       | Question's position in the survey (e.g. Q1I).                      |

*Identify Some Answers*

| Scale | Description   |
|-------|---|
| 1     | Did not apply to me at all.   |
| 2     | Applied to me to some degree, or some of the time.                  |
| 3     | Applied to me to a considerable degree, or a good part of the time. |
| 4     | Applied to me very much, or most of the time.                       |

See This Photo for How This Looked 😊 !!!

In the past week...

I felt that I had nothing to look forward to.

- Did not apply to me at all
- Applied to me to some degree, or some of the time
- Applied to me to a considerable degree, or a good part of the time
- Applied to me very much, or most of the time

[↻ redo last question](#) 5 / 42

## Part Two(other durations were also recorded & measured on the server's side) :

| Column       | Identifications   |
|--------------|---|
| introelapse  | The time spent on the introduction/landing page (in seconds).   |
| testelapse   | The time spent on all the DASS questions (should be equivalent to the time elapsed on all the individual questions combined). |
| surveyelapse | The time spent answering the rest of the demographic and survey questions.  |

## Part Three (TIPI) :

*The Ten Item Personality Inventory was administered (see Gosling, S. D., Rentfrow, P. J., & Swann, W. B., Jr. (2003). A Very Brief Measure of the Big Five Personality Domains. Journal of Research in Personality, 37, 504-528.) ".*

| Column | Identifications                   |
|--------|-----------------------------------|
| TIPI1  | Extraverted, enthusiastic.        |
| TIPI2  | Critical, quarrelsome.            |
| TIPI3  | Dependable, self-disciplined.     |
| TIPI4  | Anxious, easily upset.            |
| TIPI5  | Open to new experiences, complex. |
| TIPI6  | Reserved, quiet.                  |
| TIPI7  | Sympathetic, warm.                |
| TIPI8  | Disorganized, careless.           |
| TIPI9  | Calm, emotionally stable.         |
| TIPI10 | Conventional, uncreative.         |

The TIPI items were rated "I see myself as:" \_\_\_\_\_ such that

| Scale | Description                |
|-------|----------------------------|
| 1     | Disagree strongly          |
| 2     | Disagree moderately        |
| 3     | Disagree a little          |
| 4     | Neither agree nor disagree |
| 5     | Agree a little             |
| 6     | Agree moderately           |
| 7     | Agree strongly             |

## Part Four (VCL) :

| Column | Identifications |
|--------|-----------------|
| VCL1   | boat            |
| VCL2   | incoherent      |
| VCL3   | pallid          |
| VCL4   | robot           |
| VCL5   | audible         |
| VCL6   | cuivocal        |
| VCL7   | paucity         |
| VCL8   | epistemology    |
| VCL9   | florted         |
| VCL10  | decide          |
| VCL11  | pastiche        |
| VCL12  | verdid          |
| VCL13  | abysmal         |
| VCL14  | lucid           |
| VCL15  | betray          |
| VCL16  | funny           |

Identify Some Answers

| Scale | Description |
|-------|-------------|
| 0     | Unchecked   |
| 1     | Checked     |

## Part Five (General Info) :

A bunch more questions were then asked":

| Column | Identifications |
|--------|-----------------|
|--------|-----------------|

education

How much education have you completed?

*Identify Some Answers*

| Scale | Description           |
|-------|-----------------------|
| 1     | Less than high school |
| 2     | High school           |
| 3     | University degree     |
| 4     | Graduate degree       |

| Column | Identifications                                       |
|--------|---|
| urban  | What type of area did you live when you were a child? |

*Identify Some Answers*

| Scale | Description         |
|-------|---------------------|
| 1     | Rural (countryside) |
| 2     | Suburban            |
| 3     | Urban (town, city)  |

| Column | Identifications      |
|--------|----------------------|
| gender | What is your gender? |

*Identify Some Answers*

| Scale | Description |
|-------|-------------|
| 1     | Male        |
| 2     | Female      |
| 3     | Other       |

| Column | Identifications                  |
|--------|----------------------------------|
| engnat | Is English your native language? |

*Identify Some Answers*

| Scale | Description |
|-------|-------------|
| 1     | Yes         |
| 2     | No          |

| Column | Identifications             |
|--------|-----------------------------|
| age    | How many years old are you? |

| Column | Identifications                     |
|--------|-------------------------------------|
| hand   | What hand do you use to write with? |

*Identify Some Answers*

| Scale | Description |
|-------|-------------|
| 1     | Right       |

|   |      |
|---|------|
| 2 | Left |
| 3 | Both |

| Column   | Identifications        |
|----------|------------------------|
| religion | What is your religion? |

*Identify Some Answers*

| Scale | Description            |
|-------|------------------------|
| 1     | Agnostic               |
| 2     | Atheist                |
| 3     | Buddhist               |
| 4     | Christian (Catholic)   |
| 5     | Christian (Mormon)     |
| 6     | Christian (Protestant) |
| 7     | Christian (Other)      |
| 8     | Hindu                  |
| 9     | Jewish                 |
| 10    | Muslim                 |
| 11    | Sikh                   |
| 12    | Other                  |

| Column      | Identifications                  |
|-------------|----------------------------------|
| orientation | What is your sexual orientation? |

*Identify Some Answers*

| Scale | Description  |
|-------|--------------|
| 1     | Heterosexual |
| 2     | Bisexual     |
| 3     | Homosexual   |
| 4     | Asexual      |
| 5     | Other        |

| Column | Identifications    |
|--------|--------------------|
| race   | What is your race? |

*Identify Some Answers*

| Scale | Description           |
|-------|-----------------------|
| 10    | Asian                 |
| 20    | Arab                  |
| 30    | Black                 |
| 40    | Indigenous Australian |
| 50    | Native American       |
| 60    | White                 |
| 70    | Other                 |

| Column | Identifications |
|--------|-----------------|
|--------|-----------------|

voted

Have you voted in a national election in the past year?

*Identify Some Answers*

| Scale | Description |
|-------|-------------|
| 1     | Yes         |
| 2     | No          |

| Column  | Identifications              |
|---------|------------------------------|
| married | What is your marital status? |

*Identify Some Answers*

| Scale | Description        |
|-------|--------------------|
| 1     | Never married      |
| 2     | Currently married  |
| 3     | Previously married |

| Column     | Identifications  |
|------------|--|
| familysize | Including you, how many children did your mother have? |

| Column | Identifications   |
|--------|---|
| major  | If you attended a university, what was your major (e.g., "psychology", "English", "civil engineering")? |

*The following values were derived from technical information"*

| Column  | Identifications                                    |
|---------|--|
| country | ISO country code of where the user connected from. |

| Column     | Identifications     |
|------------|---------------------|
| screensize | The type of screen. |

*Identify Some Answers*

| Scale | Description                                   |
|-------|---|
| 1     | Device with small screen (phone, etc)         |
| 2     | Device with big screen (laptop, desktop, etc) |

| Column                | Identifications                 |
|-----------------------|---------------------------------|
| uniquenetworklocation | One survey or multiple surveys. |

*Identify Some Answers*

| Scale | Description   |
|-------|---|
| 1     | Only one survey from the user's specific network in the dataset |
| 2     | Multiple surveys submitted from the network of this user        |

**HINT !!!**

(2) does not necessarily imply duplicate records for an individual, as it could be different students at a single school or different members of the same household; and even if 1 there still could be duplicate records from a single individual e.g. if they took it once on their wifi and once on their phone)

| Column | Identifications              |
|--------|------------------------------|
| source | How the user found the test. |

**Identify Some Answers**

| Scale | Description  |
|-------|--|
| 1     | From the front page of the site hosting the survey |
| 2     | From Google  |
| 0     | Other or unknown                                   |

| Column    | Identifications                           |
|-----------|---|
| age group | Detect which age group that user contains |

**Identify Some Answers**

| Scale | Description                    |
|-------|--------------------------------|
| 1     | Under or up to 10 years of age |
| 2     | Between 10 & 16                |
| 3     | Between 17 & 21                |
| 4     | Between 21 & 35                |
| 5     | Between 35 & 48                |
| 6     | Above 49 years                 |

## (2) Import libraries

```
In [1]: 1 !pip install --upgrade tensorflow
Requirement already satisfied: tensorflow in c:\users\scs\anaconda3\lib\site-packages (2.15.0)
WARNING: Ignoring invalid distribution -mpy (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -umpy (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -pencv-python (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution - (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -mpy (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -umpy (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -pencv-python (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution - (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -mpy (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -umpy (c:\users\scs\anaconda3\lib\site-packages)

Requirement already satisfied: tensorflow-intel==2.15.0 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow) (2.15.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (1.12.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (1.60.0)
Requirement already satisfied: google-pasta>=0.1.1 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (0.2.0)
Requirement already satisfied: setuptools in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (61.2.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (4.23.4)
Requirement already satisfied: absl-py>=1.0.0 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (2.0.0)
Requirement already satisfied: ml-dtypes~>0.2.0 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (0.2.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (0.31.0)
Requirement already satisfied: astunparse>=1.6.0 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (1.6.3)
Requirement already satisfied: typing-extensions>=3.6.6 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (4.9.0)
Requirement already satisfied: libclang>=13.0.0 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (13.0.0)
```

```
Requirement already satisfied: h5py>=2.9.0 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (16.0.6)
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (2.15.0)
Requirement already satisfied: six>=1.12.0 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (2.4.0)
Requirement already satisfied: flatbuffers>=23.5.26 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (23.5.26)
Collecting numpy<2.0.0,>=1.23.5
  Using cached numpy-1.26.4-cp39-cp39-win_amd64.whl (15.8 MB)
Requirement already satisfied: gast!=0.5.0,!0=0.5.1,!0=0.5.2,>=0.2.1 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (0.5.4)
Requirement already satisfied: tensorflow<2.16,>=2.15 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (2.15.1)
Requirement already satisfied: keras<2.16,>=2.15.0 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (2.15.0)
Requirement already satisfied: opt-einsum>=2.3.2 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (3.3.0)
Requirement already satisfied: packaging in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (21.3)
Requirement already satisfied: wheel<1.0,>=0.23.0 in c:\users\scs\anaconda3\lib\site-packages (from astunparse>=1.6.0->tensorflow-intel==2.15.0->tensorflow) (0.37.1)
Requirement already satisfied: google-auth<3,>=1.6.3 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (2.25.2)
Requirement already satisfied: tensorflow-data-server<0.8.0,>=0.7.0 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow-intel==2.15.0->tensorflow) (0.7.2)
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (1.2.0)
Requirement already satisfied: markdown>=2.6.8 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (3.3.4)
Requirement already satisfied: werkzeug>=1.0.1 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (2.0.3)
Requirement already satisfied: requests<3,>=2.21.0 in c:\users\scs\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (2.27.1)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in c:\users\scs\anaconda3\lib\site-packages (from google-auth<3,>=1.6.3->tensorflow-intel>=2.15.0->tensorflow) (4.2.2)
Requirement already satisfied: rsa<5,>=3.1.4 in c:\users\scs\anaconda3\lib\site-packages (from google-auth<3,>=1.6.3->tensorflow-intel>=2.15.0->tensorflow) (4.7.2)
Requirement already satisfied: pyasn1-modules>=0.2.1 in c:\users\scs\anaconda3\lib\site-packages (from google-auth<3,>=1.6.3->ensorboard<2.16,>=2.15->tensorflow-intel>=2.15.0->tensorflow) (0.2.8)
Requirement already satisfied: requests-oauthlib>=0.7.0 in c:\users\scs\anaconda3\lib\site-packages (from google-auth-oauthlib<2,>=0.5->ensorboard<2.16,>=2.15->tensorflow-intel>=2.15.0->tensorflow) (1.3.1)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in c:\users\scs\anaconda3\lib\site-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->ensorboard<2.16,>=2.15->tensorflow-intel>=2.15.0->tensorflow) (0.4.8)
Requirement already satisfied: charset-normalizer>~2.0.0 in c:\users\scs\anaconda3\lib\site-packages (from requests<3,>=2.21.0->ensorboard<2.16,>=2.15->tensorflow-intel>=2.15.0->tensorflow) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\scs\anaconda3\lib\site-packages (from requests<3,>=2.21.0->ensorboard<2.16,>=2.15->tensorflow-intel>=2.15.0->tensorflow) (2023.11.17)
Requirement already satisfied: idna<4,>=2.5 in c:\users\scs\anaconda3\lib\site-packages (from requests<3,>=2.21.0->ensorboard<2.16,>=2.15->tensorflow-intel>=2.15.0->tensorflow) (3.3)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\scs\anaconda3\lib\site-packages (from requests<3,>=2.21.0->ensorboard<2.16,>=2.15->tensorflow-intel>=2.15.0->tensorflow) (1.26.9)
Requirement already satisfied: oauthlib>=3.0.0 in c:\users\scs\anaconda3\lib\site-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<2,>=0.5->ensorboard<2.16,>=2.15->tensorflow-intel>=2.15.0->tensorflow) (3.2.2)
Requirement already satisfied: pyparsing>=3.0.5,>=2.0.2 in c:\users\scs\anaconda3\lib\site-packages (from packaging->tensorflow-intel>=2.15.0->tensorflow) (3.0.4)
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 1.22.4
    Uninstalling numpy-1.22.4:
      Successfully uninstalled numpy-1.22.4
Successfully installed numpy-1.26.4
```

```
WARNING: Ignoring invalid distribution -pencv-python (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution - (c:\users\scs\anaconda3\lib\site-packages)
    WARNING: Ignoring invalid distribution -mpy (c:\users\scs\anaconda3\lib\site-packages)
    WARNING: Ignoring invalid distribution -umpy (c:\users\scs\anaconda3\lib\site-packages)
    WARNING: Ignoring invalid distribution -pencv-python (c:\users\scs\anaconda3\lib\site-packages)
    WARNING: Ignoring invalid distribution - (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -mpy (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -umpy (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -pencv-python (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution - (c:\users\scs\anaconda3\lib\site-packages)
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is th
e source of the following dependency conflicts.
daal4py 2021.5.0 requires daal==2021.4.0, which is not installed.
ydata-profiling 4.6.4 requires numpy<1.26,>=1.16.0, but you have numpy 1.26.4 which is incompatible.
scipy 1.7.3 requires numpy<1.23.0,>=1.16.5, but you have numpy 1.26.4 which is incompatible.
WARNING: Ignoring invalid distribution -mpy (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -umpy (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -pencv-python (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution - (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -mpy (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -umpy (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -pencv-python (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution - (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -mpy (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -umpy (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -pencv-python (c:\users\scs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution - (c:\users\scs\anaconda3\lib\site-packages)
```

```
In [2]: 1 # !pip install ydata-profiling
```

```
In [3]: 1 # for analysis
2 import numpy as np
3 import pandas as pd
4
5 # for creating a demo profiling
6 # from ydata_profiling import ProfileReport
7 # from ydata_profiling.utils.cache import cache_file
```

```

9 # for visualization
10 import matplotlib.pyplot as plt
11 import seaborn as sns
12
13 # for mathematical & random operations
14 import math
15 import random
16
17 # for framework operations
18 import tensorflow as tf
19 from tensorflow.keras import Sequential
20 from tensorflow.keras.layers import Dense
21 from tensorflow.keras.activations import relu, linear, softmax
22 from tensorflow.keras.optimizers import Adam, SGD, RMSprop
23 from tensorflow.keras.models import save_model as sm
24 from tensorflow.keras.models import load_model as lm
25
26 # for data rescaling
27 from sklearn.preprocessing import MinMaxScaler
28
29 # for splitting data
30 from sklearn.model_selection import train_test_split, cross_val_score
31
32 # for building model
33 from lazypredict.Supervised import LazyClassifier
34 from sklearn.neighbors import KNeighborsClassifier
35 from sklearn.ensemble import RandomForestClassifier
36 from sklearn.tree import DecisionTreeClassifier
37 from sklearn.linear_model import LogisticRegression
38 from sklearn.naive_bayes import GaussianNB
39 from sklearn.svm import SVC, SVR
40
41 #for evaluation
42 from sklearn.metrics import confusion_matrix, plot_confusion_matrix, accuracy_score, precision_score, recall_score, f1_score,
43
44 #for check high bias & high variance
45 from sklearn.model_selection import learning_curve
46 from sklearn.model_selection import ShuffleSplit
47
48 #for saving model
49 import pickle
50 import joblib

```

C:\Users\scs\anaconda3\lib\site-packages\scipy\\_\_init\_\_.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of Scipy (detected version 1.26.4)  
 warnings.warn(f"A NumPy version >={np\_minversion} and <{np\_maxversion}"

WARNING:tensorflow:From C:\Users\scs\anaconda3\lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse\_softmax\_crossentropy is deprecated. Please use tf.compat.v1.losses.sparse\_softmax\_cross\_entropy instead.

### (3) Reading & Showing Data

```

In [4]: 1 pd.set_option('display.max_columns',172)
2 path = 'D:/COURSES/Programming Language/python/data_science_track/projects/NTI/Graduation_Project/Datasets/Depression Anxiety'
3 df = pd.read_csv(path, delimiter='\t')
4 pd.set_option('display.max_columns',172)
5 df

```

|       | Q1A | Q1I | Q1E   | Q2A | Q2I | Q2E  | Q3A | Q3I | Q3E   | Q4A | Q4I | Q4E  | Q5A | Q5I | Q5E  | Q6A | Q6I | Q6E   | Q7A | Q7I | Q7E  | Q8A | Q8I | Q8E  | Q9A | Q9I |
|-------|-----|-----|-------|-----|-----|------|-----|-----|-------|-----|-----|------|-----|-----|------|-----|-----|-------|-----|-----|------|-----|-----|------|-----|-----|
| 0     | 4   | 28  | 3890  | 4   | 25  | 2122 | 2   | 16  | 1944  | 4   | 8   | 2044 | 4   | 34  | 2153 | 4   | 33  | 2416  | 4   | 10  | 2818 | 4   | 13  | 2259 | 2   | 21  |
| 1     | 4   | 2   | 8118  | 1   | 36  | 2890 | 2   | 35  | 4777  | 3   | 28  | 3090 | 4   | 10  | 5078 | 4   | 40  | 2790  | 3   | 18  | 3408 | 4   | 1   | 8342 | 3   | 37  |
| 2     | 3   | 7   | 5784  | 1   | 33  | 4373 | 4   | 41  | 3242  | 1   | 13  | 6470 | 4   | 11  | 3927 | 3   | 9   | 3704  | 1   | 17  | 4550 | 3   | 5   | 3021 | 2   | 32  |
| 3     | 2   | 23  | 5081  | 3   | 11  | 6837 | 2   | 37  | 5521  | 1   | 27  | 4556 | 3   | 28  | 3269 | 3   | 26  | 3231  | 4   | 2   | 7138 | 2   | 19  | 3079 | 3   | 31  |
| 4     | 2   | 36  | 3215  | 2   | 13  | 7731 | 3   | 5   | 4156  | 4   | 10  | 2802 | 4   | 2   | 5628 | 2   | 9   | 6522  | 4   | 34  | 2374 | 4   | 11  | 3054 | 4   | 7   |
| ...   | ... | ... | ...   | ... | ... | ...  | ... | ... | ...   | ... | ... | ...  | ... | ... | ...  | ... | ... | ...   | ... | ... | ...  | ... | ... | ...  | ... |     |
| 39770 | 2   | 31  | 3287  | 1   | 5   | 2216 | 3   | 29  | 3895  | 2   | 37  | 2767 | 3   | 39  | 1745 | 2   | 4   | 6424  | 1   | 36  | 3737 | 3   | 24  | 1977 | 1   | 18  |
| 39771 | 3   | 14  | 4792  | 4   | 41  | 2604 | 3   | 15  | 2668  | 4   | 33  | 4609 | 3   | 17  | 2434 | 4   | 26  | 2503  | 4   | 34  | 2598 | 4   | 11  | 2133 | 3   | 13  |
| 39772 | 2   | 1   | 25147 | 1   | 4   | 4555 | 2   | 14  | 3388  | 1   | 27  | 2753 | 1   | 6   | 5455 | 1   | 41  | 4938  | 1   | 24  | 3738 | 1   | 40  | 3820 | 2   | 28  |
| 39773 | 3   | 36  | 4286  | 1   | 34  | 2736 | 2   | 10  | 5968  | 2   | 20  | 5655 | 3   | 9   | 2296 | 3   | 16  | 2627  | 3   | 32  | 3143 | 4   | 2   | 3164 | 3   | 19  |
| 39774 | 2   | 28  | 32251 | 1   | 22  | 3317 | 2   | 4   | 11734 | 1   | 19  | 4659 | 4   | 32  | 4236 | 2   | 1   | 33800 | 1   | 3   | 7855 | 1   | 34  | 3151 | 1   | 31  |

39775 rows × 172 columns

```

In [5]: 1 df.columns

```

```

Out[5]: Index(['Q1A', 'Q1I', 'Q1E', 'Q2A', 'Q2I', 'Q2E', 'Q3A', 'Q3I', 'Q3E', 'Q4A', 'Q4I', 'Q4E', 'Q5A', 'Q5I', 'Q5E', 'Q6A', 'Q6I', 'Q6E', 'Q7A', 'Q7I', 'Q7E', 'Q8A', 'Q8I', 'Q8E', 'Q9A', 'Q9I'],
              ...,
              'screensize', 'uniquenetworklocation', 'hand', 'religion',
              'orientation', 'race', 'voted', 'married', 'familysize', 'major'],
              dtype='object', length=172)

```

```

In [6]: 1 print(f'The Shape of DataFrame: {df.shape}')
2 print(f'The Number of Rows: {df.shape[0]}')
3 print(f'The Number of Columns: {df.shape[1]}')

```

The Shape of DataFrame: (39775, 172)  
 The Number of Rows: 39775  
 The Number of Columns: 172

```
The Number of Rows: 39775  
The Number of Columns: 172
```

```
In [7]: 1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 39775 entries, 0 to 39774  
Columns: 172 entries, Q1A to major  
dtypes: int64(170), object(2)  
memory usage: 52.2+ MB
```

```
In [8]: 1 df.select_dtypes(include=['object']).columns
```

```
Out[8]: Index(['country', 'major'], dtype='object')
```

```
In [9]: 1 description = df[['age', 'familysize']].describe()  
2 np.ceil(description)
```

```
Out[9]:
```

|       | age      | familysize |
|-------|----------|------------|
| count | 39775.00 | 39775.00   |
| mean  | 24.00    | 4.00       |
| std   | 22.00    | 3.00       |
| min   | 13.00    | 0.00       |
| 25%   | 18.00    | 2.00       |
| 50%   | 21.00    | 3.00       |
| 75%   | 25.00    | 4.00       |
| max   | 1998.00  | 133.00     |

```
In [10]: 1 np.array(df)
```

```
Out[10]: array([[4, 28, 3890, ..., 1, 2, nan],  
[4, 2, 8118, ..., 1, 4, nan],  
[3, 7, 5784, ..., 1, 3, nan],  
...,  
[2, 1, 25147, ..., 2, 3, 'Computer Science'],  
[3, 36, 4286, ..., 1, 2, 'History'],  
[2, 28, 32251, ..., 1, 4, 'Cognitive Science']], dtype=object)
```

```
In [11]: 1 df.ndim
```

```
Out[11]: 2
```

### (3) Data Analysis

```
In [12]: 1 # report=ProfileReport(df, title="Demo Profiling", explorative=True)  
2 # report.to_file("D:/COURSES/Programming Language/python/data_science_track/projects/NTI/Graduation_Project/Datasets/Depressi
```

```
In [13]: 1 def get_value_counts_per_column_range(df, column_range):  
2     value_counts_per_column = {}  
3     for column in df.columns[column_range]:  
4         value_counts = df[column].value_counts()  
5         value_counts_per_column[column] = value_counts  
6     return value_counts_per_column
```

```
In [14]: 1 def plot_null_values(df, columns_to_check):  
2     plt.figure(figsize=(20, 10))  
3     null_values = df[columns_to_check].isnull()  
4     print("Null values in the specified columns:")  
5     print(null_values)  
6     sns.heatmap(null_values, yticklabels=False, cmap='viridis')  
7     plt.xticks(ticks=range(len(columns_to_check)), labels=columns_to_check, rotation=45, ha='right')  
8     plt.title('Null Values Distribution in Dataframe')  
9     plt.show()
```

```
In [15]: 1 def sum_null_values(dataframe, columns_to_sum=None):  
2     nan_sum_per_column = {}  
3     if columns_to_sum is None:  
4         columns_to_sum = dataframe.columns  
5     for column in columns_to_sum:  
6         nan_sum = dataframe[column].isna().sum()  
7         nan_sum_per_column[column] = nan_sum  
8     return nan_sum_per_column
```

#### Part One (Q) :

##### NOTE !!!

The Columns of Part One(Q) That Contain I & E that Determine The Position of Question and The Time That The User Taken to Solve are not Needed for The User so That We Will Drop Them.

```
In [16]: 1 removedcolumns=[f'Q{i}I' for i in range(1,43)]  
2 removedcolumns.extend([f'Q{i}E' for i in range(1,43)])  
3 df=df.drop(removedcolumns ,axis=1)
```

```

4 df.head()
Out[16]:
   Q1A  Q2A  Q3A  Q4A  Q5A  Q6A  Q7A  Q8A  Q9A  Q10A  Q11A  Q12A  Q13A  Q14A  Q15A  Q16A  Q17A  Q18A  Q19A  Q20A  Q21A  Q22A  Q23A  Q24A
0    4    4    4    2    4    4    4    4    2    1    4    4    4    4    4    4    4    3    4    3    1    4    4
1    4    1    2    3    4    4    3    4    3    2    2    2    4    4    4    3    3    4    2    1    1    2    3    1    2
2    3    1    4    1    4    3    1    3    2    4    2    1    4    1    4    4    4    2    2    1    4    3    2    4
3    2    3    2    1    3    3    4    2    3    3    2    1    1    4    2    2    2    3    1    1    2    1    1    1    1
4    2    2    3    4    4    2    4    4    4    3    2    4    4    4    4    3    4    4    4    4    3    3    4    2

In [17]: 1 df.shape
Out[17]: (39775, 88)

In [18]: 1 for i in df.columns[0:42]:
2     unique_values = df[i].unique()
3     print(f"Unique values in {i}: {unique_values}")

Unique values in Q1A: [4 3 2 1]
Unique values in Q2A: [4 1 3 2]
Unique values in Q3A: [2 4 3 1]
Unique values in Q4A: [4 3 1 2]
Unique values in Q5A: [4 3 1 2]
Unique values in Q6A: [4 3 2 1]
Unique values in Q7A: [4 3 1 2]
Unique values in Q8A: [4 3 2 1]
Unique values in Q9A: [2 3 4 1]
Unique values in Q10A: [1 2 4 3]
Unique values in Q11A: [4 2 1 3]
Unique values in Q12A: [4 2 1 3]
Unique values in Q13A: [4 1 2 3]
Unique values in Q14A: [4 1 3 2]
Unique values in Q15A: [4 3 2 1]
Unique values in Q16A: [4 3 2 1]
Unique values in Q17A: [3 4 2 1]
Unique values in Q18A: [4 2 1 3]
Unique values in Q19A: [3 1 2 4]
Unique values in Q20A: [3 1 2 4]
Unique values in Q21A: [1 2 4 3]
Unique values in Q22A: [4 3 1 2]
Unique values in Q23A: [4 1 2 3]
Unique values in Q24A: [4 2 1 3]
Unique values in Q25A: [4 2 1 3]
Unique values in Q26A: [4 3 1 2]
Unique values in Q27A: [4 3 2 1]
Unique values in Q28A: [3 4 1 2]
Unique values in Q29A: [4 3 2 1]
Unique values in Q30A: [2 3 4 1]
Unique values in Q31A: [4 2 3 1]
Unique values in Q32A: [4 3 1 2]
Unique values in Q33A: [2 3 1 4]
Unique values in Q34A: [3 2 4 1]
Unique values in Q35A: [4 2 3 1]
Unique values in Q36A: [4 3 1 2]
Unique values in Q37A: [1 4 2 3]
Unique values in Q38A: [2 4 1 3]
Unique values in Q39A: [4 2 3 1]
Unique values in Q40A: [3 1 2 4]
Unique values in Q41A: [4 2 1 3]
Unique values in Q42A: [4 2 3 1]

In [19]: 1 selected_column_range = range(0, 42)
2 result = get_value_counts_per_column_range(df, selected_column_range)
3 for column, counts in result.items():
4     print(f"Value counts for {column}:\n{counts}\n")

Value counts for Q1A:
2    13320
4    10393
3    9958
1    6104
Name: Q1A, dtype: int64

Value counts for Q2A:
1    14385
2    11504
4    7351
3    6535
Name: Q2A, dtype: int64

Value counts for Q3A:
2    14062
1    11563
3    7744
4    6406
Name: Q3A, dtype: int64

Value counts for Q4A:
1    17757
2    11128
3    6005
4    4885
Name: Q4A, dtype: int64

Value counts for Q5A:
2    12794
4    9788
3    9179
1    8014
Name: Q5A, dtype: int64
...

```

```
value counts for Q6A:  
2    13291  
4    9649  
3    9512  
1    7323  
Name: Q6A, dtype: int64  
  
Value counts for Q7A:  
1    18092  
2    11333  
3    5594  
4    4756  
Name: Q7A, dtype: int64  
  
Value counts for Q8A:  
2    13471  
3    9241  
4    8977  
1    8086  
Name: Q8A, dtype: int64  
  
Value counts for Q9A:  
2    11788  
4    11734  
3    9709  
1    6544  
Name: Q9A, dtype: int64  
  
Value counts for Q10A:  
2    11007  
1    10625  
4    10274  
3    7869  
Name: Q10A, dtype: int64  
  
Value counts for Q11A:  
4    13598  
2    11136  
3    9898  
1    5143  
Name: Q11A, dtype: int64  
  
Value counts for Q12A:  
2    12601  
3    9439  
1    9326  
4    8409  
Name: Q12A, dtype: int64  
  
Value counts for Q13A:  
4    13922  
2    11200  
3    9007  
1    5646  
Name: Q13A, dtype: int64  
  
Value counts for Q14A:  
2    12535  
4    10842  
3    8897  
1    7501  
Name: Q14A, dtype: int64  
  
Value counts for Q15A:  
1    19587  
2    11262  
3    5150  
4    3776  
Name: Q15A, dtype: int64  
  
Value counts for Q16A:  
2    11696  
4    10579  
1    8996  
3    8504  
Name: Q16A, dtype: int64  
  
Value counts for Q17A:  
4    13509  
2    9800  
1    8644  
3    7822  
Name: Q17A, dtype: int64  
  
Value counts for Q18A:  
2    12789  
3    9277  
4    9142  
1    8567  
Name: Q18A, dtype: int64  
  
Value counts for Q19A:  
1    18506  
2    10370  
4    5471  
3    5428  
Name: Q19A, dtype: int64  
  
Value counts for Q20A:  
1    11946  
2    11489  
4    8455  
3    7885  
Name: Q20A, dtype: int64  
  
Value counts for Q21A:
```

```
1    12694
2    10308
4    9826
3    6947
Name: Q21A, dtype: int64

Value counts for Q22A:
2    14327
1    9448
3    8850
4    7150
Name: Q22A, dtype: int64

Value counts for Q23A:
1    25111
2    9139
3    3349
4    2176
Name: Q23A, dtype: int64

Value counts for Q24A:
2    13790
3    8919
1    8555
4    8511
Name: Q24A, dtype: int64

Value counts for Q25A:
1    13569
2    11924
3    7664
4    6618
Name: Q25A, dtype: int64

Value counts for Q26A:
2    11979
4    11573
3    9636
1    6587
Name: Q26A, dtype: int64

Value counts for Q27A:
2    12889
4    10634
3    9670
1    6582
Name: Q27A, dtype: int64

Value counts for Q28A:
1    12734
2    12495
3    7716
4    6830
Name: Q28A, dtype: int64

Value counts for Q29A:
2    12280
4    11338
3    9733
1    6424
Name: Q29A, dtype: int64

Value counts for Q30A:
2    12535
1    10000
3    8875
4    8365
Name: Q30A, dtype: int64

Value counts for Q31A:
2    14022
1    9254
3    8756
4    7743
Name: Q31A, dtype: int64

Value counts for Q32A:
2    14441
3    9653
4    7919
1    7762
Name: Q32A, dtype: int64

Value counts for Q33A:
2    13146
3    9498
1    9095
4    8036
Name: Q33A, dtype: int64

Value counts for Q34A:
4    13064
2    10268
1    8677
3    7766
Name: Q34A, dtype: int64

Value counts for Q35A:
2    15373
1    9317
3    8810
4    6275
Name: Q35A, dtype: int64

Value counts for Q36A:
1    11517
```

```

1 12042
2 7974
3 7242
Name: Q36A, dtype: int64

Value counts for Q37A:
1 11580
2 11362
4 9605
3 7228
Name: Q37A, dtype: int64

Value counts for Q38A:
1 12551
4 10746
2 9797
3 6681
Name: Q38A, dtype: int64

Value counts for Q39A:
2 14405
3 9657
4 8040
1 7673
Name: Q39A, dtype: int64

Value counts for Q40A:
4 12219
2 10788
3 9109
1 7659
Name: Q40A, dtype: int64

Value counts for Q41A:
1 17323
2 11617
3 5693
4 5142
Name: Q41A, dtype: int64

Value counts for Q42A:
2 12886
4 11304
3 10014
1 5571
Name: Q42A, dtype: int64

```

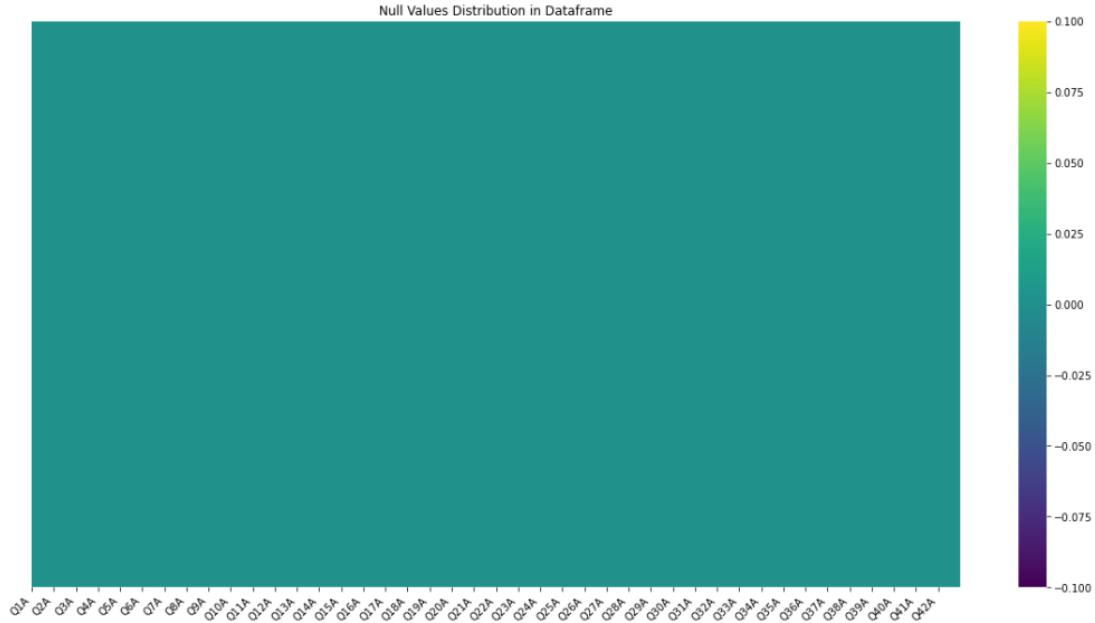
```
In [20]: 1 columns_to_check = df.columns[0:42]
2 plot_null_values(df, columns_to_check)
```

| Null values in the specified columns: |       |       |       |       |       |       |       |       |       |       |   |
|---------------------------------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|---|
|                                       | Q1A   | Q2A   | Q3A   | Q4A   | Q5A   | Q6A   | Q7A   | Q8A   | Q9A   | Q10A  | \ |
| 0                                     | False |   |
| 1                                     | False |   |
| 2                                     | False |   |
| 3                                     | False |   |
| 4                                     | False |   |
| ...                                   | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   |   |
| 39770                                 | False |   |
| 39771                                 | False |   |
| 39772                                 | False |   |
| 39773                                 | False |   |
| 39774                                 | False |   |
|                                       | Q11A  | Q12A  | Q13A  | Q14A  | Q15A  | Q16A  | Q17A  | Q18A  | Q19A  | Q20A  | \ |
| 0                                     | False |   |
| 1                                     | False |   |
| 2                                     | False |   |
| 3                                     | False |   |
| 4                                     | False |   |
| ...                                   | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   |   |
| 39770                                 | False |   |
| 39771                                 | False |   |
| 39772                                 | False |   |
| 39773                                 | False |   |
| 39774                                 | False |   |
|                                       | Q21A  | Q22A  | Q23A  | Q24A  | Q25A  | Q26A  | Q27A  | Q28A  | Q29A  | Q30A  | \ |
| 0                                     | False |   |
| 1                                     | False |   |
| 2                                     | False |   |
| 3                                     | False |   |
| 4                                     | False |   |
| ...                                   | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   |   |
| 39770                                 | False |   |
| 39771                                 | False |   |
| 39772                                 | False |   |
| 39773                                 | False |   |
| 39774                                 | False |   |
|                                       | Q31A  | Q32A  | Q33A  | Q34A  | Q35A  | Q36A  | Q37A  | Q38A  | Q39A  | Q40A  | \ |
| 0                                     | False |   |
| 1                                     | False |   |
| 2                                     | False |   |
| 3                                     | False |   |
| 4                                     | False |   |
| ...                                   | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   |   |
| 39770                                 | False |   |
| 39771                                 | False |   |
| 39772                                 | False |   |
| 39773                                 | False |   |
| 39774                                 | False |   |

Q41A Q42A

```
0      False  False
1      False  False
2      False  False
3      False  False
4      False  False
...
39770  False  False
39771  False  False
39772  False  False
39773  False  False
39774  False  False
```

[39775 rows x 42 columns]



```
In [21]: 1 columns_to_sum = df.columns[0:42]
2 sum_null_values(df,columns_to_sum)
```

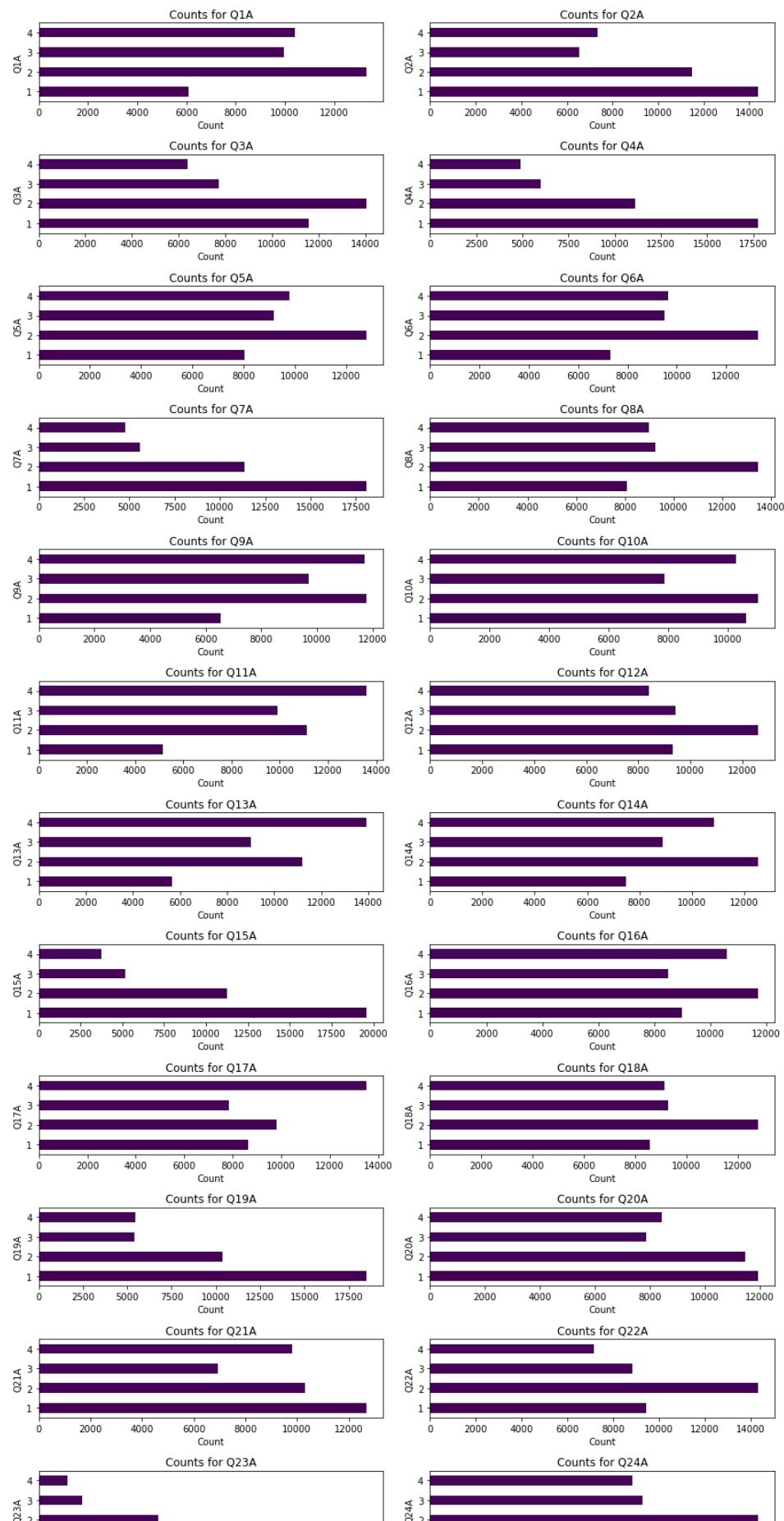
```
Out[21]: {'Q1A': 0,
 'Q2A': 0,
 'Q3A': 0,
 'Q4A': 0,
 'Q5A': 0,
 'Q6A': 0,
 'Q7A': 0,
 'Q8A': 0,
 'Q9A': 0,
 'Q10A': 0,
 'Q11A': 0,
 'Q12A': 0,
 'Q13A': 0,
 'Q14A': 0,
 'Q15A': 0,
 'Q16A': 0,
 'Q17A': 0,
 'Q18A': 0,
 'Q19A': 0,
 'Q20A': 0,
 'Q21A': 0,
 'Q22A': 0,
 'Q23A': 0,
 'Q24A': 0,
 'Q25A': 0,
 'Q26A': 0,
 'Q27A': 0,
 'Q28A': 0,
 'Q29A': 0,
 'Q30A': 0,
 'Q31A': 0,
 'Q32A': 0,
 'Q33A': 0,
 'Q34A': 0,
 'Q35A': 0,
 'Q36A': 0,
 'Q37A': 0,
 'Q38A': 0,
 'Q39A': 0,
 'Q40A': 0,
 'Q41A': 0,
 'Q42A': 0}
```

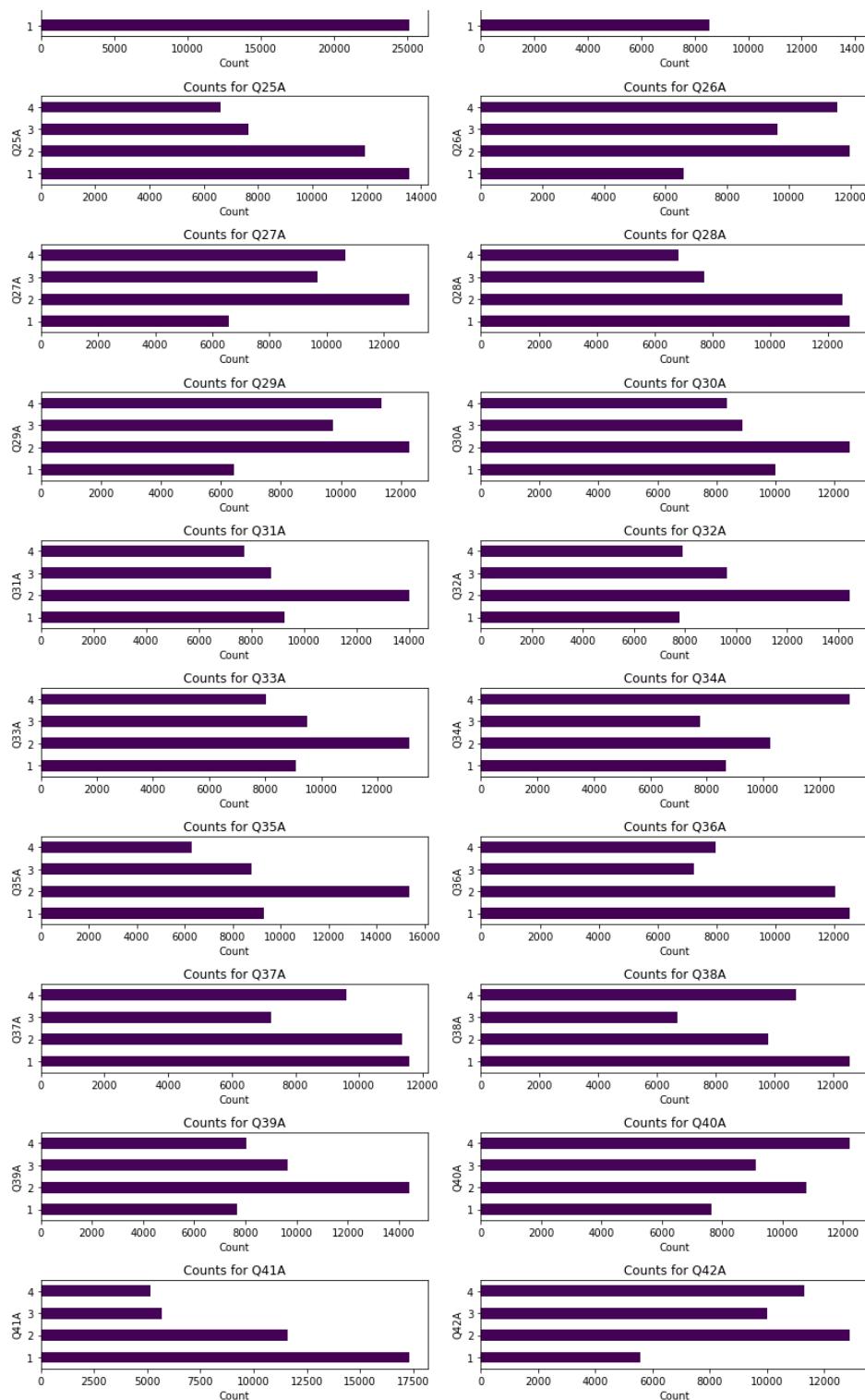
```
In [22]: 1 columns_to_plot = df.columns[0:42]
2 num_columns = 2
3 num_rows = math.ceil(len(columns_to_plot) / num_columns)
4 fig, axes = plt.subplots(nrows=num_rows, ncols=num_columns, figsize=(12, 2 * num_rows))
5 for i, column in enumerate(columns_to_plot):
6     row = i // num_columns
7     col = i % num_columns
8     ax = axes[row, col]
9     counts = df[column].value_counts().sort_index()
10    colors = plt.cm.viridis.colors[:len(counts)]
11    counts.plot(kind='barh', ax=ax, color=colors)
```

```

12     ax.set_title(f'Counts for {column}')
13     ax.set_ylabel(column)
14     ax.set_xlabel('Count')
15     for i in range(len(columns_to_plot), num_rows * num_columns):
16         fig.delaxes(axes.flatten()[i])
17     plt.tight_layout()
18     plt.show()

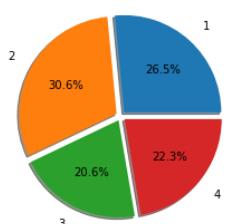
```





```
In [23]: 1 columns_to_plot = df.columns[0:42]
2 unique_values = [1, 2, 3, 4]
3 overall_value_counts = df[columns_to_plot].stack().value_counts()
4 total_count = overall_value_counts.sum()
5 overall_percentages = [overall_value_counts.get(val, 0) / total_count * 100 for val in unique_values]
6 plt.pie(overall_percentages, labels=unique_values, explode=[0.05, 0.05, 0.05, 0.05], autopct='%1.1f%%', shadow=True, startangle=90)
7 plt.title('Overall Distribution of Values of Part One(QA)')
8 plt.show()
```

Overall Distribution of Values of Part One(QA)





## Part Two(other durations were also recorded & measured on the server's side) :

### NOTE !!!

The Columns of Part Two That measured on the server's side, User Dont Use Them so We Will Dropped **introelapse** & **testelapse** & **surveyelapse**

```
In [26]: 1 columns_to_drop = ['introelapse', 'testelapse', 'surveyelapse']
2
3 if all(col in df.columns for col in columns_to_drop):
4     df = df.drop(columns=columns_to_drop, axis=1)
5 else:
6     print("'introelapse', 'testelapse', or 'surveyelapse' not found in DataFrame columns")
```

```
In [27]: 1 df.shape[1]
```

```
Out[27]: 85
```

```
In [28]: 1 df
```

```
Out[28]:
```

| Q1A   | Q2A | Q3A | Q4A | Q5A | Q6A | Q7A | Q8A | Q9A | Q10A | Q11A | Q12A | Q13A | Q14A | Q15A | Q16A | Q17A | Q18A | Q19A | Q20A | Q21A | Q22A | Q23A |     |   |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|------|-----|---|
| 0     | 4   | 4   | 2   | 4   | 4   | 4   | 4   | 4   | 2    | 1    | 4    | 4    | 4    | 4    | 4    | 4    | 3    | 4    | 3    | 3    | 1    | 4    | 4   |   |
| 1     | 4   | 1   | 2   | 3   | 4   | 4   | 3   | 4   | 3    | 2    | 2    | 2    | 4    | 4    | 4    | 3    | 3    | 4    | 2    | 1    | 1    | 2    | 3   | 1 |
| 2     | 3   | 1   | 4   | 1   | 4   | 3   | 1   | 3   | 2    | 4    | 2    | 1    | 4    | 1    | 4    | 4    | 4    | 2    | 2    | 1    | 4    | 3    | 2   |   |
| 3     | 2   | 3   | 2   | 1   | 3   | 3   | 4   | 2   | 3    | 3    | 2    | 1    | 1    | 1    | 4    | 2    | 2    | 3    | 1    | 1    | 2    | 1    | 1   |   |
| 4     | 2   | 2   | 3   | 4   | 4   | 2   | 4   | 4   | 4    | 3    | 2    | 4    | 4    | 4    | 4    | 3    | 4    | 4    | 4    | 4    | 3    | 3    | 4   |   |
| ...   | ... | ... | ... | ... | ... | ... | ... | ... | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ... |   |
| 39770 | 2   | 1   | 3   | 2   | 3   | 2   | 1   | 3   | 1    | 4    | 2    | 2    | 4    | 2    | 1    | 3    | 4    | 2    | 4    | 3    | 4    | 3    | 1   |   |
| 39771 | 3   | 4   | 3   | 4   | 3   | 4   | 4   | 4   | 3    | 4    | 4    | 4    | 4    | 4    | 4    | 4    | 3    | 3    | 4    | 4    | 4    | 4    | 3   |   |
| 39772 | 2   | 1   | 2   | 1   | 1   | 1   | 1   | 1   | 2    | 1    | 1    | 1    | 2    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1   |   |
| 39773 | 3   | 1   | 2   | 2   | 3   | 3   | 3   | 4   | 3    | 1    | 3    | 3    | 4    | 4    | 1    | 1    | 1    | 2    | 3    | 1    | 2    | 2    | 4   |   |
| 39774 | 2   | 1   | 2   | 1   | 4   | 2   | 1   | 1   | 1    | 1    | 3    | 1    | 1    | 1    | 4    | 4    | 3    | 4    | 2    | 1    | 1    | 1    | 1   |   |

39775 rows × 85 columns

## Part Three (TIPI) :

```
In [29]: 1 df.columns.get_loc('TIPI1')
Out[29]: 44

In [30]: 1 df.columns.get_loc('TIPI10')
Out[30]: 53

In [31]: 1 for i in df.columns[44:54]:
2     unique_values = df[i].unique()
3     print(f"Unique values in {i}: {unique_values}")

Unique values in TIPI1: [1 6 2 7 5 3 4 0]
Unique values in TIPI2: [5 1 6 4 7 3 2 0]
Unique values in TIPI3: [7 4 2 3 6 5 1 0]
Unique values in TIPI4: [7 2 4 6 1 5 0 3]
Unique values in TIPI5: [7 5 6 3 2 1 4 0]
Unique values in TIPI6: [7 4 6 5 2 1 3 0]
Unique values in TIPI7: [7 5 6 1 4 3 2 0]
Unique values in TIPI8: [5 7 1 6 2 3 4 0]
Unique values in TIPI9: [1 3 6 5 2 7 4 0]
Unique values in TIPI10: [1 5 2 3 7 4 6 0]

In [32]: 1 selected_column_range = range(44, 54)
2 result = get_value_counts_per_column_range(df, selected_column_range)
3 for column, counts in result.items():
4     print(f"Value counts for {column}:\n{counts}\n")

Value counts for TIPI1:
5    8420
1    6229
6    6046
4    5877
2    5603
3    4633
7    2482
0    485
Name: TIPI1, dtype: int64

Value counts for TIPI2:
5    18373
6    6990
4    6171
3    4326
2    4040
1    3985
7    3320
0    570
Name: TIPI2, dtype: int64

Value counts for TIPI3:
6    18230
5    9121
7    6242
4    4516
3    3981
2    2843
1    2252
0    590
Name: TIPI3, dtype: int64

Value counts for TIPI4:
7    11360
6    9561
5    8751
4    2720
2    2444
3    2434
1    2042
0    463
Name: TIPI4, dtype: int64

Value counts for TIPI5:
6    9697
5    9525
7    7682
4    5140
3    3341
2    2226
1    1647
0    517
Name: TIPI5, dtype: int64

Value counts for TIPI6:
7    9626
6    8894
5    7733
4    4943
3    3475
2    2748
1    2666
0    490
Name: TIPI6, dtype: int64

Value counts for TIPI7:
6    10997
7    10007
5    9097
4    4421
3    2270
```

```

2    1367
1    1015
0     601
Name: TIPI7, dtype: int64

Value counts for TIPI8:
5    9300
6    6369
7    5841
4    4577
3    4446
2    4445
1    4138
0     659
Name: TIPI8, dtype: int64

Value counts for TIPI9:
4    6837
3    6819
2    6401
5    6250
1    5507
6    4943
7    2591
0     427
Name: TIPI9, dtype: int64

Value counts for TIPI10:
4    7665
5    6653
3    6184
2    5714
1    5404
6    4106
7    3494
0     555
Name: TIPI10, dtype: int64

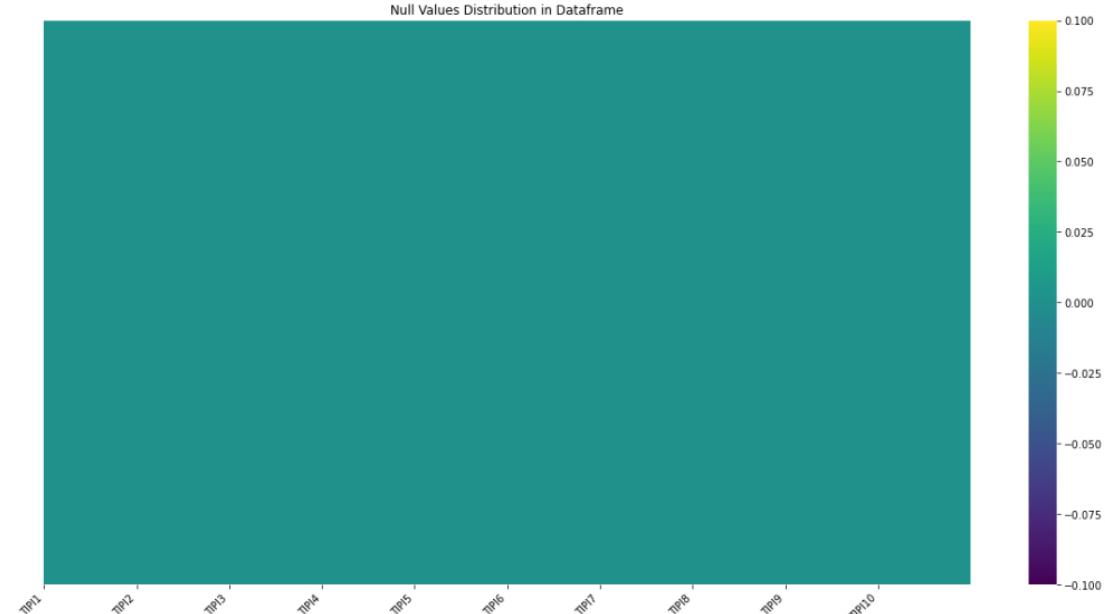
```

```
In [33]: 1 columns_to_check = df.columns[44:54]
2 plot_null_values(df, columns_to_check)
```

Null values in the specified columns:

|       | TIPI1 | TIPI2 | TIPI3 | TIPI4 | TIPI5 | TIPI6 | TIPI7 | TIPI8 | TIPI9 | TIPI10 |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|--------|
| 0     | False  |
| 1     | False  |
| 2     | False  |
| 3     | False  |
| 4     | False  |
| ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...    |
| 39770 | False  |
| 39771 | False  |
| 39772 | False  |
| 39773 | False  |
| 39774 | False  |

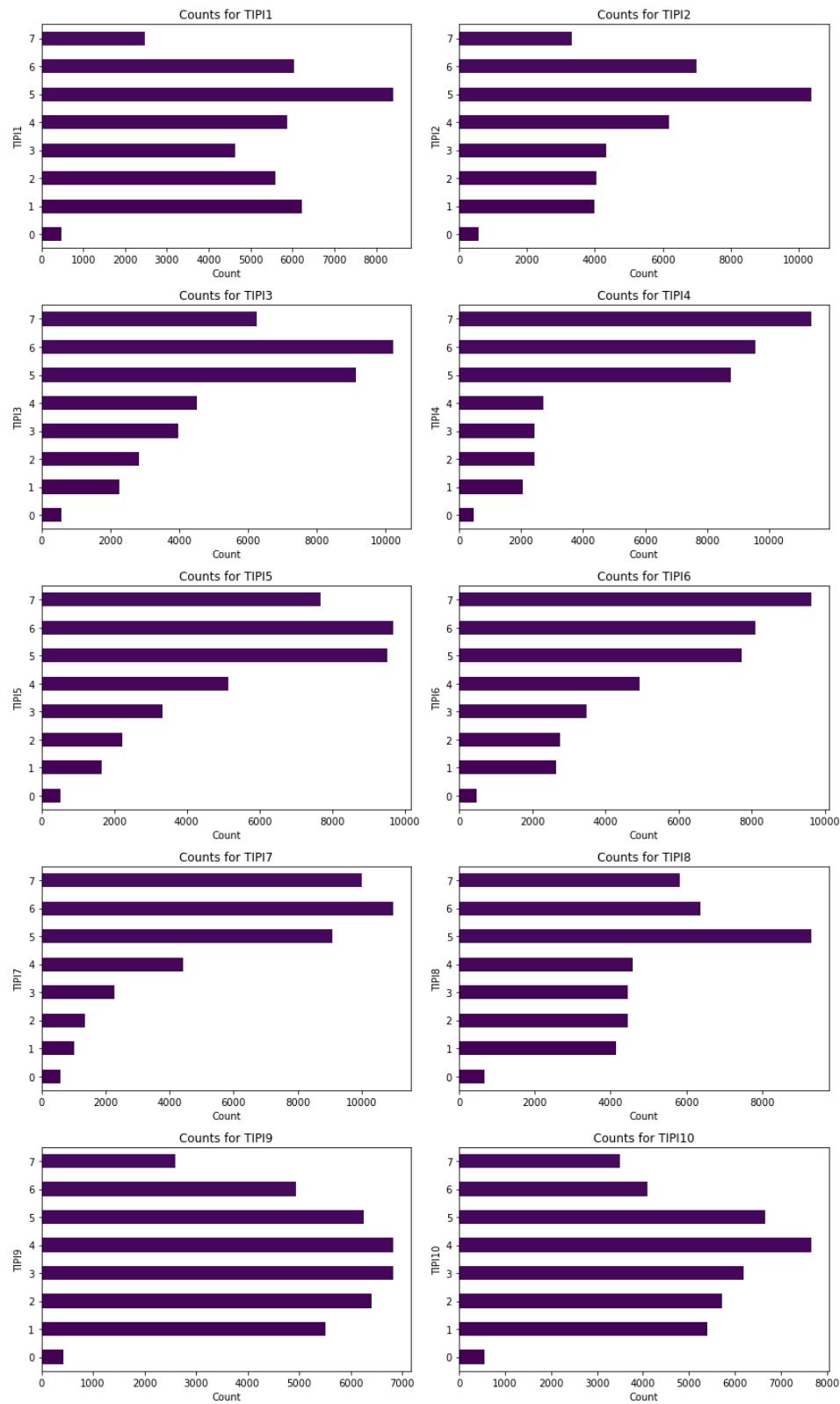
[39775 rows x 10 columns]



```
In [34]: 1 columns_to_sum = df.columns[44:54]
2 sum_null_values(df,columns_to_sum)
```

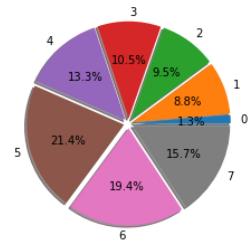
```
Out[34]: {'TIPI1': 0,
 'TIPI2': 0,
 'TIPI3': 0,
 'TIPI4': 0,
 'TIPI5': 0,
 'TIPI6': 0,
 'TIPI7': 0,
 'TIPI8': 0,
 'TIPI9': 0,
 'TIPI10': 0}
```

```
In [35]: 1 columns_to_plot = df.columns[44:54]
2 num_columns = 2
3 num_rows = math.ceil(len(columns_to_plot) / num_columns)
4 fig, axes = plt.subplots(nrows=num_rows, ncols=num_columns, figsize=(12, 4 * num_rows))
5 for i, column in enumerate(columns_to_plot):
6     row = i // num_columns
7     col = i % num_columns
8     ax = axes[row, col]
9     counts = df[column].value_counts().sort_index()
10    colors = plt.cm.viridis.colors[:len(counts)]
11    counts.plot(kind='barh', ax=ax, color=colors)
12    ax.set_title(f'Counts for {column}')
13    ax.set_ylabel(column)
14    ax.set_xlabel('Count')
15 for i in range(len(columns_to_plot), num_rows * num_columns):
16     fig.delaxes(axes.flatten()[i])
17 plt.tight_layout()
18 plt.show()
```



```
In [36]: 1 columns_to_plot = df.columns[44:54]
2 unique_values = [0, 1, 2, 3, 4, 5, 6, 7]
3 overall_value_counts = df[columns_to_plot].stack().value_counts()
4 total_count = overall_value_counts.sum()
5 overall_percentages = [overall_value_counts.get(val, 0) / total_count * 100 for val in unique_values]
6 explode = [0.05] * len(unique_values)
7 plt.pie(overall_percentages, labels=unique_values, explode=explode, autopct='%.1f%%', shadow=True, startangle=0, labellimit=1, title='Overall Distribution of Values of Part Three(TIPI)')
8 plt.show()
```

Overall Distribution of Values of Part Three(TIPI)



```
In [37]: 1 def replace_zeros_with_max_frequency_inplace(df, column_range):
2     for column in df.columns[column_range]:
3         highest_frequency = df[column].value_counts().idxmax()
4         df[column].replace(0, highest_frequency, inplace=True)
```

```
In [38]: 1 replace_zeros_with_max_frequency_inplace(df, range(44,54))
```

```
In [39]: 1 selected_column_range = range(44, 54)
2 result = get_value_counts_per_column_range(df, selected_column_range)
3 for column, counts in result.items():
4     print(f"Value counts for {column}:\n{counts}\n")
```

```
Value counts for TIPI1:
5    8905
1    6229
6    6046
4    5877
2    5603
3    4633
7    2482
Name: TIPI1, dtype: int64
```

```
Value counts for TIPI2:
5    10943
6    6990
4    6171
3    4326
2    4040
1    3985
7    3320
Name: TIPI2, dtype: int64
```

```
Value counts for TIPI3:
6    10820
5    9121
7    6242
4    4516
3    3981
2    2843
1    2252
Name: TIPI3, dtype: int64
```

```
Value counts for TIPI4:
7    11823
6    9561
5    8751
4    2720
2    2444
3    2434
1    2042
Name: TIPI4, dtype: int64
```

```
Value counts for TIPI5:
6    10214
5    9525
7    7682
4    5140
3    3341
2    2226
1    1647
Name: TIPI5, dtype: int64
```

```
Value counts for TIPI6:
7    10116
6    8094
5    7733
4    4943
3    3475
2    2748
1    2666
Name: TIPI6, dtype: int64
```

```
Value counts for TIPI7:
6    11598
7    10007
5    9097
4    4421
3    2270
^   ...
```

```
4    156/
1    1015
```

```
Name: TIPI7, dtype: int64
```

```
Value counts for TIPI8:
```

```
5    9959
6    6369
7    5841
4    4577
3    4446
2    4445
1    4138
```

```
Name: TIPI8, dtype: int64
```

```
Value counts for TIPI9:
```

```
4    7264
3    6819
2    6401
5    6250
1    5507
6    4943
7    2591
```

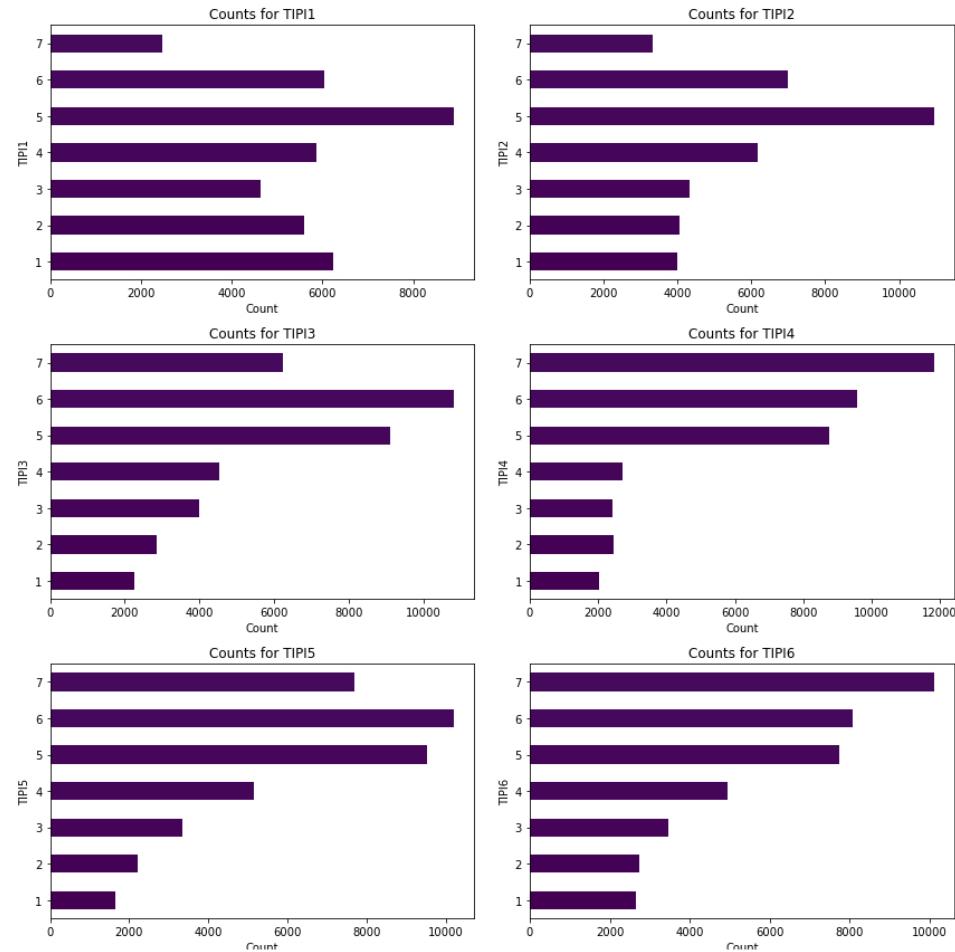
```
Name: TIPI9, dtype: int64
```

```
Value counts for TIPI10:
```

```
4    8220
5    6653
3    6184
2    5714
1    5404
6    4106
7    3494
```

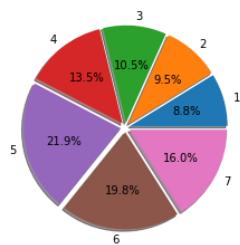
```
Name: TIPI10, dtype: int64
```

```
In [40]: 1 columns_to_plot = df.columns[44:54]
2 num_columns = 2
3 num_rows = math.ceil(len(columns_to_plot) / num_columns)
4 fig, axes = plt.subplots(nrows=num_rows, ncols=num_columns, figsize=(12, 4 * num_rows))
5 for i, column in enumerate(columns_to_plot):
6     row = i // num_columns
7     col = i % num_columns
8     ax = axes[row, col]
9     counts = df[column].value_counts().sort_index()
10    colors = plt.cm.viridis.colors[:len(counts)]
11    counts.plot(kind='barh', ax=ax, color=colors)
12    ax.set_title(f'Counts for {column}')
13    ax.set_ylabel(column)
14    ax.set_xlabel('Count')
15 for i in range(len(columns_to_plot), num_rows * num_columns):
16     fig.delaxes(axes.flatten()[i])
17 plt.tight_layout()
18 plt.show()
```



```
In [42]: 1 columns_to_plot = df.columns[44:54]
2 unique_values = [1, 2, 3, 4, 5, 6, 7]
3 overall_value_counts = df[columns_to_plot].stack().value_counts()
4 total_count = overall_value_counts.sum()
5 overall_percentages = [overall_value_counts.get(val, 0) / total_count * 100 for val in unique_values]
6 explode = [0.05] * len(unique_values)
7 plt.pie(overall_percentages, labels=unique_values, explode=explode, autopct='%.1f%%', shadow=True, startangle=0, labellimit=1, title='Overall Distribution of Values of Part Three(TIPI)')
8 plt.show()
```

Overall Distribution of Values of Part Three(TIPI)



```
Name: VCL3, dtype: int64
```

```
Value counts for VCL4:
```

```
1 34479  
0 5296
```

```
Name: VCL4, dtype: int64
```

```
Value counts for VCL5:
```

```
1 27239  
0 12536
```

```
Name: VCL5, dtype: int64
```

```
Value counts for VCL6:
```

```
0 38177  
1 1598
```

```
Name: VCL6, dtype: int64
```

```
Value counts for VCL7:
```

```
0 36450  
1 3325
```

```
Name: VCL7, dtype: int64
```

```
Value counts for VCL8:
```

```
0 33041  
1 6734
```

```
Name: VCL8, dtype: int64
```

```
Value counts for VCL9:
```

```
0 36423  
1 3352
```

```
Name: VCL12, dtype: int64
```

```
Value counts for VCL13:
```

```
0 28182  
1 11593
```

```
Name: VCL13, dtype: int64
```

```
Value counts for VCL14:
```

```
1 22501  
0 17274
```

```
Name: VCL14, dtype: int64
```

```
Value counts for VCL15:
```

```
1 33688  
0 6087
```

```
Name: VCL15, dtype: int64
```

```
Value counts for VCL16:
```

```
1 37026  
0 2749
```

```
Name: VCL16, dtype: int64
```

```
In [49]: 1 columns_to_check = df.columns[54:70]
```

```
0 5296
```

```
Name: VCL4, dtype: int64
```

```
Value counts for VCL5:
```

```
1 27239  
0 12536
```

```
Name: VCL5, dtype: int64
```

```
Value counts for VCL6:
```

```
0 38177  
1 1598
```

```
Name: VCL6, dtype: int64
```

```
Value counts for VCL7:
```

```
0 36450  
1 3325
```

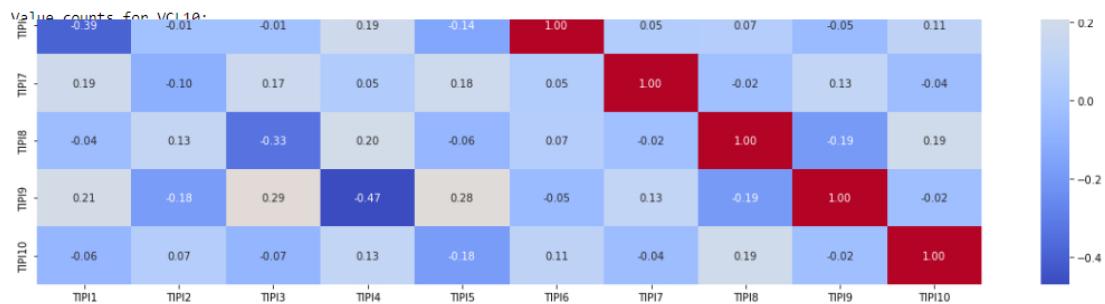
```
Name: VCL7, dtype: int64
```

```
Value counts for VCL8:
```

```
0 33041  
1 6734
```

```
Name: VCL8, dtype: int64
```

```
Value counts for VCL9:  
0    38059  
1    1716  
Name: VCL9, dtype: int64
```



## Part Four (VCL) :

```
In [45]: 1 df.columns.get_loc('VCL1')  
Out[45]: 54
```

```
In [46]: 1 df.columns.get_loc('VCL16')  
Out[46]: 69
```

```
In [47]: 1 for i in df.columns[54:70]:  
2     unique_values = df[i].unique()  
3     print(f"Unique values in {i}: {unique_values}")
```

```
Unique values in VCL1: [1 0]  
Unique values in VCL2: [0 1]  
Unique values in VCL3: [0 1]  
Unique values in VCL4: [1 0]  
Unique values in VCL5: [1 0]  
Unique values in VCL6: [0 1]  
Unique values in VCL7: [1 0]  
Unique values in VCL8: [0 1]  
Unique values in VCL9: [0 1]  
Unique values in VCL10: [1 0]  
Unique values in VCL11: [0 1]  
Unique values in VCL12: [0 1]  
Unique values in VCL13: [0 1]  
Unique values in VCL14: [1 0]  
Unique values in VCL15: [1 0]  
Unique values in VCL16: [1 0]
```

```
In [48]: 1 selected_column_range = range(54, 70)  
2 result = get_value_counts_per_column_range(df, selected_column_range)  
3 for column, counts in result.items():  
4     print(f"Value counts for {column}:\n{counts}\n")
```

```
Value counts for VCL1:  
1    32312  
0    7463  
Name: VCL1, dtype: int64
```

```
Value counts for VCL2:  
1    23064  
0    16711  
Name: VCL2, dtype: int64
```

```
Value counts for VCL3:  
0    33715  
1    6060  
Name: VCL3, dtype: int64
```

```
Value counts for VCL4:  
1    34479  
0    5296  
Name: VCL4, dtype: int64
```

```
Value counts for VCL5:  
1    27239  
0    12536  
Name: VCL5, dtype: int64
```

```
Value counts for VCL6:  
0    38177  
1    1598  
Name: VCL6, dtype: int64
```

```
Value counts for VCL7:  
0    36450  
1    3325  
Name: VCL7, dtype: int64
```

```
Value counts for VCL8:  
0    33041  
1    6734
```

```
Name: VCL8, dtype: int64
```

```
Value counts for VCL9:
```

```
0    38059
```

```
1    1716
```

```
Name: VCL9, dtype: int64
```

```
Value counts for VCL10:
```

```
1    34628
```

```
0    5147
```

```
Name: VCL10, dtype: int64
```

```
Value counts for VCL11:
```

```
0    36844
```

```
1    2931
```

```
Name: VCL11, dtype: int64
```

```
Value counts for VCL12:
```

```
0    36423
```

```
1    3352
```

```
Name: VCL12, dtype: int64
```

```
Value counts for VCL13:
```

```
0    28182
```

```
1    11593
```

```
Name: VCL13, dtype: int64
```

```
Value counts for VCL14:
```

```
1    22501
```

```
0    17274
```

```
Name: VCL14, dtype: int64
```

```
Value counts for VCL15:
```

```
1    33688
```

```
0    6087
```

```
Name: VCL15, dtype: int64
```

```
Value counts for VCL16:
```

```
1    37026
```

```
0    2749
```

```
Name: VCL16, dtype: int64
```

```
In [49]: 1 columns_to_check = df.columns[54:70]
2 plot_null_values(df, columns_to_check)
```

```
Null values in the specified columns:
   VCL1  VCL2  VCL3  VCL4  VCL5  VCL6  VCL7  VCL8  VCL9  VCL10 \
0  False  False  False  False  False  False  False  False  False  False
1  False  False  False  False  False  False  False  False  False  False
2  False  False  False  False  False  False  False  False  False  False
3  False  False  False  False  False  False  False  False  False  False
4  False  False  False  False  False  False  False  False  False  False
...  ...  ...  ...  ...  ...  ...  ...  ...  ...
39770  False  False  False  False  False  False  False  False  False  False
39771  False  False  False  False  False  False  False  False  False  False
39772  False  False  False  False  False  False  False  False  False  False
39773  False  False  False  False  False  False  False  False  False  False
39774  False  False  False  False  False  False  False  False  False  False

   VCL11  VCL12  VCL13  VCL14  VCL15  VCL16
0  False  False  False  False  False  False
1  False  False  False  False  False  False
2  False  False  False  False  False  False
3  False  False  False  False  False  False
4  False  False  False  False  False  False
...  ...  ...  ...  ...  ...
39770  False  False  False  False  False  False
39771  False  False  False  False  False  False
39772  False  False  False  False  False  False
39773  False  False  False  False  False  False
39774  False  False  False  False  False  False
```

```
[39775 rows x 16 columns]
```

Null Values Distribution in Dataframe

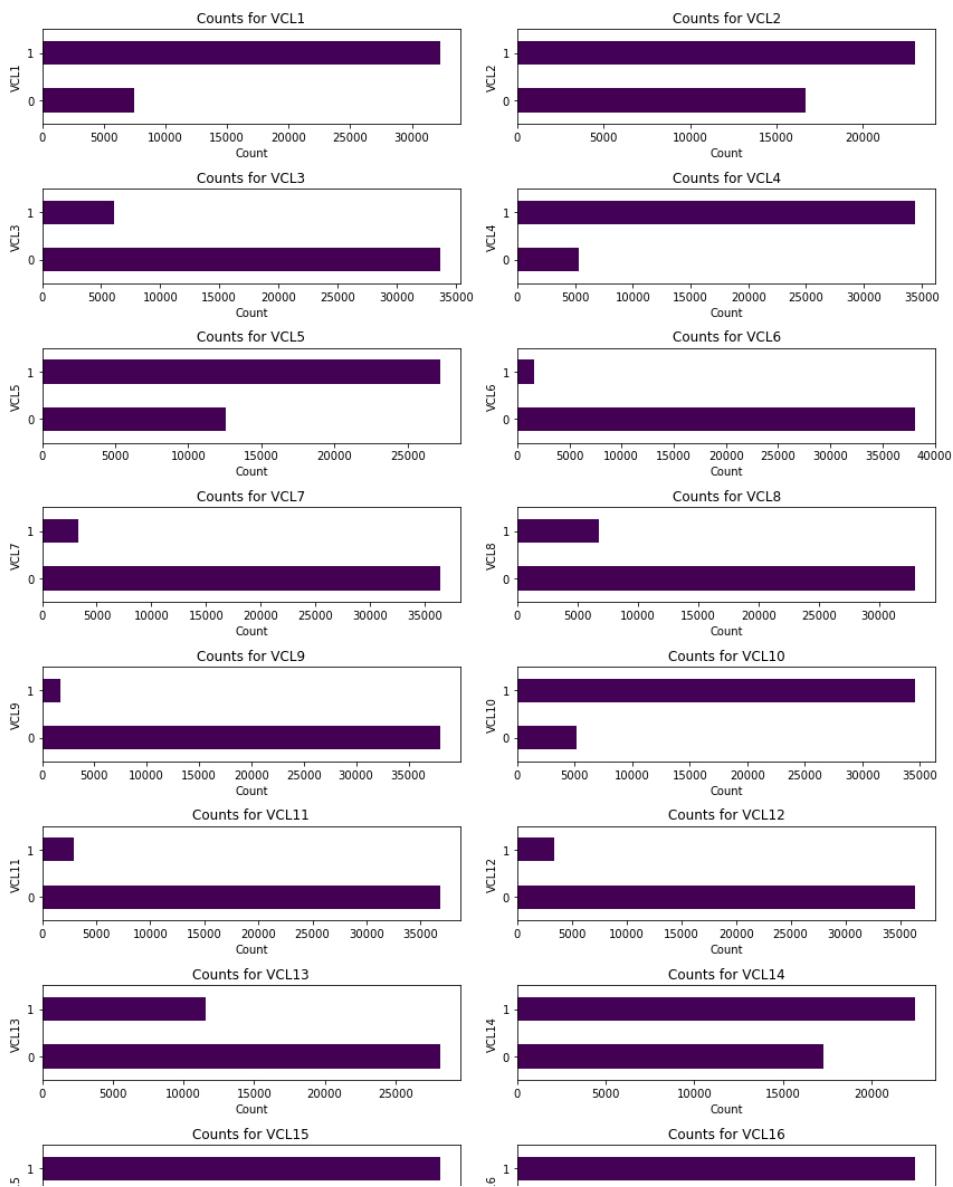


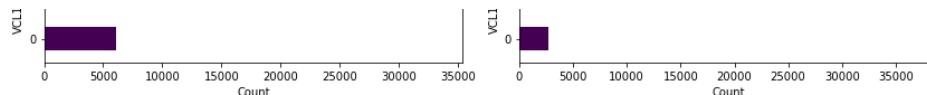


```
In [50]: 1 columns_to_sum = df.columns[54:70]
2 sum_null_values(df,columns_to_sum)
```

```
Out[50]: {'VCL1': 0,
 'VCL2': 0,
 'VCL3': 0,
 'VCL4': 0,
 'VCL5': 0,
 'VCL6': 0,
 'VCL7': 0,
 'VCL8': 0,
 'VCL9': 0,
 'VCL10': 0,
 'VCL11': 0,
 'VCL12': 0,
 'VCL13': 0,
 'VCL14': 0,
 'VCL15': 0,
 'VCL16': 0}
```

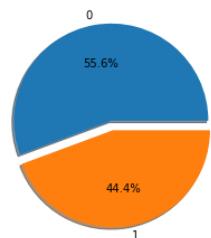
```
In [51]: 1 columns_to_plot = df.columns[54:70]
2 num_columns = 2
3 num_rows = math.ceil(len(columns_to_plot) / num_columns)
4 fig, axes = plt.subplots(nrows=num_rows, ncols=num_columns, figsize=(12, 2 * num_rows))
5 for i, column in enumerate(columns_to_plot):
6     row = i // num_columns
7     col = i % num_columns
8     ax = axes[row, col]
9     counts = df[column].value_counts().sort_index()
10    colors = plt.cm.viridis.colors[:len(counts)]
11    counts.plot(kind='barh', ax=ax, color=colors)
12    ax.set_title(f'Counts for {column}')
13    ax.set_ylabel(column)
14    ax.set_xlabel('Count')
15    for i in range(len(columns_to_plot), num_rows * num_columns):
16        fig.delaxes(axes.flatten()[i])
17 plt.tight_layout()
18 plt.show()
```





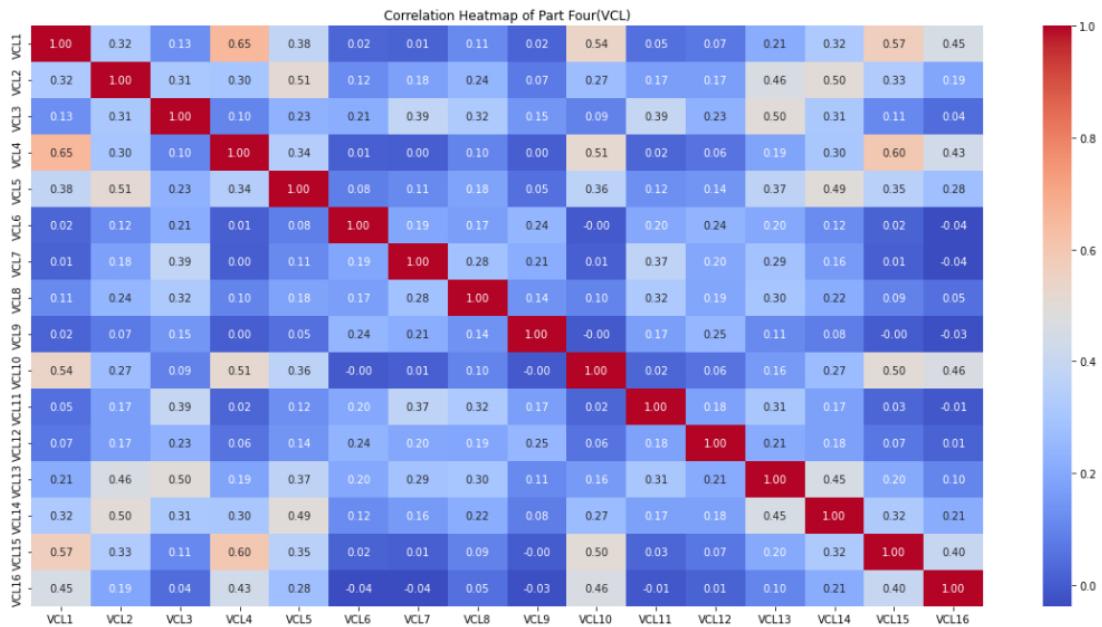
```
In [52]: 1 columns_to_plot = df.columns[54:70]
2 unique_values = [0, 1]
3 overall_value_counts = df[columns_to_plot].stack().value_counts()
4 total_count = overall_value_counts.sum()
5 overall_percentages = [overall_value_counts.get(val, 0) / total_count * 100 for val in unique_values]
6 explode = [.05] * len(unique_values)
7 plt.pie(overall_percentages, labels=unique_values, explode=explode, autopct='%.1f%%', shadow=True, startangle=0, labeldistance=1.1)
8 plt.title('Overall Distribution of Values of Part Four(VCL)')
9 plt.show()
```

Overall Distribution of Values of Part Four(VCL)



```
In [53]: 1 selected_columns = df.iloc[:, 54:70]
```

```
In [54]: 1 plt.figure(figsize=(20, 10))
2 sns.heatmap(selected_columns.corr(), annot=True, cmap='coolwarm', fmt=".2f")
3 plt.title('Correlation Heatmap of Part Four(VCL)')
4 plt.show()
```



## Part Five (General Info) :

### in Education Column

```
In [55]: 1 df.columns.get_loc('education')
```

```
Out[55]: 70
```

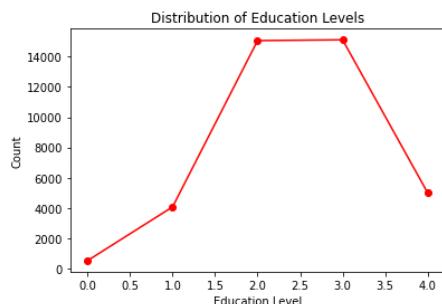
```
In [56]: 1 df['education'].unique()
```

```
Out[56]: array([2, 1, 3, 4, 0], dtype=int64)
```

```
In [57]: 1 df['education'].value_counts()
```

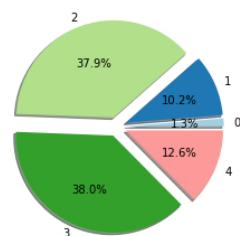
```
Out[57]: 3    15120
2    15066
4     5008
1     4066
0      515
Name: education, dtype: int64
```

```
In [58]: 1 value_counts = df['education'].value_counts().sort_index()  
In [59]: 1 plt.plot(value_counts.index, value_counts.values, color='r', marker='o')  
2 plt.xlabel('Education Level')  
3 plt.ylabel('Count')  
4 plt.title('Distribution of Education Levels')  
5 plt.show()
```



```
In [60]: 1 explode_values = [0.1] * len(value_counts)  
2 plt.pie(value_counts, labels=value_counts.index, explode=explode_values, autopct='%1.1f%%', shadow=True, startangle=0, label  
3 4 plt.show()
```

Distribution of Education Levels



```
In [61]: 1 replace_zeros_with_max_frequency_inplace(df, range(70, 71))
```

```
In [62]: 1 df['education'].unique()
```

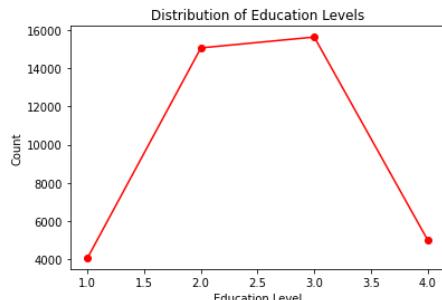
```
Out[62]: array([2, 1, 3, 4], dtype=int64)
```

```
In [63]: 1 df['education'].value_counts()
```

```
Out[63]: 3    15635  
2    15066  
4    5008  
1    4066  
Name: education, dtype: int64
```

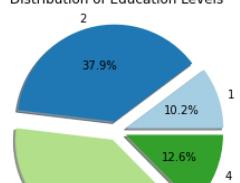
```
In [64]: 1 value_counts = df['education'].value_counts().sort_index()
```

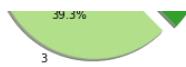
```
In [65]: 1 plt.plot(value_counts.index, value_counts.values, color='r', marker='o')  
2 plt.xlabel('Education Level')  
3 plt.ylabel('Count')  
4 plt.title('Distribution of Education Levels')  
5 plt.show()
```



```
In [66]: 1 explode_values = [0.1] * len(value_counts)  
2 plt.pie(value_counts, labels=value_counts.index, explode=explode_values, autopct='%1.1f%%', shadow=True, startangle=0, label  
3 4 plt.show()
```

Distribution of Education Levels

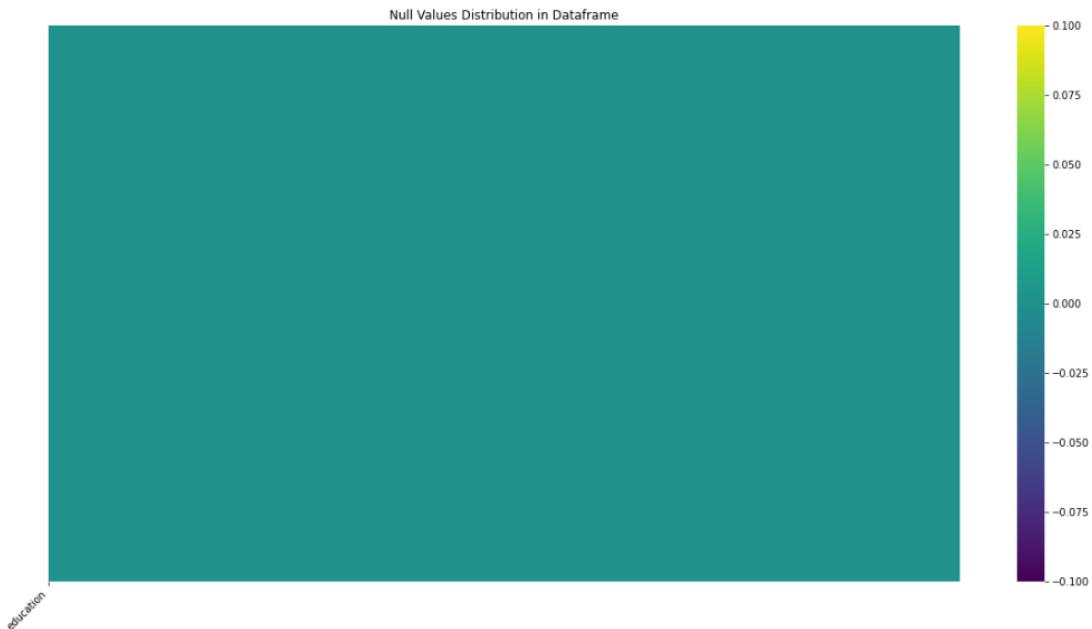




```
In [67]: 1 columns_to_check = df.columns[70:71]
2 plot_null_values(df, columns_to_check)

Null values in the specified columns:
   education
0      False
1      False
2      False
3      False
4      False
...
39770     False
39771     False
39772     False
39773     False
39774     False

[39775 rows x 1 columns]
```

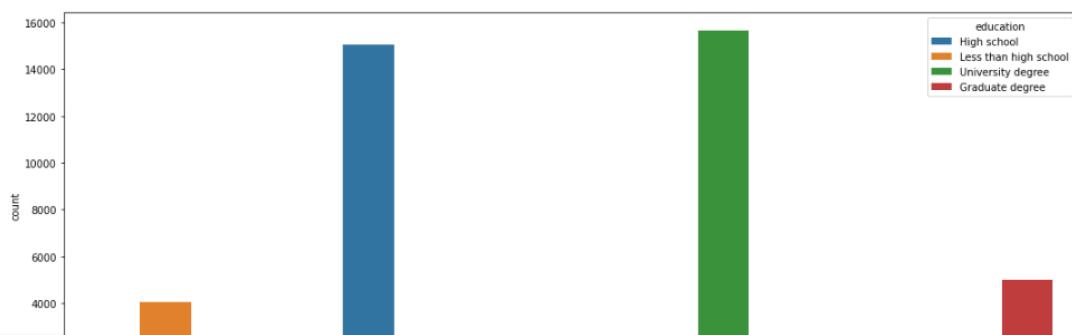


```
In [68]: 1 def makeEducationGroup(value):
2     if value == 1:
3         return 'Less than high school'
4     if value == 2:
5         return 'High school'
6     if value == 3:
7         return 'University degree'
8     if value == 4:
9         return 'Graduate degree'
```

```
In [69]: 1 education_Groups = df['education'].apply(makeEducationGroup)
```

```
In [70]: 1 plt.figure(figsize=(18, 7))
2 sns.countplot(x=df['education'], hue=education_Groups)
```

```
Out[70]: <AxesSubplot:xlabel='education', ylabel='count'>
```



## in Urban Column

```
In [71]: 1 df.columns.get_loc('urban')
```

```
Out[71]: 71
```

```
In [72]: 1 df['urban'].unique()
```

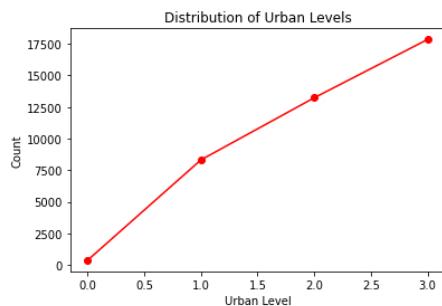
```
Out[72]: array([3, 2, 1, 0], dtype=int64)
```

```
In [73]: 1 df['urban'].value_counts()
```

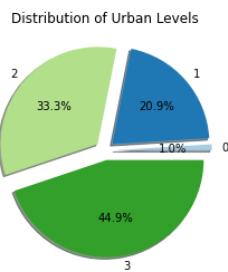
```
Out[73]: 3    17843
2    13232
1     8318
0      382
Name: urban, dtype: int64
```

```
In [74]: 1 value_counts = df['urban'].value_counts().sort_index()
```

```
In [75]: 1 plt.plot(value_counts.index, value_counts.values, color='r', marker='o')
2 plt.xlabel('Urban Level')
3 plt.ylabel('Count')
4 plt.title('Distribution of Urban Levels')
5 plt.show()
```



```
In [76]: 1 explode_values = [0.1] * len(value_counts)
2 plt.pie(value_counts, labels=value_counts.index, explode=explode_values, autopct='%1.1f%%', shadow=True, startangle=0, label
3 plt.title('Distribution of Urban Levels')
4 plt.show()
```



```
In [77]: 1 replace_zeros_with_max_frequency_inplace(df, range(71, 72))
```

```
In [78]: 1 df['urban'].unique()
```

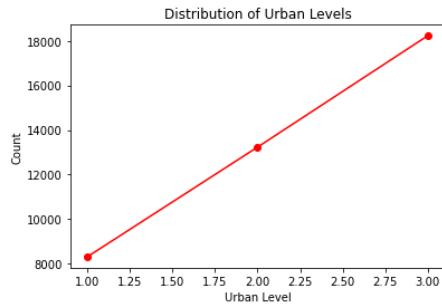
```
Out[78]: array([3, 2, 1], dtype=int64)
```

```
In [79]: 1 df['urban'].value_counts()
```

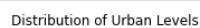
```
Out[79]: 3    18225
2    13232
1     8318
Name: urban, dtype: int64
```

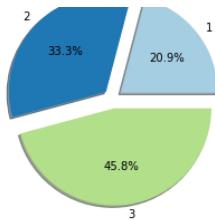
```
In [80]: 1 value_counts = df['urban'].value_counts().sort_index()
```

```
In [81]: 1 plt.plot(value_counts.index, value_counts.values, color='r', marker='o')
2 plt.xlabel('Urban Level')
3 plt.ylabel('Count')
4 plt.title('Distribution of Urban Levels')
5 plt.show()
```



```
In [82]: 1 explode_values = [0.1] * len(value_counts)
2 plt.pie(value_counts, labels=value_counts.index, explode=explode_values, autopct='%1.1f%%', shadow=True, startangle=0, label
3 plt.title('Distribution of Urban Levels')
4 plt.show()
```





```
In [83]: 1 columns_to_check = df.columns[71:72]
2 plot_null_values(df, columns_to_check)
```

Null values in the specified columns:

```
urban
0    False
1    False
2    False
3    False
4    False
...
39770  False
39771  False
39772  False
39773  False
39774  False
```

[39775 rows x 1 columns]

Null Values Distribution in Dataframe

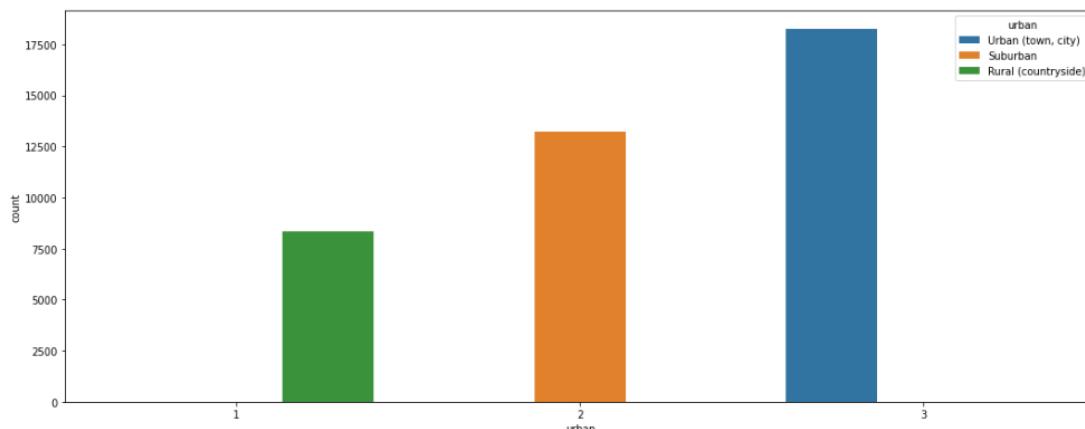


```
In [84]: 1 def makeUrbanGroup(value):
2     if value == 1:
3         return 'Rural (countryside)'
4     if value == 2:
5         return 'Suburban'
6     if value == 3:
7         return 'Urban (town, city)'
```

```
In [85]: 1 Urban = df['urban'].apply(makeUrbanGroup)
```

```
In [86]: 1 plt.figure(figsize=(18, 7))
2 sns.countplot(x=df['urban'], hue=Urban)
```

```
Out[86]: <AxesSubplot:xlabel='urban', ylabel='count'>
```



## in Gender Column

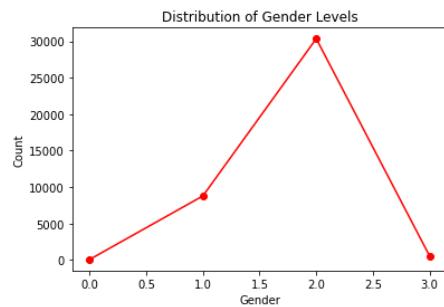
```
In [87]: 1 df.columns.get_loc('gender')
Out[87]: 72

In [88]: 1 df['gender'].unique()
Out[88]: array([2, 1, 3, 0], dtype=int64)

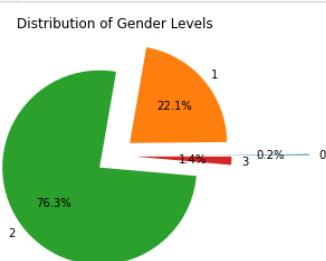
In [89]: 1 df['gender'].value_counts()
Out[89]: 2    30367
1     8789
3      552
0       67
Name: gender, dtype: int64

In [90]: 1 value_counts = df['gender'].value_counts().sort_index()

In [91]: 1 plt.plot(value_counts.index, value_counts.values, color='r', marker='o')
2 plt.xlabel('Gender')
3 plt.ylabel('Count')
4 plt.title('Distribution of Gender Levels')
5 plt.show()
```



```
In [92]: 1 explode_values = [0.1] * len(value_counts)
2 plt.pie(value_counts, labels=value_counts.index, explode=explode, autopct='%1.1f%%', shadow=False, startangle=0, lab
3 plt.title('Distribution of Gender Levels')
4 plt.show()
```



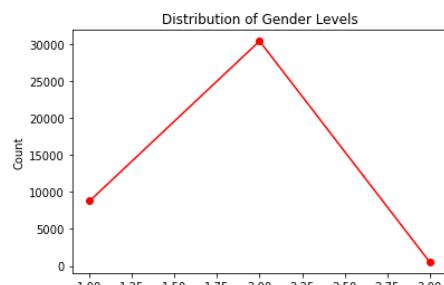
```
In [93]: 1 replace_zeros_with_max_frequency_inplace(df, range(72, 73))
```

```
In [94]: 1 df['gender'].unique()
Out[94]: array([2, 1, 3], dtype=int64)
```

```
In [95]: 1 df['gender'].value_counts()
Out[95]: 2    30434
1     8789
3      552
Name: gender, dtype: int64
```

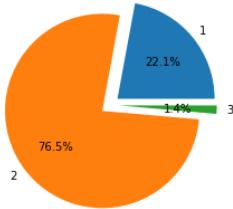
```
In [96]: 1 value_counts = df['gender'].value_counts().sort_index()

In [97]: 1 plt.plot(value_counts.index, value_counts.values, color='r', marker='o')
2 plt.xlabel('Gender')
3 plt.ylabel('Count')
4 plt.title('Distribution of Gender Levels')
5 plt.show()
```



```
Gender  
In [98]: 1 explode_values = [0.1] * len(value_counts)  
2 plt.pie(value_counts, labels=value_counts.index, explode=explode_values, autopct='%.1f%%', shadow=False, startangle=0, labe  
3 plt.title('Distribution of Gender Levels')  
4 plt.show()
```

Distribution of Gender Levels



```
In [99]: 1 columns_to_check = df.columns[72:73]  
2 plot_null_values(df, columns_to_check)
```

Null values in the specified columns:

```
gender  
0      False  
1      False  
2      False  
3      False  
4      False  
...     ...  
39770   False  
39771   False  
39772   False  
39773   False  
39774   False
```

[39775 rows x 1 columns]

Null Values Distribution in Dataframe

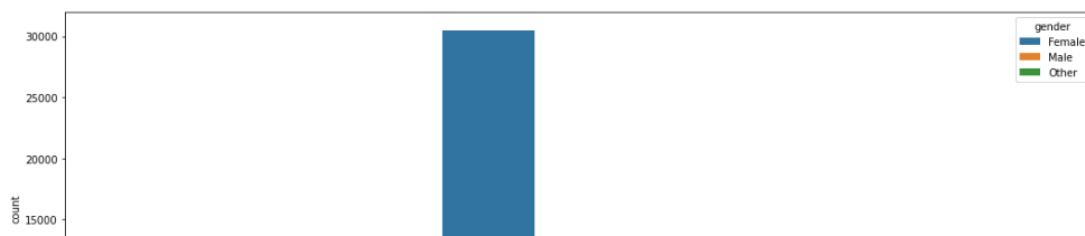


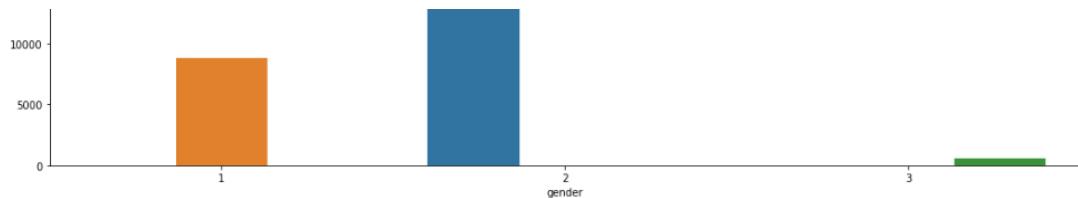
```
In [100]: 1 def makeGenderGroup(value):  
2     if value == 1:  
3         return 'Male'  
4     if value == 2:  
5         return 'Female'  
6     if value == 3:  
7         return 'Other'
```

```
In [101]: 1 gender = df['gender'].apply(makeGenderGroup)
```

```
In [102]: 1 plt.figure(figsize=(18, 7))  
2 sns.countplot(x=df['gender'], hue=gender)
```

```
Out[102]: <AxesSubplot:xlabel='gender', ylabel='count'>
```





### in Engnat column

```
In [103]: 1 df.columns.get_loc('engnat')
```

```
Out[103]: 73
```

```
In [104]: 1 df['engnat'].unique()
```

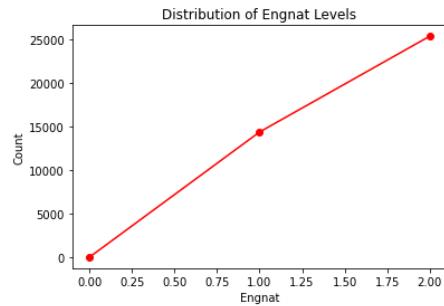
```
Out[104]: array([2, 1, 0], dtype=int64)
```

```
In [105]: 1 df['gender'].value_counts()
```

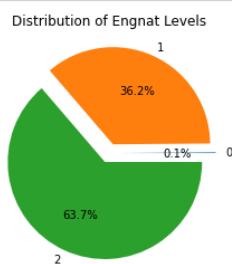
```
Out[105]: 2    30434
1     8789
3      552
Name: gender, dtype: int64
```

```
In [106]: 1 value_counts = df['engnat'].value_counts().sort_index()
```

```
In [107]: 1 plt.plot(value_counts.index, value_counts.values, color='r', marker='o')
2 plt.xlabel('Engnat')
3 plt.ylabel('Count')
4 plt.title('Distribution of Engnat Levels')
5 plt.show()
```



```
In [108]: 1 explode_values = [0.1] * len(value_counts)
2 plt.pie(value_counts, labels=value_counts.index, explode=explode_values, autopct='%1.1f%%', shadow=False, startangle=0, labe
3 plt.title('Distribution of Engnat Levels')
4 plt.show()
```



```
In [109]: 1 replace_zeros_with_max_frequency_inplace(df, range(73, 74))
```

```
In [110]: 1 df['engnat'].unique()
```

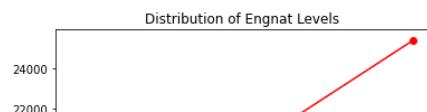
```
Out[110]: array([2, 1, 0], dtype=int64)
```

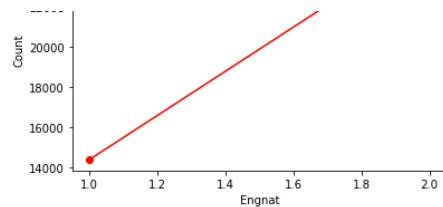
```
In [111]: 1 df['engnat'].value_counts()
```

```
Out[111]: 2    25395
1    14380
Name: engnat, dtype: int64
```

```
In [112]: 1 value_counts = df['engnat'].value_counts().sort_index()
```

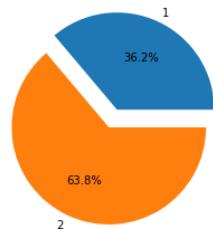
```
In [113]: 1 plt.plot(value_counts.index, value_counts.values, color='r', marker='o')
2 plt.xlabel('Engnat')
3 plt.ylabel('Count')
4 plt.title('Distribution of Engnat Levels')
5 plt.show()
```





```
In [114]: 1 explode_values = [0.1] * len(value_counts)
2 plt.pie(value_counts, labels=value_counts.index, explode=explode_values, autopct='%1.1f%%', shadow=False, startangle=0, labe
3 plt.title('Distribution of Engnat Levels')
4 plt.show()
```

Distribution of Engnat Levels



```
In [115]: 1 columns_to_check = df.columns[73:74]
2 plot_null_values(df, columns_to_check)
```

Null values in the specified columns:

```
engnat
0      False
1      False
2      False
3      False
4      False
...
39770  False
39771  False
39772  False
39773  False
39774  False
```

[39775 rows x 1 columns]

Null Values Distribution in Dataframe



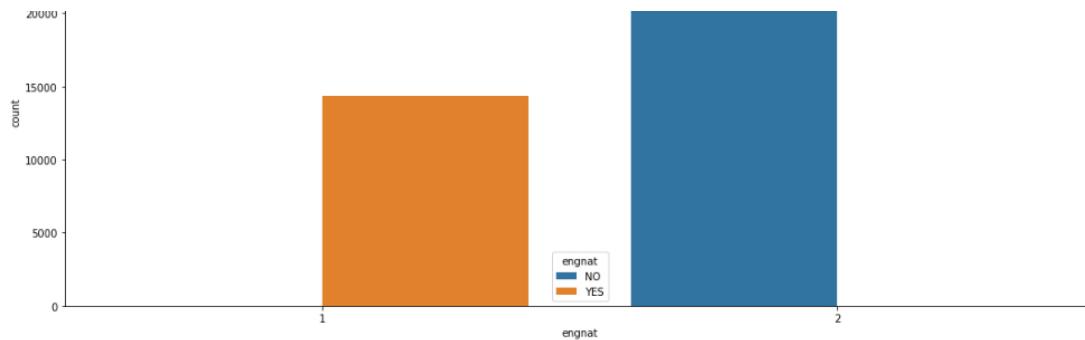
```
In [116]: 1 def makeEngnatGroup(value):
2     if value == 1:
3         return 'YES'
4     if value == 2:
5         return 'NO'
```

```
In [117]: 1 engnat = df['engnat'].apply(makeEngnatGroup)
```

```
In [118]: 1 plt.figure(figsize=(18, 7))
2 sns.countplot(x=df['engnat'], hue=engnat)
```

```
Out[118]: <AxesSubplot:xlabel='engnat', ylabel='count'>
```





## in Age column

```
In [119]: 1 df.columns.get_loc('age')
```

```
Out[119]: 74
```

```
In [120]: 1 df['age'].unique()
```

```
Out[120]: array([ 16,  17,  13,  19,  20,  29,  18,  15,  31,  34,  22,
   42,  21,  23,  26,  43,  35,  25,  24,  61,  40,  27,
   28,  60,  14,  37,  46,  57,  41,  55,  36,  30,  32,
   56,  71,  67,  44,  38,  49,  45,  47,  33,  54,  53,
   50,  48,  58,  68,  39,  51,  62,  59,  52,  77,  85,
   63,  69,  65,  75,  72,  78,  73,  64,  66,  79,  80,
   78,  223, 1996,  89,  117,  82,  74,  1998,  76,  115, 1993,
  1991,  99], dtype=int64)
```

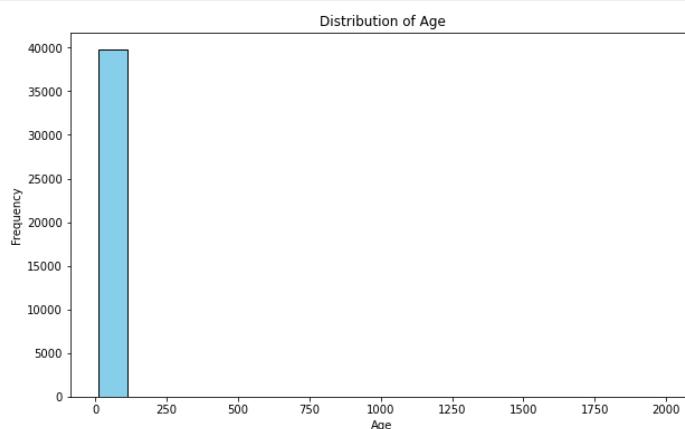
```
In [121]: 1 df['age'].describe()
```

```
Out[121]: count    39775.00
mean      23.61
std       21.58
min      13.00
25%     18.00
50%     21.00
75%     25.00
max    1998.00
Name: age, dtype: float64
```

```
In [122]: 1 df['age'].value_counts()
```

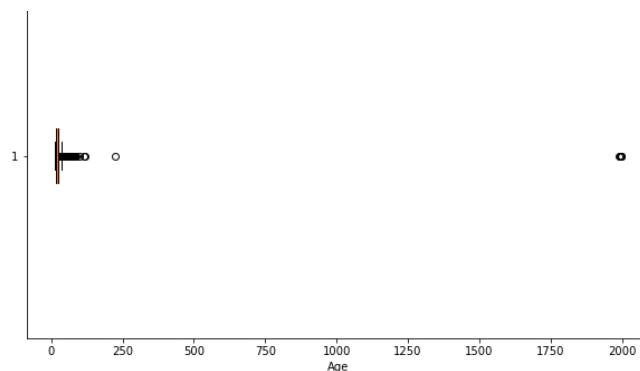
```
Out[122]: 20      3789
21      3535
19      3510
18      3046
22      3009
...
89      1
1996     1
223      1
78      1
99      1
Name: age, Length: 79, dtype: int64
```

```
In [123]: 1 plt.figure(figsize=(10, 6))
2 plt.hist(df['age'], bins=20, color='skyblue', edgecolor='black')
3 plt.title('Distribution of Age')
4 plt.xlabel('Age')
5 plt.ylabel('Frequency')
6 plt.show()
```

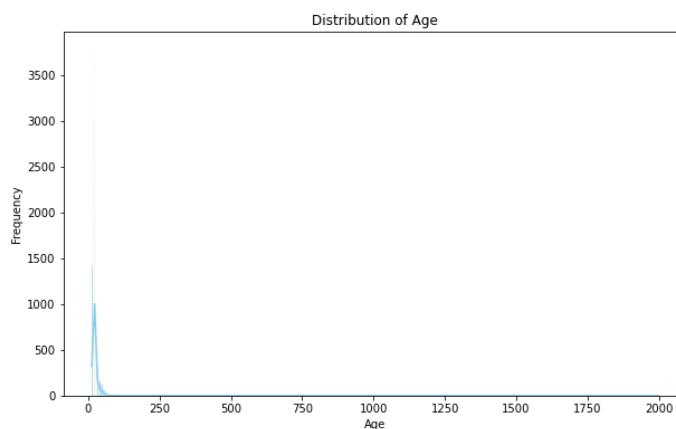


```
In [124]: 1 plt.figure(figsize=(10, 6))
2 plt.boxplot(df['age'], vert=False, patch_artist=True)
3 plt.title('Boxplot of Age')
4 plt.xlabel('Age')
5 plt.show()
```

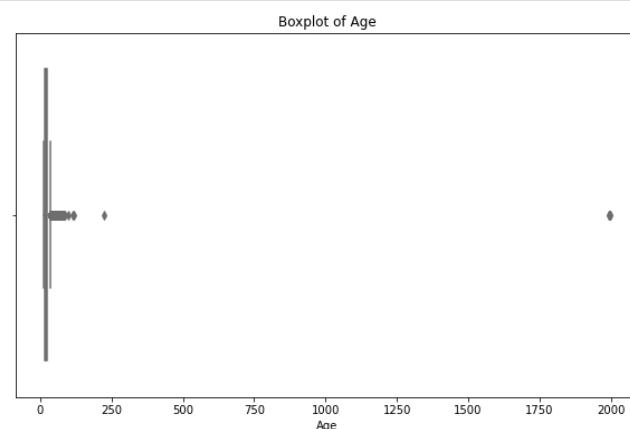
Boxplot of Age



```
In [125]: 1 plt.figure(figsize=(10, 6))
2 sns.histplot(df['age'], kde=True, color='skyblue')
3 plt.title('Distribution of Age')
4 plt.xlabel('Age')
5 plt.ylabel('Frequency')
6 plt.show()
```



```
In [126]: 1 plt.figure(figsize=(10, 6))
2 sns.boxplot(x=df['age'], color='lightcoral')
3 plt.title('Boxplot of Age')
4 plt.xlabel('Age')
5 plt.show()
```



```
In [127]: 1 age = df[(df['age'] >= 13) & (df['age'] <= 100)]
```

```
In [128]: 1 scaler = MinMaxScaler()
2 df['scaled_age'] = scaler.fit_transform(df[['age']])
```

```
In [129]: 1 df['age']
```

```
Out[129]: 0      16
1      16
2      17
3      13
4      19
..
39770   16
39771   21
39772   48
39773   20
39774   26
Name: age, Length: 39775, dtype: int64
```

```
In [130]: 1 columns_to_check = df.columns[74:75]
2 plot_null_values(df, columns_to_check)
```

```
Null values in the specified columns:  
    age  
0    False  
1    False  
2    False  
3    False  
4    False  
...  
39770  False  
39771  False  
39772  False  
39773  False  
39774  False
```

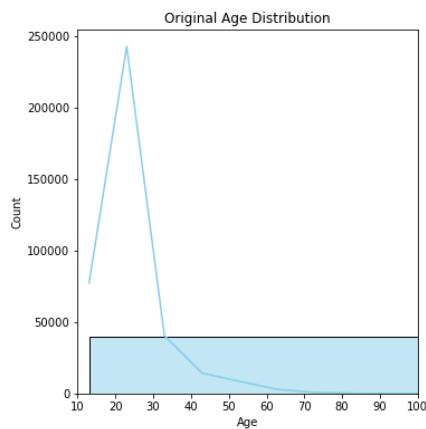
[39775 rows x 1 columns]

Null Values Distribution in Dataframe



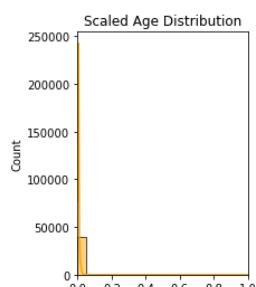
```
In [131]: 1 plt.figure(figsize=(12, 6))  
2 plt.subplot(1, 2, 1)  
3 sns.histplot(df['age'], bins=20, kde=True, color='skyblue')  
4 plt.title('Original Age Distribution')  
5 plt.xlabel('Age')  
6 plt.xlim(10, 100)
```

Out[131]: (10.0, 100.0)



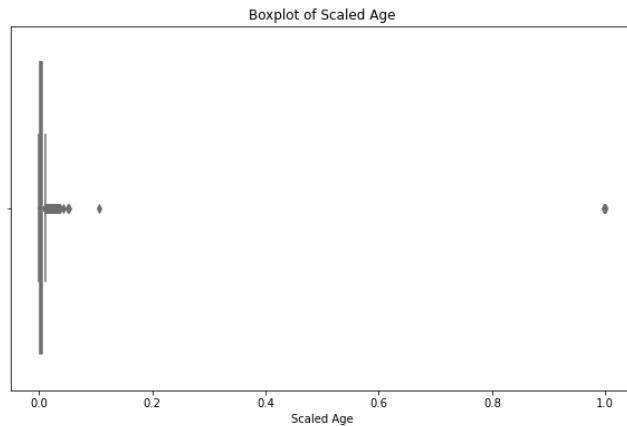
```
In [132]: 1 plt.subplot(1, 2, 2)  
2 sns.histplot(df['scaled_age'], bins=20, kde=True, color='orange')  
3 plt.title('Scaled Age Distribution')  
4 plt.xlabel('Scaled Age')  
5 plt.xlim(0, 1)
```

Out[132]: (0.0, 1.0)



## Scaled Age

```
In [133]: 1 plt.figure(figsize=(10, 6))
2 sns.boxplot(x=df['scaled_age'], color='lightcoral')
3 plt.title('Boxplot of Scaled Age')
4 plt.xlabel('Scaled Age')
5 plt.show()
```

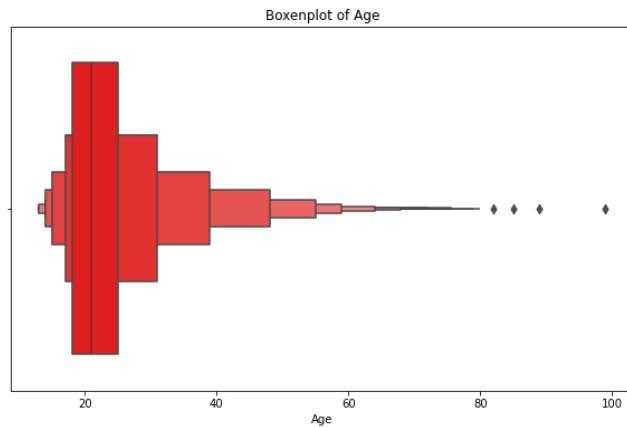


```
In [134]: 1 age_indexes = df[df['age'] > 100]['age'].index
2 display(age_indexes)
3 print(f'DataFrame size before: {df.shape[0]}')
4 df.drop(age_indexes, axis=0, inplace=True)
5 print(f'DataFrame size after: {df.shape[0]}')

Int64Index([5340, 10231, 14236, 21383, 24898, 30027, 33730], dtype='int64')

DataFrame size before: 39775
DataFrame size after: 39768
```

```
In [135]: 1 plt.figure(figsize=(10, 6))
2 sns.boxplot(x=df['age'], color='red')
3 plt.title('Boxenplot of Age')
4 plt.xlabel('Age')
5 plt.show()
```



```
In [136]: 1 df.columns

Out[136]: Index(['Q1A', 'Q2A', 'Q3A', 'Q4A', 'Q5A', 'Q6A', 'Q7A', 'Q8A', 'Q9A', 'Q10A',
 'Q11A', 'Q12A', 'Q13A', 'Q14A', 'Q15A', 'Q16A', 'Q17A', 'Q18A', 'Q19A',
 'Q20A', 'Q21A', 'Q22A', 'Q23A', 'Q24A', 'Q25A', 'Q26A', 'Q27A', 'Q28A',
 'Q29A', 'Q30A', 'Q31A', 'Q32A', 'Q33A', 'Q34A', 'Q35A', 'Q36A', 'Q37A',
 'Q38A', 'Q39A', 'Q40A', 'Q41A', 'Q42A', 'country', 'source', 'TIP1I',
 'TIP1I2', 'TIP1I3', 'TIP1I4', 'TIP1I5', 'TIP1I6', 'TIP1I7', 'TIP1I8', 'TIP1I9',
 'TIP1I10', 'VCL1', 'VCL1', 'VCL2', 'VCL3', 'VCL4', 'VCL5', 'VCL6', 'VCL7',
 'VCL8', 'VCL9', 'VCL10', 'VCL11', 'VCL12', 'VCL13', 'VCL14', 'VCL15',
 'VCL16', 'education', 'urban', 'gender', 'engnat', 'age', 'screensize',
 'uniquenetworklocation', 'hand', 'religion', 'orientation', 'race',
 'voted', 'married', 'familysize', 'major', 'scaled_age'],
 dtype='object')
```

```
In [137]: 1 df = df.drop('scaled_age', axis=1)
```

```
In [138]: 1 df.columns
```

```
Out[138]: Index(['Q1A', 'Q2A', 'Q3A', 'Q4A', 'Q5A', 'Q6A', 'Q7A', 'Q8A', 'Q9A', 'Q10A',
 'Q11A', 'Q12A', 'Q13A', 'Q14A', 'Q15A', 'Q16A', 'Q17A', 'Q18A', 'Q19A',
 'Q20A', 'Q21A', 'Q22A', 'Q23A', 'Q24A', 'Q25A', 'Q26A', 'Q27A', 'Q28A',
 'Q29A', 'Q30A', 'Q31A', 'Q32A', 'Q33A', 'Q34A', 'Q35A', 'Q36A', 'Q37A',
 'Q38A', 'Q39A', 'Q40A', 'Q41A', 'Q42A', 'country', 'source', 'TIP1I',
 'TIP1I2', 'TIP1I3', 'TIP1I4', 'TIP1I5', 'TIP1I6', 'TIP1I7', 'TIP1I8', 'TIP1I9',
 'TIP1I10', 'VCL1', 'VCL2', 'VCL3', 'VCL4', 'VCL5', 'VCL6', 'VCL7',
 'VCL8', 'VCL9', 'VCL10', 'VCL11', 'VCL12', 'VCL13', 'VCL14', 'VCL15',
 'VCL16', 'education', 'urban', 'gender', 'engnat', 'age', 'screensize',
 'uniquenetworklocation', 'hand', 'religion', 'orientation', 'race',
 'voted', 'married', 'familysize', 'major'],
 dtype='object')
```

```
In [139]: 1 df.shape
Out[139]: (39768, 85)

In [140]: 1 df['age'].unique()
Out[140]: array([16, 17, 13, 19, 20, 29, 18, 15, 31, 34, 22, 42, 21, 23, 26, 43, 35,
   25, 24, 61, 40, 27, 28, 60, 14, 37, 46, 57, 41, 55, 36, 30, 32, 56,
   71, 67, 44, 38, 49, 45, 47, 33, 54, 53, 50, 48, 58, 68, 39, 51, 62,
   59, 52, 77, 85, 63, 69, 65, 75, 72, 70, 73, 64, 66, 79, 80, 78, 89,
   82, 74, 76, 99], dtype=int64)

In [141]: 1 def makeAgeGroup(value):
2     if value <= 10:
3         return 'Under 10'
4     if 10 <= value <= 16:
5         return 'Primary Children'
6     if 17 <= value <= 21:
7         return 'Secondary Children'
8     if 21 <= value <= 35:
9         return 'Adults'
10    if 36 <= value <= 48:
11        return 'Elder Adults'
12    if value >= 49:
13        return 'Older People'

In [142]: 1 age_group = df['age'].apply(makeAgeGroup)

In [143]: 1 plt.figure(figsize=(18, 7))
2 sns.countplot(x=age_group)
3 plt.title('Count of People in Each Age Group')
4 plt.xlabel('Age Group')
5 plt.show()

Count of People in Each Age Group
```

| Age Group          | Count  |
|--------------------|--------|
| Primary Children   | ~5000  |
| Secondary Children | ~16000 |
| Adults             | ~15000 |
| Elder Adults       | ~2000  |
| Older People       | ~1000  |

```
In [144]: 1 def makeAgeGroupFeature(value):
2     if value <= 10:
3         return 1
4     if 10 <= value <= 16:
5         return 2
6     if 17 <= value <= 21:
7         return 3
8     if 21 <= value <= 35:
9         return 4
10    if 36 <= value <= 48:
11        return 5
12    if value >= 49:
13        return 6

In [145]: 1 df['age_group'] = df['age'].apply(makeAgeGroupFeature)
2 df

Out[145]:
Q1A Q2A Q3A Q4A Q5A Q6A Q7A Q8A Q9A Q10A Q11A Q12A Q13A Q14A Q15A Q16A Q17A Q18A Q19A Q20A Q21A Q22A Q23A
0 4 4 2 4 4 4 4 4 2 1 4 4 4 4 4 4 3 4 3 1 4 4 4
1 4 1 2 3 4 4 3 4 3 2 2 2 4 4 4 3 3 4 2 1 1 2 3 1
2 3 1 4 1 4 3 1 3 2 4 2 1 4 1 4 4 4 4 2 2 1 4 3 2
3 2 3 2 1 3 3 4 2 3 3 2 1 1 4 2 2 3 1 1 2 1 1 1
4 2 2 3 4 4 2 4 4 4 3 2 4 4 4 4 3 4 4 4 4 3 3 4
...
39770 2 1 3 2 3 2 1 3 1 4 2 2 4 2 1 3 4 2 4 3 4 3 1
39771 3 4 3 4 3 4 4 4 3 4 4 4 4 4 4 4 3 3 4 4 4 4 3
39772 2 1 2 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1
39773 3 1 2 2 3 3 3 4 3 1 3 3 4 4 1 1 2 3 1 2 2 4 1
39774 2 1 2 1 4 2 1 1 1 1 3 1 3 1 1 4 4 4 3 4 2 1

39768 rows × 86 columns
```

## In Hand Column

```
In [146]: 1 df.columns.get_loc('hand')
```

```
Out[146]: 77
```

```
In [147]: 1 df['hand'].unique()
```

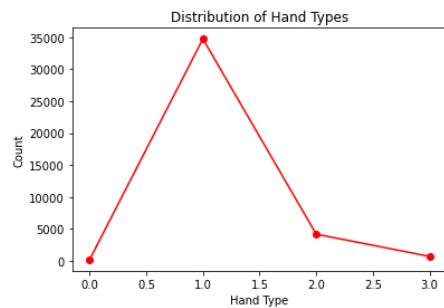
```
Out[147]: array([1, 2, 3, 0], dtype=int64)
```

```
In [148]: 1 df['hand'].value_counts()
```

```
Out[148]: 1    34735
2     4171
3      689
0      173
Name: hand, dtype: int64
```

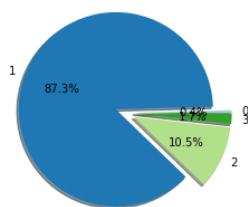
```
In [149]: 1 value_counts = df['hand'].value_counts().sort_index()
```

```
In [150]: 1 plt.plot(value_counts.index, value_counts.values, color='r', marker='o')
2 plt.xlabel('Hand Type')
3 plt.ylabel('Count')
4 plt.title('Distribution of Hand Types')
5 plt.show()
```



```
In [151]: 1 explode_values = [0.1] * len(value_counts)
2 plt.pie(value_counts, labels=value_counts.index, explode=explode_values, autopct='%1.1f%%', shadow=True, startangle=0, label
3 4 plt.show()
```

Distribution of Hand Types



```
In [152]: 1 replace_zeros_with_max_frequency_inplace(df, range(77, 78))
```

```
In [153]: 1 df['hand'].unique()
```

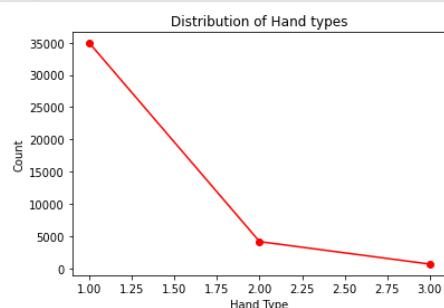
```
Out[153]: array([1, 2, 3], dtype=int64)
```

```
In [154]: 1 df['hand'].value_counts()
```

```
Out[154]: 1    34908
2     4171
3      689
Name: hand, dtype: int64
```

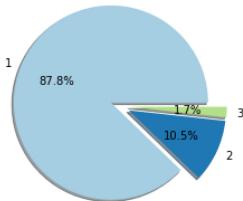
```
In [155]: 1 value_counts = df['hand'].value_counts().sort_index()
```

```
In [156]: 1 plt.plot(value_counts.index, value_counts.values, color='r', marker='o')
2 plt.xlabel('Hand Type')
3 plt.ylabel('Count')
4 plt.title('Distribution of Hand types')
5 plt.show()
```



```
In [157]: 1 explode_values = [0.1] * len(value_counts)
2 plt.pie(value_counts, labels=value_counts.index, explode=explode_values, autopct='%1.1f%%', shadow=True, startangle=0, label
3 plt.title('Distribution of Urban Levels')
4 plt.show()
```

Distribution of Urban Levels



```
In [158]: 1 columns_to_check = df.columns[77:78]
2 plot_null_values(df, columns_to_check)
```

Null values in the specified columns:

```
hand
0    False
1    False
2    False
3    False
4    False
...
39770  False
39771  False
39772  False
39773  False
39774  False
```

[39768 rows x 1 columns]

Null Values Distribution in Dataframe

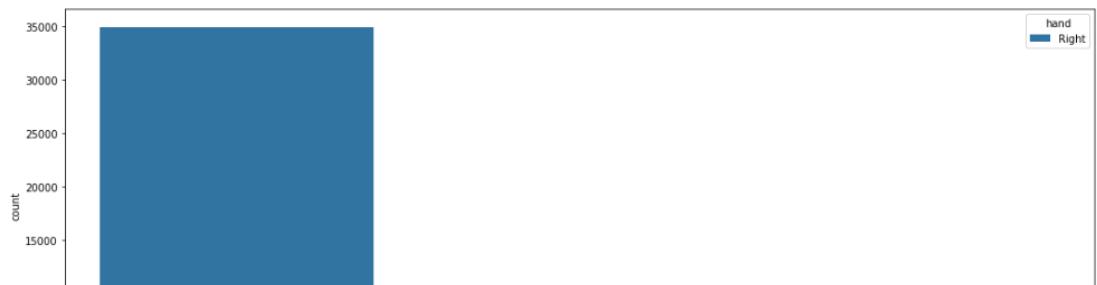


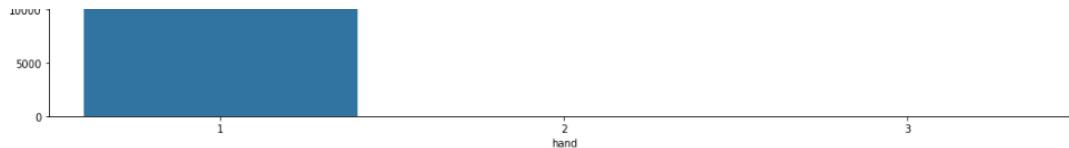
```
In [159]: 1 def makeHandGroup(value):
2     if value == 1:
3         return 'Right'
4     if 10 == 2:
5         return 'Left'
6     if 17 == 3:
7         return 'Both'
```

```
In [160]: 1 hand_group = df['hand'].apply(makeHandGroup)
```

```
In [161]: 1 plt.figure(figsize=(18, 7))
2 sns.countplot(x=df['hand'], hue=hand_group)
```

Out[161]: <AxesSubplot:xlabel='hand', ylabel='count'>





## in Religion Column

```
In [162]: 1 df.columns.get_loc('religion')
```

```
Out[162]: 78
```

```
In [163]: 1 df['religion'].unique()
```

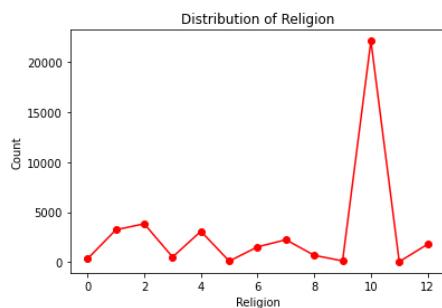
```
Out[163]: array([12, 7, 4, 10, 2, 6, 1, 8, 3, 0, 5, 11, 9], dtype=int64)
```

```
In [164]: 1 df['religion'].value_counts()
```

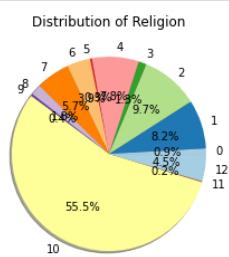
```
Out[164]: 10    22069
2     3844
1     3245
4     3096
7     2253
12    1884
6     1544
8     700
3     527
0     356
9     144
5     122
11    64
Name: religion, dtype: int64
```

```
In [165]: 1 value_counts = df['religion'].value_counts().sort_index()
```

```
In [166]: 1 plt.plot(value_counts.index, value_counts.values, color='r', marker='o')
2 plt.xlabel('Religion')
3 plt.ylabel('Count')
4 plt.title('Distribution of Religion')
5 plt.show()
```



```
In [167]: 1 explode_values = [0.1] * len(value_counts)
2 plt.pie(value_counts, labels=value_counts.index, autopct='%.1f%%', shadow=True, startangle=0, labeldistance=1.1, colors=plt
3 plt.title('Distribution of Religion')
4 plt.show()
```



```
In [168]: 1 replace_zeros_with_max_frequency(inplace=df, range(78, 79))
```

```
In [169]: 1 df['religion'].unique()
```

```
Out[169]: array([12, 7, 4, 10, 2, 6, 1, 8, 3, 5, 11, 9], dtype=int64)
```

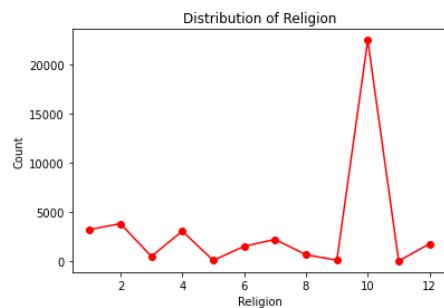
```
In [170]: 1 df['religion'].value_counts()
```

```
Out[170]: 10    22425
2     3844
1     3245
4     3096
7     2253
12    1884
6     1544
8     700
3     527
9     144
5     122
Name: religion, dtype: int64
```

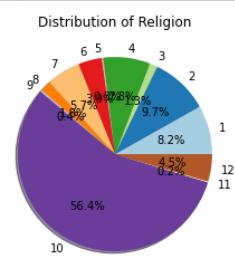
```
11      64  
Name: religion, dtype: int64
```

```
In [171]: 1 value_counts = df['religion'].value_counts().sort_index()
```

```
In [172]: 1 plt.plot(value_counts.index, value_counts.values, color='r', marker='o')  
2 plt.xlabel('Religion')  
3 plt.ylabel('Count')  
4 plt.title('Distribution of Religion')  
5 plt.show()
```



```
In [173]: 1 explode_values = [0.1] * len(value_counts)  
2 plt.pie(value_counts, labels=value_counts.index, autopct='%1.1f%%', shadow=True, startangle=0, labeldistance=1.1, colors=plt  
3 plt.title('Distribution of Religion')  
4 plt.show()
```



```
In [174]: 1 columns_to_check = df.columns[78:79]  
2 plot_null_values(df, columns_to_check)
```

```
Null values in the specified columns:  
    religion  
0      False  
1      False  
2      False  
3      False  
4      False  
...  
39770     False  
39771     False  
39772     False  
39773     False  
39774     False
```

```
[39768 rows x 1 columns]
```

Null Values Distribution in Dataframe



```
In [175]: 1 def makeReligiongroup(value):
```

```

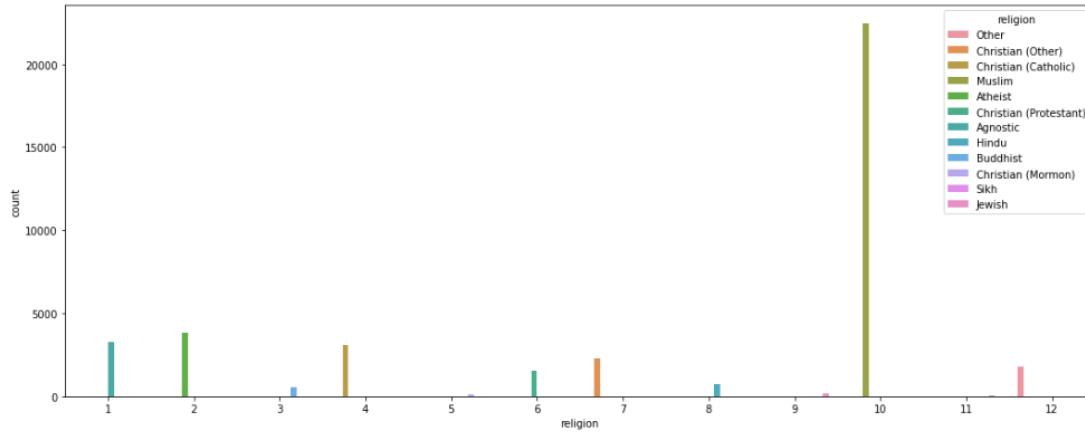
2     if value == 1:
3         return 'Agnostic'
4     if value == 2:
5         return 'Atheist'
6     if value == 3:
7         return 'Buddhist'
8     if value == 4:
9         return 'Christian (Catholic)'
10    if value == 5:
11        return 'Christian (Mormon)'
12    if value == 6:
13        return 'Christian (Protestant)'
14    if value == 7:
15        return 'Christian (Other)'
16    if value == 8:
17        return 'Hindu'
18    if value == 9:
19        return 'Jewish'
20    if value == 10:
21        return 'Muslim'
22    if value == 11:
23        return 'Sikh'
24    if value == 12:
25        return 'Other'

```

In [176]: 1 religion = df['religion'].apply(makeReligiongroup)

In [177]: 1 plt.figure(figsize=(18, 7))  
2 sns.countplot(x=df['religion'], hue=religion)

Out[177]: <AxesSubplot:xlabel='religion', ylabel='count'>



## in Orientation Column

In [178]: 1 df.columns.get\_loc('orientation')

Out[178]: 79

In [179]: 1 df['orientation'].unique()

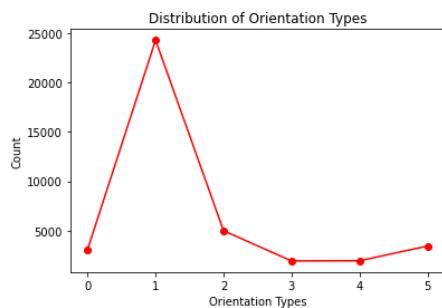
Out[179]: array([1, 0, 3, 5, 2, 4], dtype=int64)

In [180]: 1 df['orientation'].value\_counts()

Out[180]: 1 24266  
2 5006  
5 3459  
0 3109  
4 1975  
3 1953  
Name: orientation, dtype: int64

In [181]: 1 value\_counts = df['orientation'].value\_counts().sort\_index()

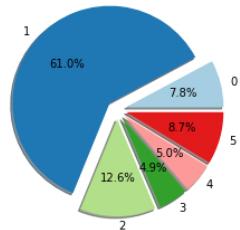
In [182]: 1 plt.plot(value\_counts.index, value\_counts.values, color='r', marker='o')  
2 plt.xlabel('Orientation Types')  
3 plt.ylabel('Count')  
4 plt.title('Distribution of Orientation Types')  
5 plt.show()



In [183]: 1 explode values = [0.1] \* len(value\_counts)

```
1 plt.pie(value_counts, labels=value_counts.index, explode=explode_values, autopct='%.1f%%', shadow=True, startangle=0, label
2 plt.title('Distribution of Orientation Types')
3 plt.show()
```

Distribution of Orientation Types



```
In [184]: 1 replace_zeros_with_max_frequency_inplace(df, range(79, 80))
```

```
In [185]: 1 df['orientation'].unique()
```

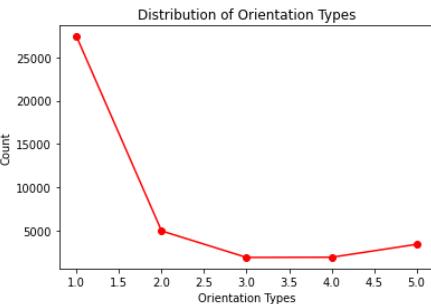
```
Out[185]: array([1, 3, 5, 2, 4], dtype=int64)
```

```
In [186]: 1 df['orientation'].value_counts()
```

```
Out[186]: 1    27375
2     5006
5     3459
4     1975
3     1953
Name: orientation, dtype: int64
```

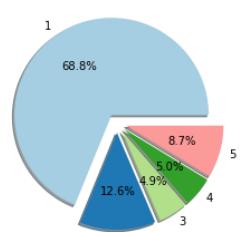
```
In [187]: 1 value_counts = df['orientation'].value_counts().sort_index()
```

```
In [188]: 1 plt.plot(value_counts.index, value_counts.values, color='r', marker='o')
2 plt.xlabel('Orientation Types')
3 plt.ylabel('Count')
4 plt.title('Distribution of Orientation Types')
5 plt.show()
```



```
In [189]: 1 explode_values = [0.1] * len(value_counts)
2 plt.pie(value_counts, labels=value_counts.index, explode=explode_values, autopct='%.1f%%', shadow=True, startangle=0, label
3 plt.title('Distribution of Orientation Types')
4 plt.show()
```

Distribution of Orientation Types



```
In [190]: 1 columns_to_check = df.columns[79:80]
2 plot_null_values(df, columns_to_check)
```

Null values in the specified columns:

```
orientation
0      False
1      False
2      False
3      False
4      False
...
39770    False
39771    False
39772    False
39773    False
39774    False
```

[39768 rows x 1 columns]

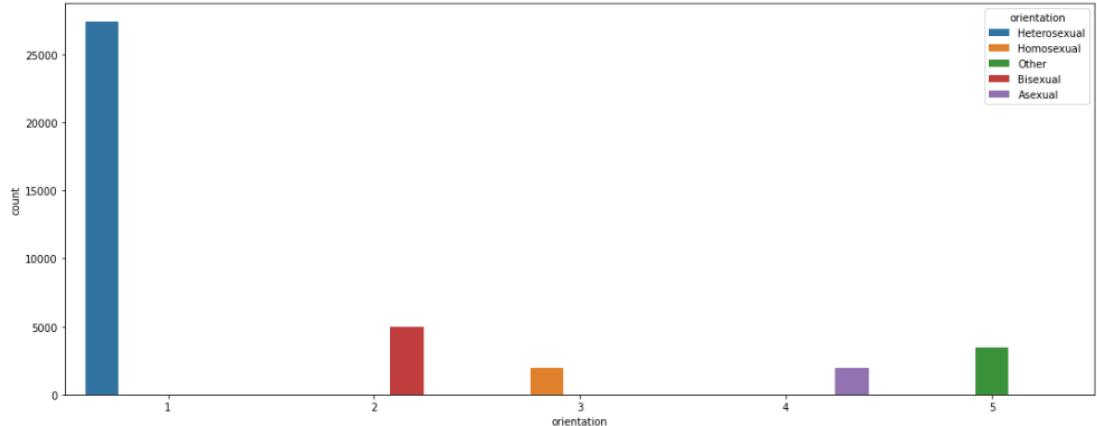


```
In [191]: 1 def makeOrientationGroup(value):
2     if value == 1:
3         return 'Heterosexual'
4     if value == 2:
5         return 'Bisexual'
6     if value == 3:
7         return 'Homosexual'
8     if value == 4:
9         return 'Asexual'
10    if value == 5:
11        return 'Other'
```

```
In [192]: 1 orientation = df['orientation'].apply(makeOrientationGroup)
```

```
In [193]: 1 plt.figure(figsize=(18, 7))
2 sns.countplot(x=df['orientation'], hue=orientation)
```

```
Out[193]: <AxesSubplot:xlabel='orientation', ylabel='count'>
```



## in Race Column

```
In [194]: 1 df.columns.get_loc('race')
```

```
Out[194]: 80
```

```
In [195]: 1 df['race'].unique()
```

```
Out[195]: array([10, 70, 60, 50, 30, 40, 20], dtype=int64)
```

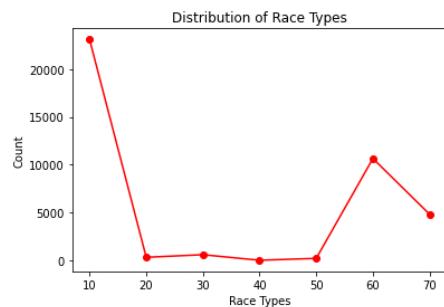
```
In [196]: 1 df['race'].value_counts()
```

```
Out[196]: 10    23101
60    10658
70     4831
30      603
20      333
50      220
40       22
Name: race, dtype: int64
```

```
In [197]: 1 value_counts = df['race'].value_counts().sort_index()
```

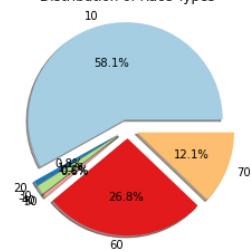
```
In [198]: 1 plt.plot(value_counts.index, value_counts.values, color='r', marker='o')
2 plt.xlabel('Race Types')
3 plt.ylabel('Count')
4 plt.title('Distribution of Race Types')
```

```
4 plt.title('Distribution of Race Types')
5 plt.show()
```



```
In [199]: 1 explode_values = [0.1] * len(value_counts)
2 plt.pie(value_counts, labels=value_counts.index, explode=explode_values, autopct='%.1f%%', shadow=True, startangle=0, label
3 plt.title('Distribution of Race Types')
4 plt.show()
```

Distribution of Race Types



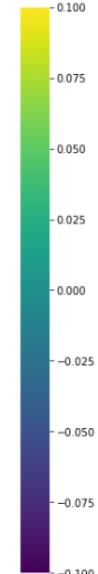
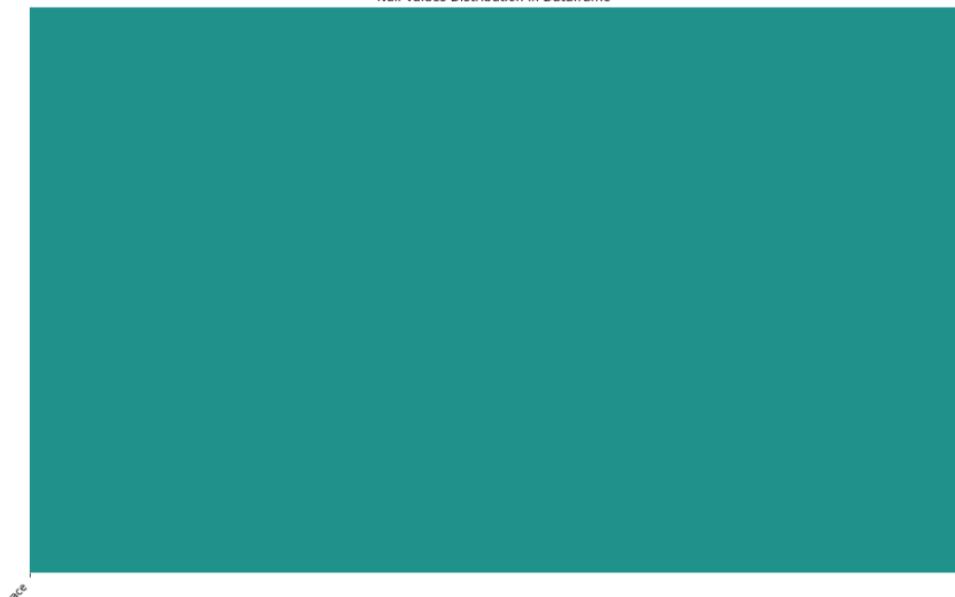
```
In [200]: 1 columns_to_check = df.columns[80:81]
2 plot_null_values(df, columns_to_check)
```

Null values in the specified columns:

```
race
0    False
1    False
2    False
3    False
4    False
...
39770  False
39771  False
39772  False
39773  False
39774  False
```

[39768 rows x 1 columns]

Null Values Distribution in Dataframe



```
In [201]: 1 def makeRaceGroup(value):
2     if value == 10:
3         return 'Asian'
4     if value == 20:
5         return 'Arab'
6     if value == 30:
7         return 'Black'
8     if value == 40:
9         return 'Indigenous Australian'
10    if value == 50:
11        return 'Native American'
```

```

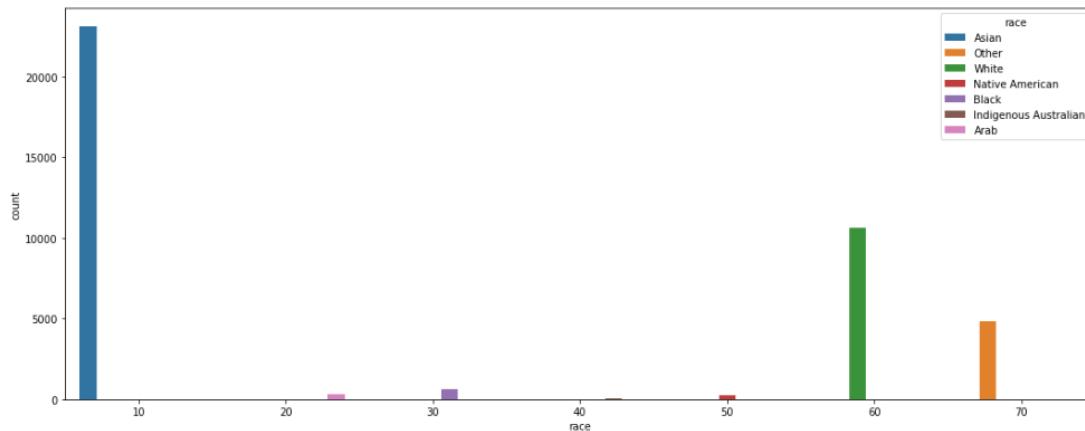
12     if value == 60:
13         return 'White'
14     if value == 70:
15         return 'Other'

In [202]: 1 race = df['race'].apply(makeRaceGroup)

In [203]: 1 plt.figure(figsize=(18, 7))
2 sns.countplot(x=df['race'], hue=race)

Out[203]: <AxesSubplot:xlabel='race', ylabel='count'>

```



## in Voted column

```

In [204]: 1 df.columns.get_loc('voted')

Out[204]: 81

In [205]: 1 df['voted'].unique()

Out[205]: array([2, 1, 0], dtype=int64)

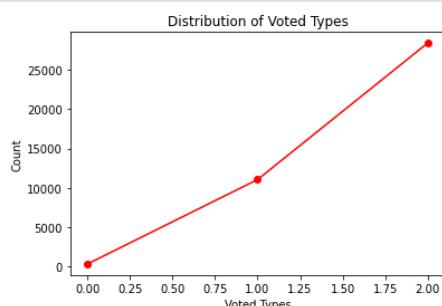
In [206]: 1 df['voted'].value_counts()

Out[206]: 2    28393
1    11048
0     327
Name: voted, dtype: int64

In [207]: 1 value_counts = df['voted'].value_counts().sort_index()

In [208]: 1 plt.plot(value_counts.index, value_counts.values, color='r', marker='o')
2 plt.xlabel('Voted Types')
3 plt.ylabel('Count')
4 plt.title('Distribution of Voted Types')
5 plt.show()

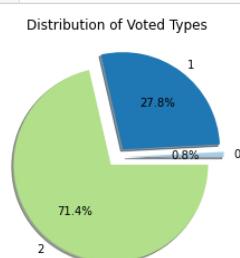
```



```

In [209]: 1 explode_values = [0.1] * len(value_counts)
2 plt.pie(value_counts, labels=value_counts.index, explode=explode_values, autopct='%1.1f%%', shadow=True, startangle=0, label
3 plt.title('Distribution of Voted Types')
4 plt.show()

```



```

In [210]: 1 replace_zeros_with_max_frequency_inplace(df, range(81, 82))

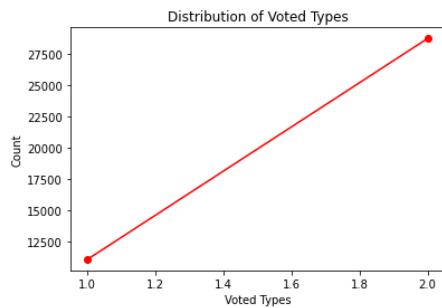
```

```
In [211]: 1 df['voted'].unique()
Out[211]: array([2, 1], dtype=int64)

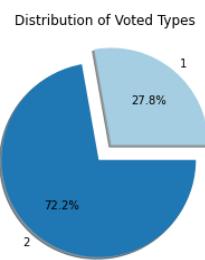
In [212]: 1 df['voted'].value_counts()
Out[212]: 2    28720
1    11048
Name: voted, dtype: int64

In [213]: 1 value_counts = df['voted'].value_counts().sort_index()

In [214]: 1 plt.plot(value_counts.index, value_counts.values, color='r', marker='o')
2 plt.xlabel('Voted Types')
3 plt.ylabel('Count')
4 plt.title('Distribution of Voted Types')
5 plt.show()
```



```
In [215]: 1 explode_values = [0.1] * len(value_counts)
2 plt.pie(value_counts, labels=value_counts.index, explode=explode_values, autopct='%1.1f%%', shadow=True, startangle=0, label
3 plt.title('Distribution of Voted Types')
4 plt.show()
```



```
In [216]: 1 columns_to_check = df.columns[81:82]
2 plot_null_values(df, columns_to_check)
```

Null values in the specified columns:

| voted |
|-------|
| 0     |
| 1     |
| 2     |
| 3     |
| 4     |
| ...   |
| 39770 |
| 39771 |
| 39772 |
| 39773 |
| 39774 |

[39768 rows x 1 columns]



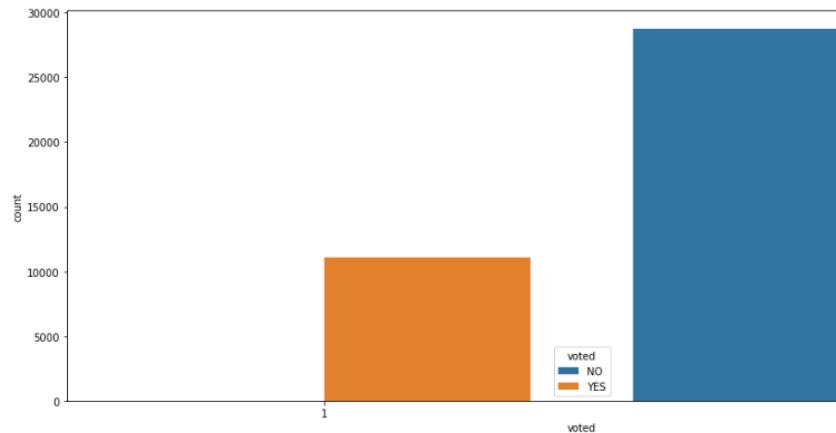
-0.100

```
In [217]: 1 def makeVotedGroup(value):
2     if value == 1:
3         return 'YES'
4     if value == 2:
5         return 'NO'
```

```
In [218]: 1 voted = df['voted'].apply(makeVotedGroup)
```

```
In [219]: 1 plt.figure(figsize=(18, 7))
2 sns.countplot(x=df['voted'], hue=voted)
```

```
Out[219]: <AxesSubplot:xlabel='voted', ylabel='count'>
```



## in Married column

```
In [220]: 1 df.columns.get_loc('married')
```

```
Out[220]: 82
```

```
In [221]: 1 df['married'].unique()
```

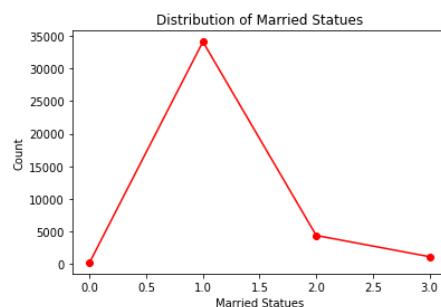
```
Out[221]: array([1, 3, 2, 0], dtype=int64)
```

```
In [222]: 1 df['married'].value_counts()
```

```
Out[222]: 1    34125
2     4356
3     1092
0      195
Name: married, dtype: int64
```

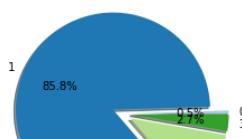
```
In [223]: 1 value_counts = df['married'].value_counts().sort_index()
```

```
In [224]: 1 plt.plot(value_counts.index, value_counts.values, color='r', marker='o')
2 plt.xlabel('Married Statuses')
3 plt.ylabel('Count')
4 plt.title('Distribution of Married Statuses')
5 plt.show()
```



```
In [225]: 1 explode_values = [0.1] * len(value_counts)
2 plt.pie(value_counts, labels=value_counts.index, explode=explode_values, autopct='%1.1f%%', shadow=True, startangle=0, label
3 4 plt.show()
```

Distribution of Married Statuses





```
In [226]: 1 replace_zeros_with_max_frequency_inplace(df, range(82, 83))
```

```
In [227]: 1 df['married'].unique()
```

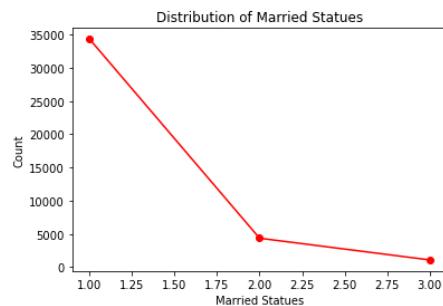
```
Out[227]: array([1, 3, 2], dtype=int64)
```

```
In [228]: 1 df['married'].value_counts()
```

```
Out[228]: 1    34320
2     4356
3     1092
Name: married, dtype: int64
```

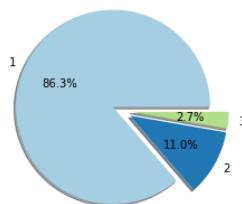
```
In [229]: 1 value_counts = df['married'].value_counts().sort_index()
```

```
In [230]: 1 plt.plot(value_counts.index, value_counts.values, color='r', marker='o')
2 plt.xlabel('Married Statuses')
3 plt.ylabel('Count')
4 plt.title('Distribution of Married Statuses')
5 plt.show()
```



```
In [231]: 1 explode_values = [0.1] * len(value_counts)
2 plt.pie(value_counts, labels=value_counts.index, explode=explode_values, autopct='%1.1f%%', shadow=True, startangle=0, label
3 plt.title('Distribution of Married Statuses')
4 plt.show()
```

Distribution of Married Statuses



```
In [232]: 1 columns_to_check = df.columns[82:83]
2 plot_null_values(df, columns_to_check)
```

Null values in the specified columns:

```
    married
0      False
1      False
2      False
3      False
4      False
...
39770   False
39771   False
39772   False
39773   False
39774   False
```

[39768 rows x 1 columns]

Null Values Distribution in Dataframe



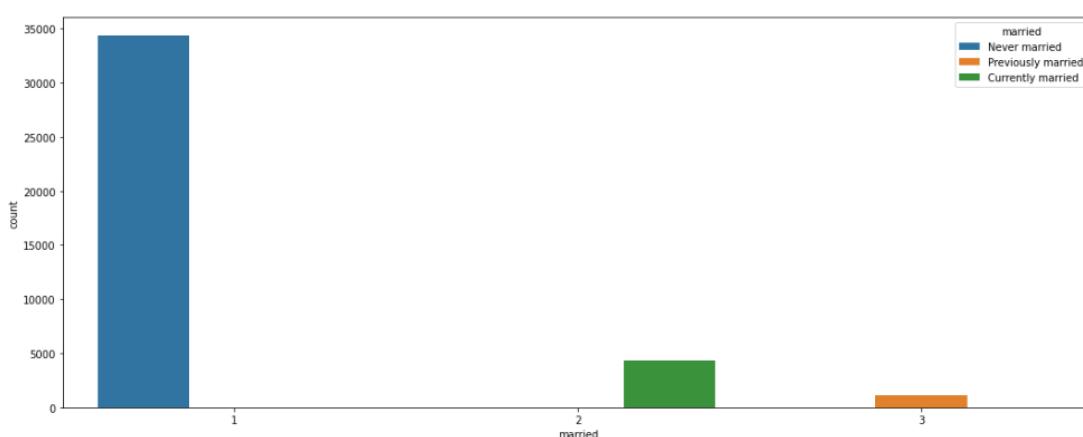


```
In [233]: 1 def makeMarriedGroup(value):
2     if value == 1:
3         return 'Never married'
4     if value == 2:
5         return 'Currently married'
6     if value == 3:
7         return 'Previously married'

In [234]: 1 married = df['married'].apply(makeMarriedGroup)

In [235]: 1 plt.figure(figsize=(18, 7))
2 sns.countplot(x=df['married'], hue=married)

Out[235]: <AxesSubplot:xlabel='married', ylabel='count'>
```



## in Familysize Column

```
In [236]: 1 df.columns.get_loc('familysize')

Out[236]: 83

In [237]: 1 df['familysize'].unique()

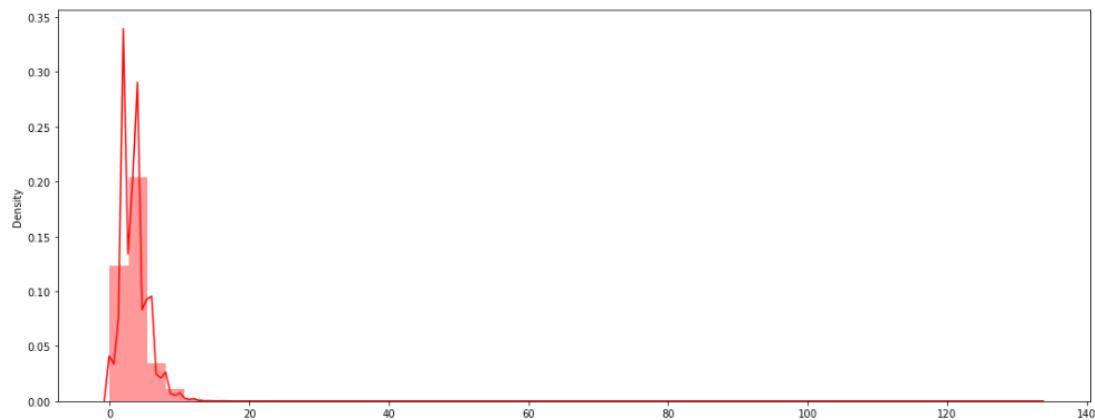
Out[237]: array([ 2,  4,  3,  5,  1,  6,  8, 12,  7,  0, 11,  9, 13,
   18, 62, 16, 14, 99, 17, 54, 65, 15, 26, 24, 19, 21,
   23, 133], dtype=int64)

In [238]: 1 df['familysize'].value_counts()

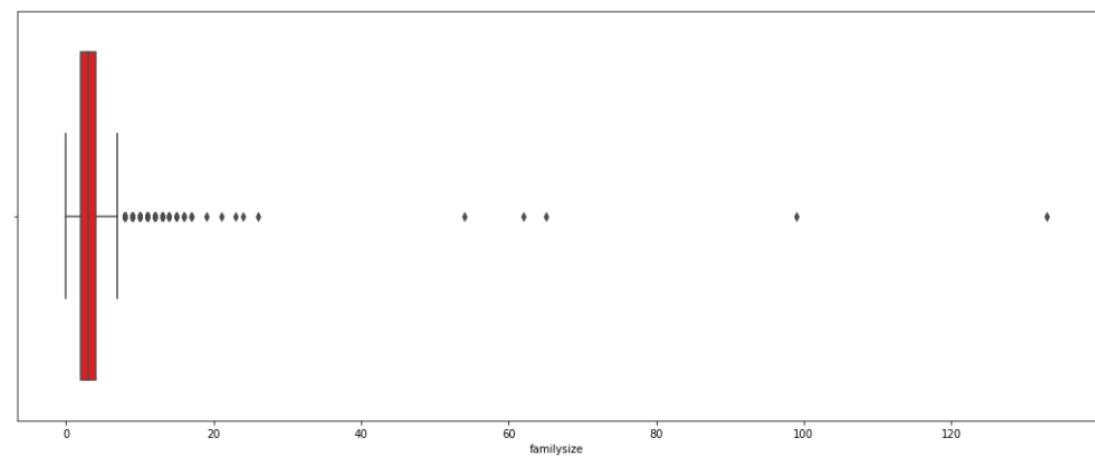
Out[238]: 3    9204
2    9016
4    7538
5    4829
1    2946
6    2449
7    1243
0    1125
8    676
9    331
10   195
11   109
12   56
13   19
14   8
15   6
16   5
17   3
24   1
23   1
21   1
19   1
62   1
26   1
65   1
54   1
99   1
133  1
Name: familysize, dtype: int64
```

```
In [239]: 1 plt.figure(figsize=(18, 7))
2 sns.distplot(x=df['familysize'], color='red')

Out[239]: <AxesSubplot:ylabel='Density'>
```



```
In [240]: 1 plt.figure(figsize=(18, 7))
2 sns.boxplot(x=df['familysize'], color='red')
Out[240]: <AxesSubplot:xlabel='familysize'>
```



```
In [241]: 1 familysize_indexes = df[df['familysize'] > 15]['familysize'].index
2 display(familysize_indexes)
3 print(f'DataFrame size before: {df.shape[0]}')
4 df = df.drop(familysize_indexes, axis=0)
5 print(f'DataFrame size after: {df.shape[0]}')

In[241]: Int64Index([ 1174,  2137,  6550, 16897, 17452, 17512, 17682, 18026, 18268,
   19781, 20779, 21771, 29800, 33483, 34247, 34522, 34829, 34901],
   dtype='int64')

DataFrame size before: 39768
DataFrame size after: 39750
```

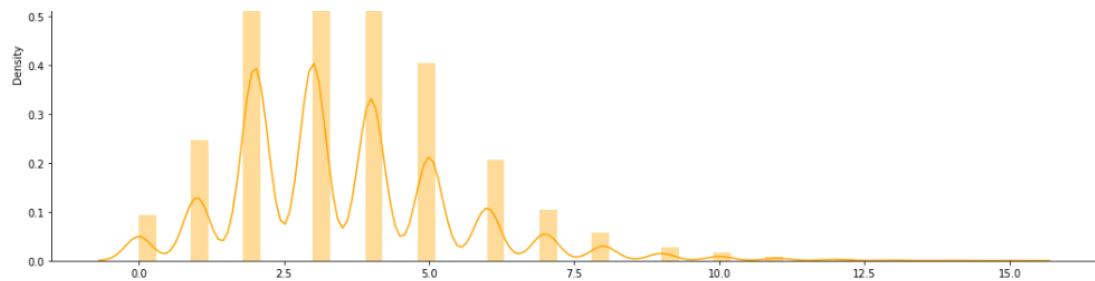
```
In [242]: 1 df['familysize'].unique()
Out[242]: array([ 2,  4,  3,  5,  1,  6,  8, 12,  7,  0, 11,  9, 13, 10, 14, 15],
   dtype=int64)
```

```
In [243]: 1 df['familysize'].value_counts()
```

```
Out[243]: 3    9204
2    9016
4    7538
5    4829
1    2946
6    2449
7    1243
0    1125
8    676
9    331
10   195
11   109
12   56
13   19
14    8
15    6
Name: familysize, dtype: int64
```

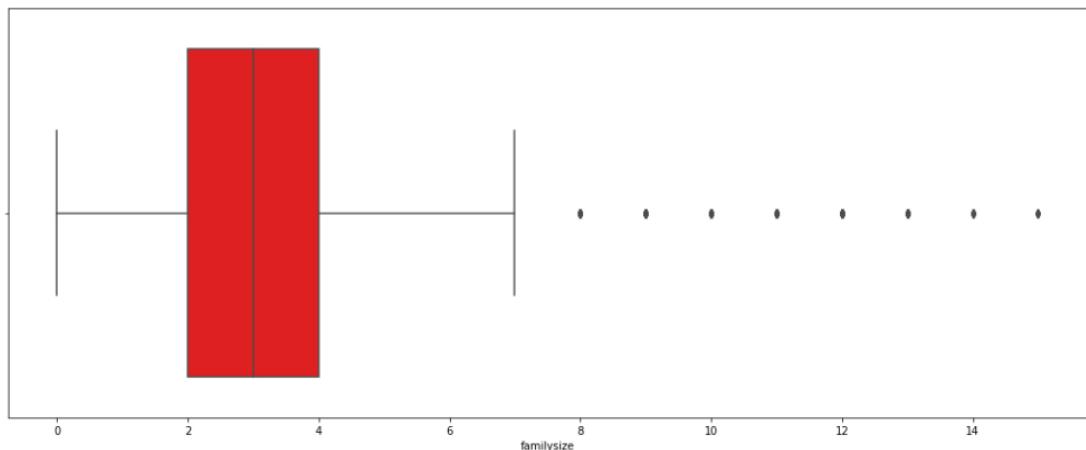
```
In [244]: 1 plt.figure(figsize=(18, 7))
2 sns.distplot(x=df['familysize'], color='orange')
Out[244]: <AxesSubplot:ylabel='Density'>
```





```
In [245]: 1 plt.figure(figsize=(18, 7))
2 sns.boxplot(x=df['familysize'], color='red')
```

Out[245]: <AxesSubplot:xlabel='familysize'>



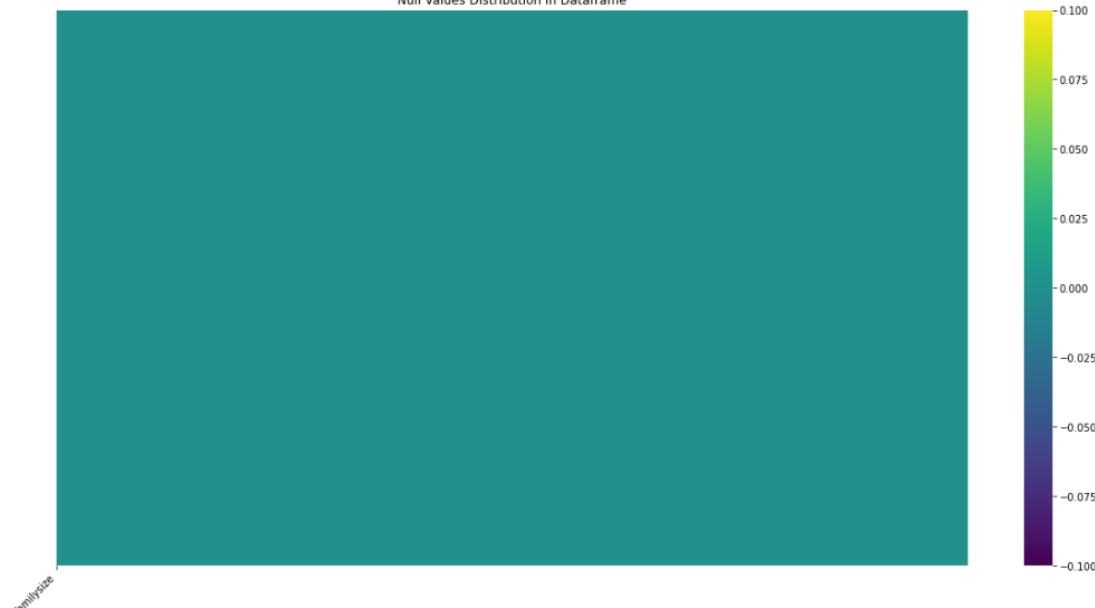
```
In [246]: 1 columns_to_check = df.columns[83:84]
2 plot_null_values(df, columns_to_check)
```

Null values in the specified columns:

|       | familysize |
|-------|------------|
| 0     | False      |
| 1     | False      |
| 2     | False      |
| 3     | False      |
| 4     | False      |
| ...   | ...        |
| 39770 | False      |
| 39771 | False      |
| 39772 | False      |
| 39773 | False      |
| 39774 | False      |

[39750 rows x 1 columns]

Null Values Distribution in Dataframe



## in Country Column

```
In [247]: 1 df.columns.get_loc('country')
2
```

Out[247]: 42

```
In [248]: 1 df['country'].unique()

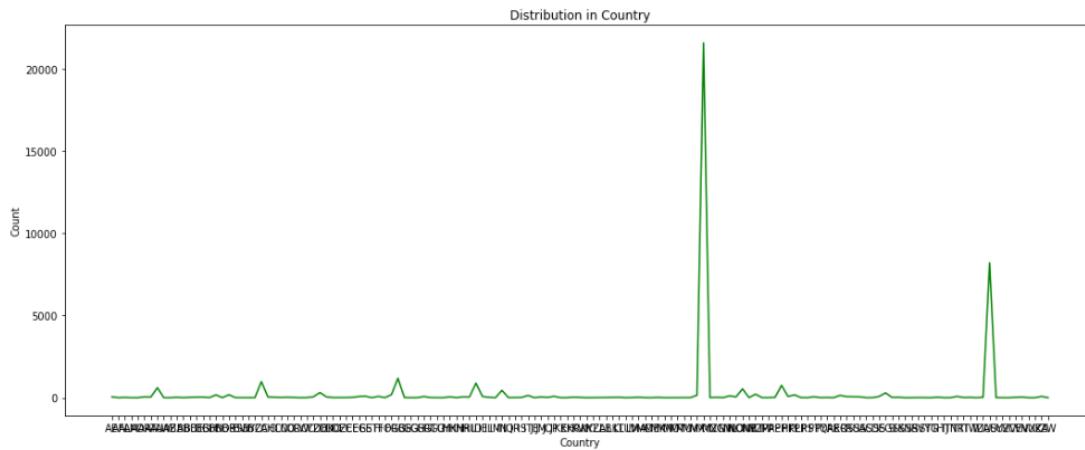
Out[248]: array(['IN', 'US', 'PL', 'MY', 'MX', 'GB', 'DE', 'EE', 'CA', 'EC', 'AU',
       'FI', 'ID', 'AE', 'NONE', 'ES', 'NI', 'DO', 'TR', 'NO', 'GR', 'LK',
       'BR', 'AR', 'KH', 'IT', 'MK', 'PT', 'BA', 'RU', 'HU', 'KR', 'IE',
       'RS', 'CZ', 'PH', 'RO', 'FR', 'SG', 'AT', 'UA', 'KW', 'CO', 'GE',
       'SE', 'HK', 'NZ', 'ZA', 'NL', 'DK', 'JM', 'LB', 'CL', 'AM', 'IL',
       'GT', 'PE', 'PK', 'NP', 'HR', 'CR', 'VE', 'BG', 'GH', 'MA', 'PR',
       'SI', 'LV', 'AL', 'VN', 'GU', 'BD', 'TH', 'NG', 'FO', 'UY', 'EG',
       'BE', 'SA', 'SV', 'CH', 'PY', 'SK', 'SY', 'TN', 'TT', 'SR', 'KE',
       'BV', 'PA', 'CY', 'BW', 'MZ', 'BB', 'GG', 'CW', 'IS', 'IR', 'BN',
       'MV', 'OM', 'DZ', 'JP', 'MU', 'nan', 'IQ', 'BO', 'MD', 'CN', 'LT',
       'BZ', 'BH', 'ME', 'QA', 'TW', 'PS', 'JO', 'SD', 'KY', 'AZ', 'AO',
       'BS', 'GY', 'MO', 'TZ', 'IM', 'AF', 'AM', 'UZ', 'LU', 'ET', 'JE',
       'VU', 'TJ', 'KZ', 'XK', 'HN', 'ZW', 'LA', 'MT', 'SN', 'MN', 'TG',
       'SC', 'RE', 'VC'], dtype=object)
```

```
In [249]: 1 df['country'].value_counts()

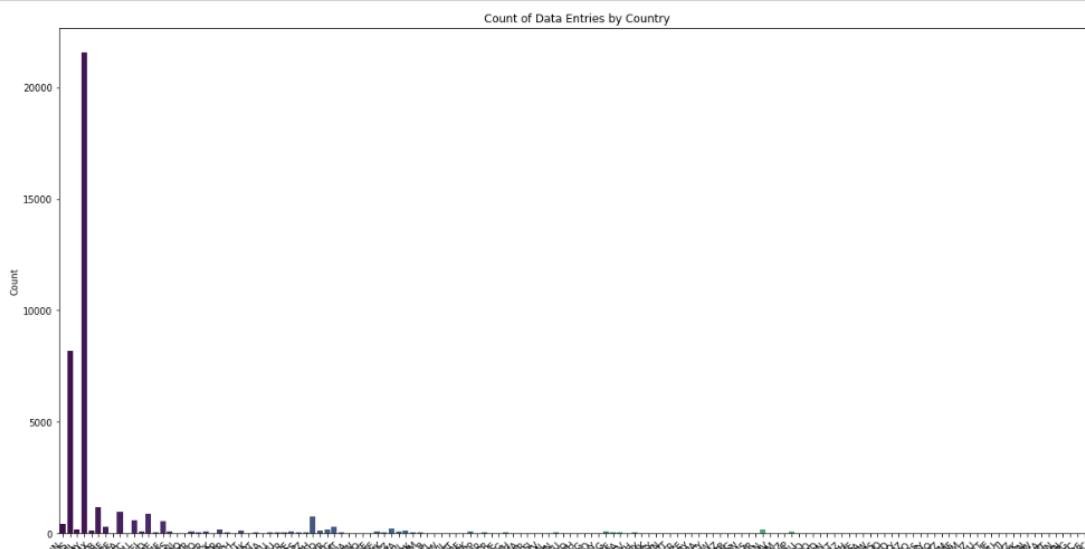
Out[249]: MY    21589
US    8283
GB    1179
CA     978
ID     884
...
UZ      1
AM      1
AF      1
IM      1
VC      1
Name: country, Length: 145, dtype: int64
```

```
In [250]: 1 value_counts = df['country'].value_counts().sort_index()
```

```
In [251]: 1 plt.figure(figsize=(18, 7))
2 plt.plot(value_counts.index, value_counts.values, color='g')
3 plt.xlabel('Country')
4 plt.ylabel('Count')
5 plt.title('Distribution in Country')
6 plt.show()
```

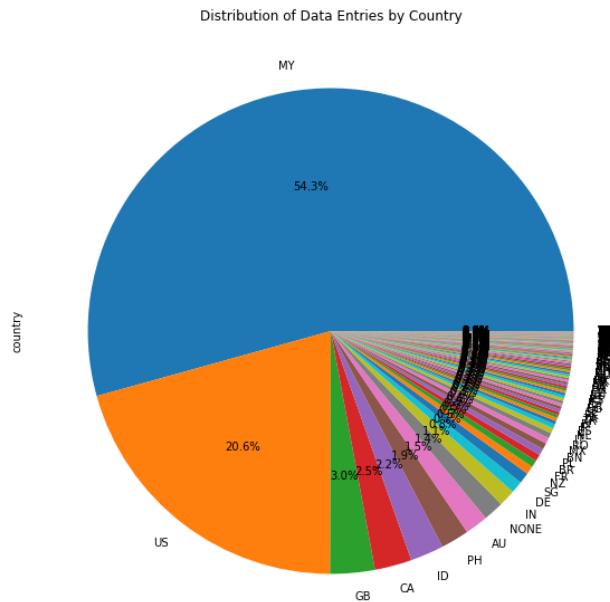


```
In [252]: 1 plt.figure(figsize=(20, 10))
2 sns.countplot(x=df['country'], data=df, palette='viridis')
3 plt.title('Count of Data Entries by Country')
4 plt.xlabel('Country')
5 plt.ylabel('Count')
6 plt.xticks(rotation=45, ha='right')
7 plt.show()
```



### Country

```
In [253]: 1 plt.figure(figsize=(10, 10))
2 df['country'].value_counts().plot.pie(autopct='%1.1f%%', startangle=0)
3 plt.title('Distribution of Data Entries by Country')
4 plt.show()
```



```
In [254]: 1 nan_values = df['country'].isnull().sum()
2 print(f"Number of NaN values in the 'country' column: {nan_values}")
Number of NaN values in the 'country' column: 2
```

```
In [255]: 1 sns.heatmap(df[['country']].isnull(), cbar=False, cmap='viridis')
```

Out[255]: <AxesSubplot:>

A vertical bar chart titled "country". The y-axis is labeled "count" and ranges from 0 to 37885 with major ticks every 1000 units. The x-axis is labeled "country" and has tick marks for every 1000 units, starting from 0 and ending at 37885. The bars are blue and show a clear downward trend as the value increases.

| country | count |
|---------|-------|
| 0       | 1894  |
| 1       | 3788  |
| 2       | 5682  |
| 3       | 7576  |
| 4       | 9470  |
| 5       | 11363 |
| 6       | 13256 |
| 7       | 15150 |
| 8       | 17044 |
| 9       | 18947 |
| 10      | 20840 |
| 11      | 22732 |
| 12      | 24625 |
| 13      | 26519 |
| 14      | 28412 |
| 15      | 30305 |
| 16      | 32200 |
| 17      | 34095 |
| 18      | 35992 |
| 19      | 37885 |

```
In [256]: 1 most_frequent_country = df['country'].value_counts().idxmax()
2 most_frequent_country
```

Out[256]: 'MY'

```
In [257]: 1 df['country'].fillna(most_frequent_country, inplace=True)
```

```
In [258]: 1 print("NaN values in 'country' column after filling:")
2 print(df['country'].isnull().sum())
```

NaN values in 'country' column after filling:  
0

## **in Screensize Column**

```
In [259]: 1 df.columns.get_loc('screensize')  
Out[259]: 75
```

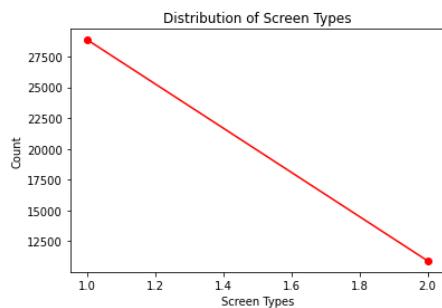
```
In [260]: 1 df['screensize'].unique()
```

```
In [261]: 1 df['orientation'].value_counts()

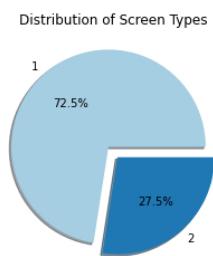
Out[261]: 1    27367
2     5004
5     3453
4     1974
3     1952
Name: orientation, dtype: int64
```

```
In [262]: 1 value_counts = df['screensize'].value_counts().sort_index()
```

```
In [263]: 1 plt.plot(value_counts.index, value_counts.values, color='r', marker='o')
2 plt.xlabel('Screen Types')
3 plt.ylabel('Count')
4 plt.title('Distribution of Screen Types')
5 plt.show()
```



```
In [264]: 1 explode_values = [0.071] * len(value_counts)
2 plt.pie(value_counts, labels=value_counts.index, explode=explode_values, autopct='%1.1f%%', shadow=True, startangle=0, label
3 plt.title('Distribution of Screen Types')
4 plt.show()
```

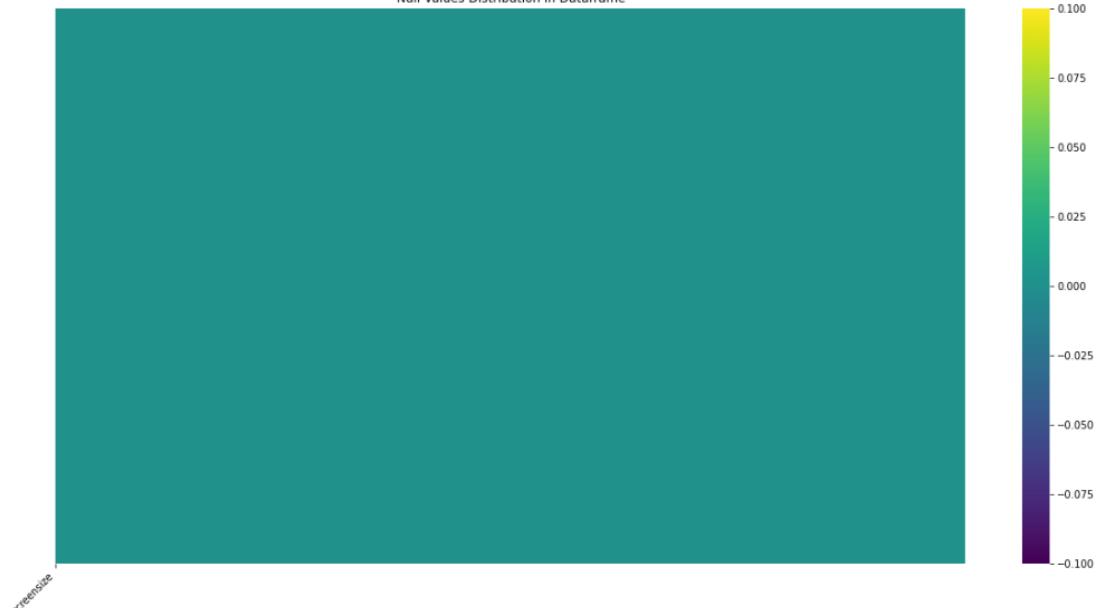


```
In [265]: 1 columns_to_check = df.columns[75:76]
2 plot_null_values(df, columns_to_check)
```

```
Null values in the specified columns:  
    screensize  
0      False  
1      False  
2      False  
3      False  
4      False  
...  
39770     False  
39771     False  
39772     False  
39773     False  
39774     False
```

[39750 rows x 1 columns]

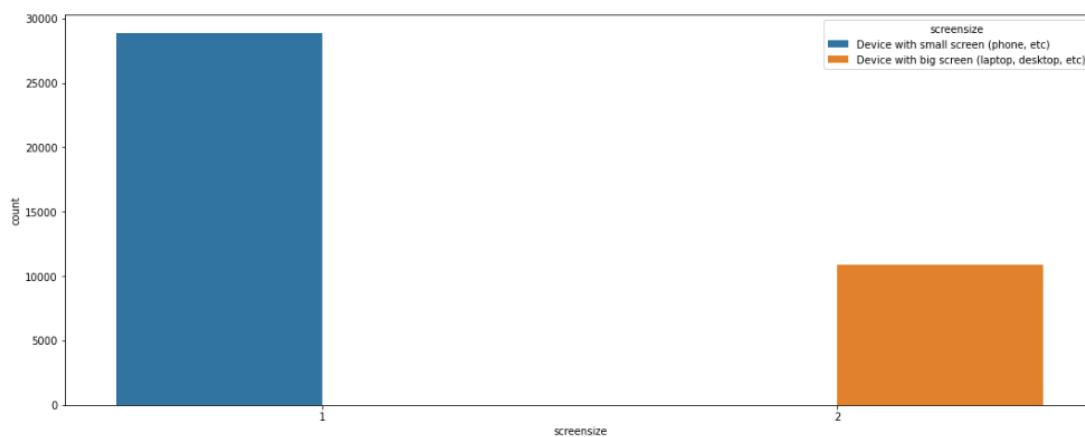
Null Values Distribution in Dataframe



```
In [266]: 1 def makeScreensizeGroup(value):
2     if value == 1:
3         return 'Device with small screen (phone, etc)'
4     if value == 2:
5         return 'Device with big screen (laptop, desktop, etc)'
```

```
In [267]: 1 married = df['screensize'].apply(makeScreensizeGroup)
```

```
In [268]: 1 plt.figure(figsize=(18, 7))
2 sns.countplot(x=df['screensize'], hue=married)
Out[268]: <AxesSubplot:xlabel='screensize', ylabel='count'>
```



### in Uniquenetworklocation Column

```
In [269]: 1 df.columns.get_loc('uniquenetworklocation')
Out[269]: 76
```

```
In [270]: 1 df['uniquenetworklocation'].unique()
```

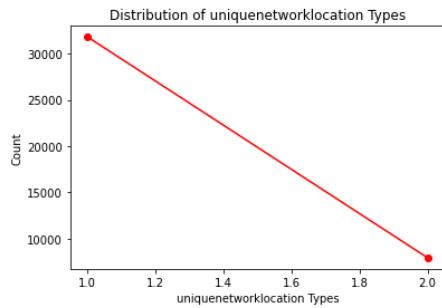
```
Out[270]: array([1, 2], dtype=int64)
```

```
In [271]: 1 df['uniquenetworklocation'].value_counts()
```

```
Out[271]: 1    31799
2    7951
Name: uniquenetworklocation, dtype: int64
```

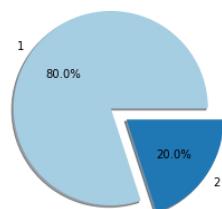
```
In [272]: 1 value_counts = df['uniquenetworklocation'].value_counts().sort_index()
```

```
In [273]: 1 plt.plot(value_counts.index, value_counts.values, color='r', marker='o')
2 plt.xlabel('uniquenetworklocation Types')
3 plt.ylabel('Count')
4 plt.title('Distribution of uniquenetworklocation Types')
5 plt.show()
```



```
In [274]: 1 explode_values = [0.1] * len(value_counts)
2 plt.pie(value_counts, labels=value_counts.index, explode=explode_values, autopct='%1.1f%%', shadow=True, startangle=0, label
```

Distribution of uniquenetworklocation Types



```
In [275]: 1 nan_values = df['uniquenetworklocation'].isnull().sum()
2 print(f"Number of NaN values in the 'country' column: {nan_values}")

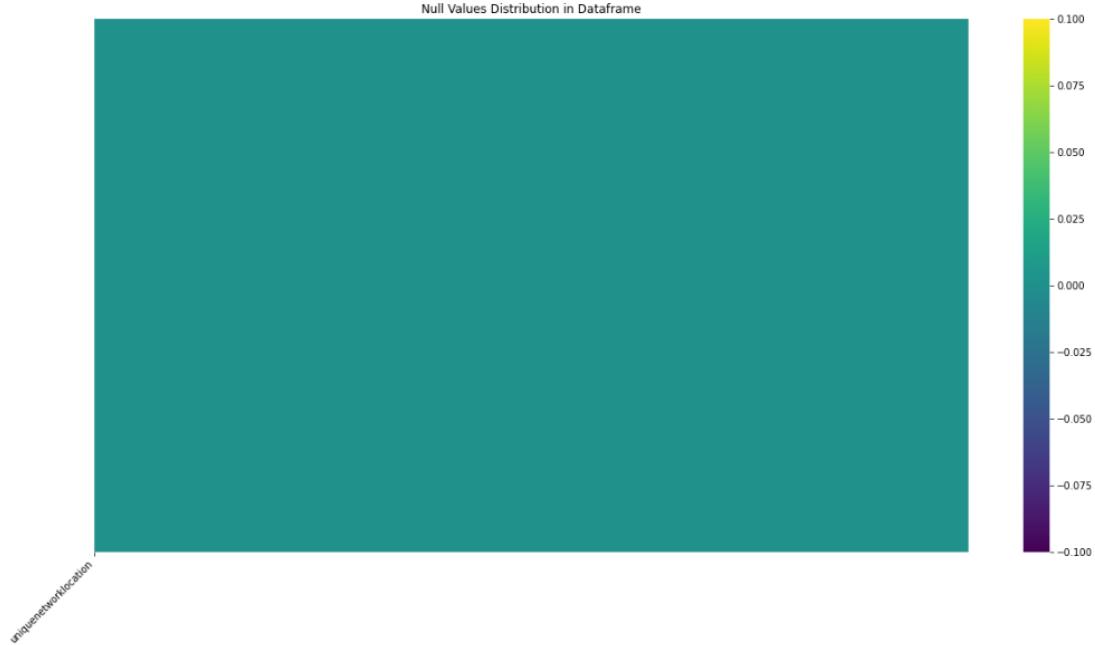
Number of NaN values in the 'country' column: 0
```

```
In [276]: 1 columns_to_check = df.columns[76:77]
2 plot_null_values(df, columns_to_check)
```

Null values in the specified columns:

```
uniquenetworklocation
0                  False
1                  False
2                  False
3                  False
4                  False
...
39770                 ...
39771                 False
39772                 False
39773                 False
39774                 False

[39750 rows x 1 columns]
```

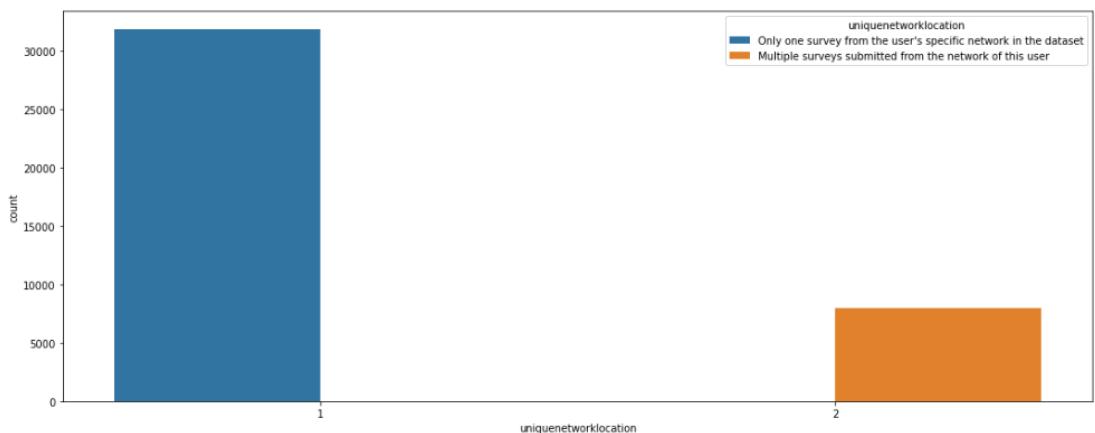


```
In [277]: 1 def makeUniquenetworklocationGroup(value):
2     if value == 1:
3         return 'Only one survey from the user\'s specific network in the dataset'
4     if value == 2:
5         return 'Multiple surveys submitted from the network of this user'
```

```
In [278]: 1 uniquenetworklocation = df['uniquenetworklocation'].apply(makeUniquenetworklocationGroup)
```

```
In [279]: 1 plt.figure(figsize=(18, 7))
2 sns.countplot(x=df['uniquenetworklocation'], hue=uniquenetworklocation)
```

```
Out[279]: <AxesSubplot:xlabel='uniquenetworklocation', ylabel='count'>
```



## in Source Column

```
In [280]: 1 df.columns.get_loc('source')
```

```
Out[280]: 43
```

```
In [281]: 1 df['source'].unique()
```

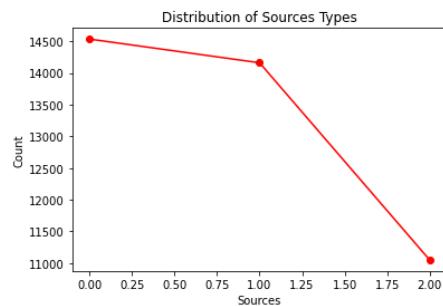
```
Out[281]: array([2, 0, 1], dtype=int64)
```

```
In [282]: 1 df['source'].value_counts()
```

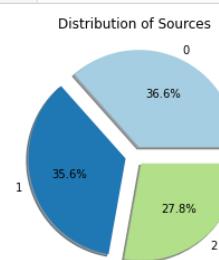
```
Out[282]: 0    14536
1    14162
2    11052
```

```
Name: source, dtype: int64
```

```
In [283]: 1 value_counts = df['source'].value_counts().sort_index()  
In [284]: 1 plt.plot(value_counts.index, value_counts.values, color='r', marker='o')  
2 plt.xlabel('Sources')  
3 plt.ylabel('Count')  
4 plt.title('Distribution of Sources Types')  
5 plt.show()
```



```
In [285]: 1 explode_values = [0.1] * len(value_counts)  
2 plt.pie(value_counts, labels=value_counts.index, explode=explode_values, autopct='%1.1f%%', shadow=True, startangle=0, label  
3 plt.title('Distribution of Sources')  
4 plt.show()
```



```
In [286]: 1 nan_values = df['uniquenetworklocation'].isnull().sum()  
2 print(f"Number of NaN values in the 'country' column: {nan_values}")
```

Number of NaN values in the 'country' column: 0

```
In [287]: 1 columns_to_check = df.columns[43:44]  
2 plot_null_values(df, columns_to_check)
```

Null values in the specified columns:

|       | source |
|-------|--------|
| 0     | False  |
| 1     | False  |
| 2     | False  |
| 3     | False  |
| 4     | False  |
| ...   | ...    |
| 39770 | False  |
| 39771 | False  |
| 39772 | False  |
| 39773 | False  |
| 39774 | False  |

[39750 rows x 1 columns]

Null Values Distribution in Dataframe

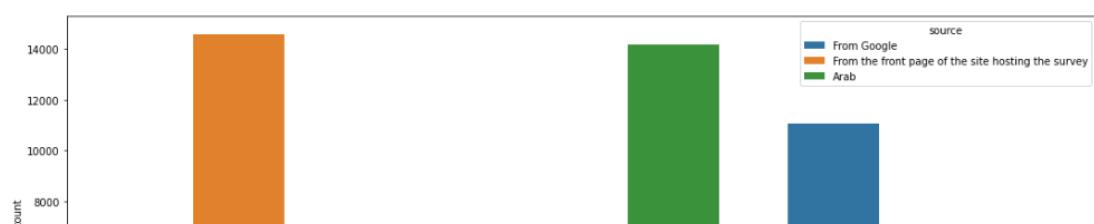


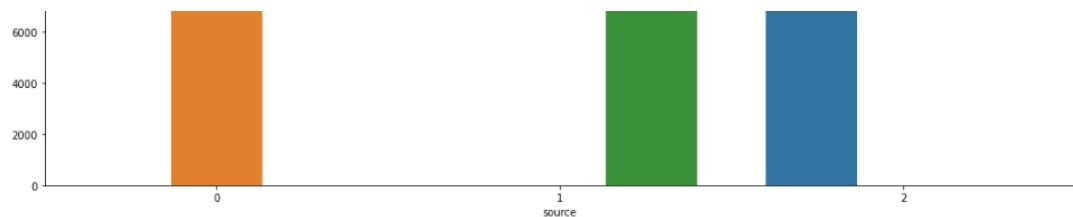
```
In [288]: 1 def makeSourceGroup(value):  
2     if value == 0:  
3         return 'From the front page of the site hosting the survey'  
4     if value == 1:  
5         return 'Arab'  
6     if value == 2:  
7         return 'From Google'
```

```
In [289]: 1 source = df['source'].apply(makeSourceGroup)
```

```
In [290]: 1 plt.figure(figsize=(18, 7))  
2 sns.countplot(x=df['source'], hue=source)
```

```
Out[290]: <AxesSubplot:xlabel='source', ylabel='count'>
```





## in Major Column

```
In [291]: 1 df.columns.get_loc('major')
```

```
Out[291]: 84
```

```
In [292]: 1 df['major'].unique()
```

```
Out[292]: array([nan, 'biology', 'Psychology', ...,
   '&#22810;&#23186;&#39636;&#35373;&#35336;',
   'public relation or administrations', 'computer sciece'],
  dtype=object)
```

```
In [293]: 1 df['major'].value_counts()
```

```
Out[293]: English          1025
Psychology        850
Business          620
Accounting        606
Engineering       582
...
Mathematics & English    1
Comp Science       1
Art and Education  1
Veterinary science  1
computer sciece    1
Name: major, Length: 5308, dtype: int64
```

```
In [294]: 1 value_counts = df['major'].value_counts().sort_index()
```

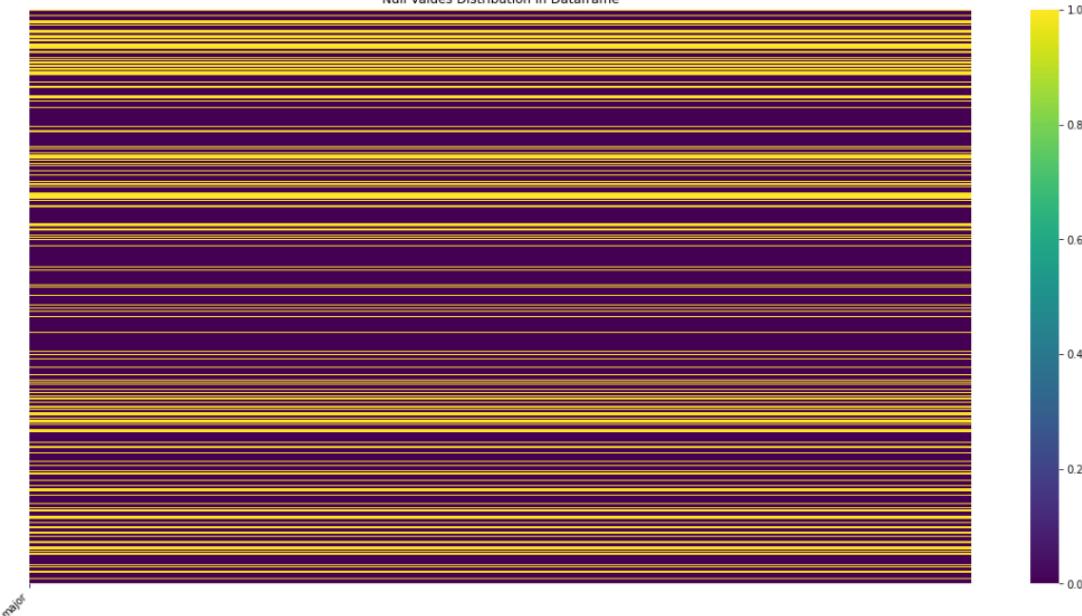
```
In [295]: 1 columns_to_check = df.columns[84:85]
2 plot_null_values(df, columns_to_check)
```

Null values in the specified columns:

|       |       |
|-------|-------|
| 0     | True  |
| 1     | True  |
| 2     | True  |
| 3     | False |
| 4     | False |
| ...   | ...   |
| 39770 | True  |
| 39771 | False |
| 39772 | False |
| 39773 | False |
| 39774 | False |

[39750 rows x 1 columns]

Null Values Distribution in Dataframe



```
In [296]: 1 nan_values = df['major'].isnull().sum()
```

```
2 print(f"Number of NaN values in the 'major' column: {nan_values}")
```

Number of NaN values in the 'major' column: 11399

```
In [297]: 1 def condition(title):
```

```

2     if 'busin' in str(title).lower() or 'manage' in str(title).lower() or 'Buss' in str(title) or 'Bisness' in str(title) or
3         return 'Business/Management'
4     elif 'information technology' in str(title).lower() or 'IT' in str(title) or 'it' in str(title):
5         return 'I.T'
6     elif 'math' in str(title).lower() or 'LOGISTICS' in str(title) or 'st' in str(title).lower() or 'math' in str(title).low
7         return 'Mathematics'
8     elif 'computer' in str(title).lower():
9         return 'I.T'
10    elif 'bio' in str(title).lower() or 'Plant' in str(title) or 'plant' in str(title).lower() or 'Micro' in str(title):
11        return 'Biology'
12    elif 'tesl' in str(title).lower() or 'TES' in str(title) or 'Teso' in str(title) or 'Enhlish' in str(title):
13        return 'English'
14    elif 'account' in str(title).lower() or 'Accoun' in str(title) or 'Acc' in str(title) or 'acc' in str(title).lower() or
15        return 'Accountacy'
16    elif 'CA' in str(title):
17        return 'CA'
18    elif 'none' in str(title).lower() or '0' in str(title) or '_' in str(title) or '.' in str(title) or 'Mine' in str(title):
19        return 'No Degree'
20    elif 'nurs' in str(title).lower() or 'BSN' in str(title):
21        return 'Nursing'
22    elif '-' in str(title).lower() or 'NIL' in str(title):
23        return 'No Degree'
24    elif 'teach' in str(title).lower() or 'Lect' in str(title) or 'eet' in str(title).lower():
25        return 'Teaching'
26    elif 'pharma' in str(title).lower() or 'medic' in str(title).lower() or 'med' in str(title).lower() or 'hospi' in str(ti
27        return 'Pharmacy/Medical'
28    elif 'doctor' in str(title).lower() or 'MBBS' in str(title) or 'Mbbs' in str(title) or 'Surge' in str(title) or 'surge'
29        return 'Doctor'
30    elif 'no' in str(title).lower() or 'Undec' in str(title) or 'Idk' in str(title) or 'idk' in str(title).lower() or 'Hahah
31        return 'No Degree'
32    elif 'film' in str(title).lower() or 'Cinema' in str(title) or 'fil' in str(title).lower() or 'Adver' in str(title) or '
33        return 'Media'
34    elif 'international' in str(title).lower() or 'Internatiional' in str(title):
35        return 'International Relations'
36    elif 'human' in str(title).lower() or 'hr' in str(title).lower() or 'Hs' in str(title) or 'Hm' in str(title) or 'Humam'
37        return 'Human Resources'
38    elif 'art' in str(title).lower() or 'Painting' in str(title) or 'Drawing' in str(title) or 'ba' in str(title) or 'Printi
39        return 'Arts'
40    elif 'islam' in str(title).lower() or 'Muamalat' in str(title) or 'Quran' in str(title) or 'Halal' in str(title) or 'Usu
41        return 'Islamic Studies'
42    elif 'physio' in str(title).lower() or 'fis' in str(title).lower():
43        return 'Physiotherapy'
44    elif 'socio' in str(title).lower() or 'social' in str(title).lower() or 'soical' in str(title).lower() or 'Sis' in str(t
45        return 'Sociology'
46    elif 'bank' in str(title).lower():
47        return 'Banking'
48    elif 'agri' in str(title).lower():
49        return 'Agriculture'
50    elif 'Market' in str(title) or 'Finan' in str(title) or 'finance' in str(title).lower() or 'MARKETING' in str(title) or
51        return 'Marketing/Finance'
52    elif 'counsel' in str(title).lower() or 'cauns' in str(title) or 'Kaunseling' in str(title) or 'kaunseling' in str(title)
53        return 'Counselling'
54    elif 'programming' in str(title).lower() or 'coding' in str(title).lower() or 'Ibm' in str(title) or 'ceit' in str(title)
55        return 'I.T'
56    elif 'civil' in str(title).lower() or 'comp' in str(title).lower() or 'Mechanical' in str(title) or 'Electrical' in str(
57        return 'Engineering'
58    elif 'ict' in str(title).lower() or 'developer' in str(title).lower() or 'I.T' in str(title) or 'CAE&D' in str(title) or
59        return 'I.T'
60    elif 'commu' in str(title).lower() or 'comm' in str(title).lower() or 'com' in str(title).lower() or 'Communication' in
61        return 'Communications'
62    elif 'administration' in str(title).lower() or 'admin' in str(title).lower():
63        return 'Administration'
64    elif 'psycho' in str(title).lower() or 'psy' in str(title).lower() or 'Clinical osychology' in str(title) or 'hschology'
65        return 'Psychology'
66    elif 'english' in str(title).lower() or 'Elglish' in str(title) or 'esl' in str(title).lower() or 'Emg' in str(title) or
67        return 'English'
68    elif 'law' in str(title).lower() or 'BBA' in str(title) or 'llb' in str(title) or 'lew' in str(title).lower() or 'kaw' i
69        return 'Law'
70    elif 'engineering' in str(title).lower() or 'engi' in str(title).lower() or 'eng' in str(title).lower() or 'Software' in
71        return 'Engineering'
72    elif 'architecture' in str(title).lower() or 'aechitecture' in str(title).lower() or 'archireecture' in str(title).lower(
73        return 'Architecture'
74    elif 'design' in str(title).lower() or 'Desig' in str(title) or 'Dssign' in str(title):
75        return 'Designer'
76    elif 'science' in str(title).lower() or 'Sceince' in str(title) or 'Sci' in str(title) or 'sciene' in str(title) or 'BS'
77        return 'Pure Sciences'
78    elif 'physics' in str(title).lower() or 'Phsyics' in str(title) or 'EMC' in str(title) or 'Physic' in str(title) or 'phy
79        return 'Physics'
80    elif 'chemistry' in str(title).lower() or 'CIS' in str(title) or 'Chem' in str(title):
81        return 'Chemistry'
82    elif 'french' in str(title).lower() or 'Fr' in str(title):
83        return 'French'
84    elif 'religi' in str(title).lower() or 'Relegion' in str(title) or 'Rel' in str(title) or 'Hukum' in str(title) or 'Sair
85        return 'Religious Studies'
86    elif title=='&#1593;&#1604;&#1605; &#1606;&#1601;&#1587;' or title=='&#22810;&#23186;&#39636;&#35336;' or title=
87        return 'No Degree'
88    elif 'Music' in str(title) or 'Dance' in str(title) or 'danc' in str(title).lower() or 'Vocational' in str(title) or 'Mu
89        return 'Music/Dance'
90    elif 'pol' in str(title).lower() or 'Govern' in str(title) or 'Right' in str(title):
91        return 'Politics'
92    elif 'photo' in str(title).lower() or 'Foto' in str(title) or 'Photo' in str(title):
93        return 'Photography'
94    elif 'Television' in str(title) or 'telev' in str(title).lower():
95        return 'Television'
96    elif 'bahasa' in str(title).lower() or 'Bahasa' in str(title) or 'Malay' in str(title) or 'malay' in str(title).lower():
97        return 'Malaysian languages'
98    elif 'Urban' in str(title) or 'Town' in str(title) or 'town' in str(title).lower() or 'planning' in str(title) or 'Planr
99        return 'Economic Developments'
100   elif 'Public' in str(title) or 'public' in str(title).lower():
101      return 'Public Relations'
102   elif 'Writing' in str(title) or 'writing' in str(title).lower() or 'Screenwritinf' in str(title) or 'Author' in str(titl
103      return 'Writing/Author'
104   elif 'philosophy' in str(title).lower() or 'Phil' in str(title) or 'philos' in str(title).lower() or 'Filo' in str(title)
105      return 'Philosophy'
106   elif 'Actua' in str(title):
107      return 'Actuarial Studiess'

```

```

    'ACCURIAL STUDIES'
108 elif 'DENTALWORKS' in str(title) or 'dental' in str(title) or 'Dental' in str(title) or 'Odont' in str(title):
109     return 'Dentist'
110 elif 'beaut' in str(title).lower() or 'Fashion' in str(title) or 'make' in str(title) or 'fashion' in str(title).lower():
111     return 'Fashion'
112 elif 'Health' in str(title) or 'health' in str(title).lower() or 'wellness' in str(title).lower() or 'Healthcare' in str(
113     return 'Healthcare'
114 elif 'Language' in str(title) or 'lang' in str(title).lower() or 'Laq' in str(title):
115     return 'Languages'
116 elif 'cook' in str(title).lower() or 'bakery' in str(title).lower() or 'Bak' in str(title) or 'CULINARY' in str(title) or
117     return 'Cookings'
118 elif 'Hotel' in str(title) or 'hotel' in str(title).lower() or 'food service' in str(title) or 'cater' in str(title).low
119     return 'Hotel Management'
120 elif 'therapy' in str(title).lower() or 'ot' in str(title).lower() or 'theraphy' in str(title):
121     return 'Therapeutical Studies'
122 elif 'veter' in str(title).lower() or 'Veter' in str(title) or 'Vet' in str(title):
123     return 'Veterinary'
124 elif 'Survey' in str(title) or 'survey' in str(title) or 'serveyors' in str(title).lower() or 'Qs' in str(title) or 'SUR
125     return 'Surveyor Studies'
126 elif 'Aircraft' in str(title) or 'aircraft' in str(title).lower() or 'aircr' in str(title).lower() or 'aviation' in str(
127     return 'Aircrafts'
128 elif 'environment' in str(title).lower() or 'Environment' in str(title) or 'envi' in str(title).lower():
129     return 'Environmental Educations'
130 elif 'Syariah' in str(title) or 'syariah' in str(title):
131     return 'Syrian Languages'
132 elif 'judicial' in str(title).lower() or 'juri' in str(title).lower() or 'legal' in str(title).lower():
133     return 'Judicial Studies'
134 elif 'Liter' in str(title) or 'literature' in str(title) or 'litt' in str(title).lower():
135     return 'Literature'
136 elif 'child' in str(title).lower() or 'Child' in str(title) or 'Preschool' in str(title):
137     return 'Child Educations'
138 elif 'Tour' in str(title) or 'tour' in str(title).lower():
139     return 'Tourisms'
140 elif 'Gam' in str(title) or 'game' in str(title).lower():
141     return 'Gaming'
142 elif 'education' in str(title).lower() or 'Education' in str(title) or 'ed' in str(title).lower() or 'acad' in str(title)
143     return 'B.Ed or M.Ed'
144 elif 'Sport' in str(title) or 'sport' in str(title).lower():
145     return 'Sports'
146 elif 'Petro' in str(title):
147     return 'Petroleum'
148 elif 'Journ' in str(title) or 'jour' in str(title).lower() or 'Joun' in str(title) or 'Jurn' in str(title):
149     return 'Journalism'
150 elif 'Mandarin' in str(title):
151     return 'Chinese/Mandarin Languages'
152 elif 'Electrician' in str(title):
153     return 'Electrician'
154 elif 'Network' in str(title) or 'network' in str(title).lower():
155     return 'Networking'
156 elif 'geo' in str(title).lower() or 'GEO' in str(title):
157     return 'Geography'
158 elif 'Librarian' in str(title) or 'lib' in str(title).lower():
159     return 'Librarian'
160 elif 'Mission' in str(title) or 'mission' in str(title).lower():
161     return 'Missionary Studies'
162 elif 'Forensic' in str(title) or 'foren' in str(title).lower() or 'Crime' in str(title) or 'crim' in str(title).lower():
163     return 'Forensic/Criminal studies'
164 elif 'Animation' in str(title) or 'animation' in str(title).lower() or 'imag' in str(title) or 'graphic' in str(title) or
165     return 'Animations'
166 elif 'aqua' in str(title).lower() or 'Aqu' in str(title):
167     return 'Aquaculture'
168 elif 'soldier' in str(title).lower() or 'lwa' in str(title).lower() or 'defence' in str(title):
169     return 'Army'
170 elif 'Kinesi' in str(title) or 'Kines' in str(title).lower() or 'hod' in str(title):
171     return 'Human Kinetics'
172 elif 'Horti' in str(title) or 'horti' in str(title) or 'Landscape' in str(title):
173     return 'Horticulture'
174 elif 'commerce' in str(title).lower() or 'Coome' in str(title):
175     return 'Commerce'
176 elif 'Speech' in str(title) or 'speech' in str(title).lower():
177     return 'Speech Pathology'
178 elif 'SECRET' in str(title) or 'secret' in str(title).lower():
179     return 'Secretary'
180 elif 'Animals' in str(title) or 'animal' in str(title).lower() or 'Pet' in str(title):
181     return 'Animal Care'
182 elif 'Organisation' in str(title) or 'organi' in str(title).lower():
183     return 'Organizational Behaviour'
184 elif 'event' in str(title).lower() or 'Event' in str(title):
185     return 'Event Managment'
186 elif 'radiology' in str(title).lower() or 'Radiography' in str(title) or 'radiographhy' in str(title).lower() or 'Radiat
187     return 'Radiography'
188 elif 'nutrition' in str(title).lower() or 'Nutrition' in str(title):
189     return 'Nutritionist'
190 elif 'Audit' in str(title) or 'audit' in str(title).lower():
191     return 'Auditing'
192 elif 'Neuro' in str(title) or 'neuroligy' in str(title).lower():
193     return 'Neurology'
194 elif 'Anato' in str(title) or 'anat' in str(title).lower():
195     return 'Anatomy'
196 elif 'trade' in str(title).lower():
197     return 'Trading'
198 elif 'Interpre' in str(title) or 'translation' in str(title).lower():
199     return 'Interpreter'
200 elif 'audio' in str(title).lower() or 'Audio' in str(title):
201     return 'Audiology'
202 elif 'insurance' in str(title).lower() or 'Insurance' in str(title):
203     return 'Insurances'
204 elif 'archeology' in str(title).lower() or 'archaeology' in str(title).lower() or 'archeology' in str(title).lower() or
205     return 'Archeology'
206 elif 'SERV'in str(title) or 'service' in str(title).lower():
207     return 'Service Training'
208 elif 'GERMAN' in str(title) or 'german' in str(title).lower():
209     return 'German'
210 elif 'KOREAN' in str(title) or 'Korea' in str(title):
211     return 'Korean'
212 elif 'valuat' in str(title).lower() or 'valuer' in str(title).lower():

```

```

213     return 'Registered Valuer'
214 elif 'skil' in str(title).lower() or 'Skill' in str(title) or 'Professional' in str(title) or 'practical' in str(title):
215     return 'Skilled Labour'
216 elif 'virology' in str(title):
217     return 'Virology'
218 elif 'lab' in str(title).lower() or 'Lab' in str(title) or 'MLT' in str(title):
219     return 'Laboratory Worker'
220 elif 'GENERAL' in str(title) or 'General' in str(title):
221     return 'General'
222 elif 'Opto' in str(title) or 'opto' in str(title).lower():
223     return 'Optometry'
224 elif 'Zoo' in str(title) or 'zoo' in str(title).lower():
225     return 'Zoology'
226 elif 'office' in str(title).lower() or 'Office' in str(title):
227     return 'Office Skills'
228 elif 'found' in str(title).lower() or 'Found' in str(title):
229     return 'Foundation Education'
230 elif 'general' in str(title).lower() or 'General' in str(title):
231     return 'General Education'
232 elif 'real estate' in str(title).lower() or 'property' in str(title).lower():
233     return 'Realtor'
234 elif 'Meteorology' in str(title) or 'Metrology' in str(title):
235     return 'Meteorology'
236 elif 'operations' in str(title).lower() or 'Operation' in str(title):
237     return 'Operational Management'
238 elif 'Merchandising' in str(title) or 'merchand' in str(title).lower():
239     return 'Merchandising'
240 elif 'Spanish' in str(title):
241     return 'Spanish'
242 elif 'Nature' in str(title) or 'natur' in str(title).lower():
243     return 'Nature Conservation/Resources'
244 elif title=='a level ' or title==' ':
245     return 'No Degree'
246 elif 'Corporate' in str(title) or 'corporate' in str(title).lower():
247     return 'Corporate'
248 elif 'greek' in str(title).lower() or 'Greek' in str(title):
249     return 'Greek'
250 elif 'Behaviour' in str(title) or 'Behavior' in str(title) or 'Organizational Behaviour' in str(title):
251     return 'Behaviour Analysis'
252 elif 'publish' in str(title).lower():
253     return 'Publishing'
254 elif 'Safety' in str(title) or 'safety' in str(title).lower():
255     return 'Safety Training'
256 elif 'genetic' in str(title).lower() or 'Genetic' in str(title):
257     return 'Genetics'
258 elif 'Dietetic' in str(title):
259     return 'Dietician'
260 elif 'Production' in str(title) or 'manufacturing' in str(title).lower():
261     return 'Production And Manufacturing'
262 elif 'Welding' in str(title):
263     return 'Welding'
264 elif 'Geron' in str(title):
265     return 'Gerontology'
266 elif 'Research' in str(title) or 'Ph D' in str(title):
267     return 'Ph.D'
268 elif 'arabic' in str(title).lower() or 'Arabic' in str(title):
269     return 'Arabic'
270 else:
271     return title

```

```
In [298]: 1 df['major'] = df['major'].apply(condition)
```

```
In [299]: 1 lis=[]
2 for x in df['major']:
3     lis.append(x)
4 print(set(lis))
```

```
{nan, 'Pathology', 'Corrosion ', 'Doctor', 'Journalism', 'Information', 'Arts', 'Geography', 'Builders', 'Mining ', 'Zoology',
'clinical measurement', 'Religious Studies', 'Animal Care', 'General', 'Languages', 'Veterinary', 'Anatomy', 'Palaeontology',
'Insurances', 'Tourisms', 'Accountancy', 'Counselling', 'Supply chain', 'Meterology', 'Dutch', 'Child Educations', 'Kiin', 'General Education',
'French', 'Aircrafts', 'Banking', 'Arabic', 'Registered Valuer', 'Politics', 'Marketing/Finance', 'Interpreter',
'Sociology', 'hvac', 'Merchandising', 'Media', 'Gerontology', 'Trading', 'Office Skills', 'Architecture', 'Missionary Studies',
'Malaysian languages', 'College diploma', 'Japanese', 'Creative Visual', 'Writing/Author', 'Bachelor', 'Hotel Management',
'Spanish', 'Youth Work', 'Law', 'Foundation Education', 'Chargeman', 'Russian philology', 'Army', 'Organizational Behaviour',
'Bookkeeping', 'Sports', 'International Relations', 'Business/Management', 'Photography', 'theology', 'Leadership', 'Optometry',
'Animations', 'Healthcare', 'Museology', 'usa', 'Product Development', 'Aquaculture', 'Human Resources', 'Service Training',
'Psychology', 'Information', 'German', 'Corporate', 'Radiography', 'Danish', 'Skilled Labour', 'Laboratory Worker', 'B.Ed or M.Ed',
'Music/Dance', 'Economic Developments', 'Ophthalmology', 'Physics', 'Physiotherapy', 'Afrikaans', 'Korean', 'Nature Conservation/Resources',
'Publishing', 'Behaviour Analysis', 'Environmental Educations', 'Networking', 'Commerce', 'Gis', 'English',
'Philosophy', 'Operational Management', 'Ph.D', 'Forensic/Criminal studies', 'Islamic Studies', 'Sculpture', 'Chinese', 'CA',
'Neurology', 'Designer', 'Therapeutical Studies', 'Librarian', 'Human Kinetics', 'Gaming', 'Realtor', 'Virology', 'Surveyor Studies',
'Crisis intervention', 'Dentist', 'Cookings', 'Singer', 'Petroleum', 'epidemiology', 'Fashion', 'Pure Sciences', 'Genetics',
'Welding', 'I.T', 'harvard', 'Audiology', 'maintenance', 'midwivery', 'No Degree', 'Teaching', 'Agriculture', 'Safety Training',
'Judicial Studies', 'Dietician', 'Greek', 'Pharmacy/Medical', 'Administration', 'Horticulture', 'Syrian Languages',
'Biology', 'Event Managment', 'Cabin Crew', 'Mathematics', 'Wildlife resources', 'Communications', 'Production And Manufacturing',
'Engineering', 'Archeology', 'portuguese', 'Nursing', 'Public Relations', 'Chemistry', 'IR'}
```

```
In [300]: 1 columns_to_check = df.columns[84:85]
2 plot_null_values(df, columns_to_check)
```

Null values in the specified columns:

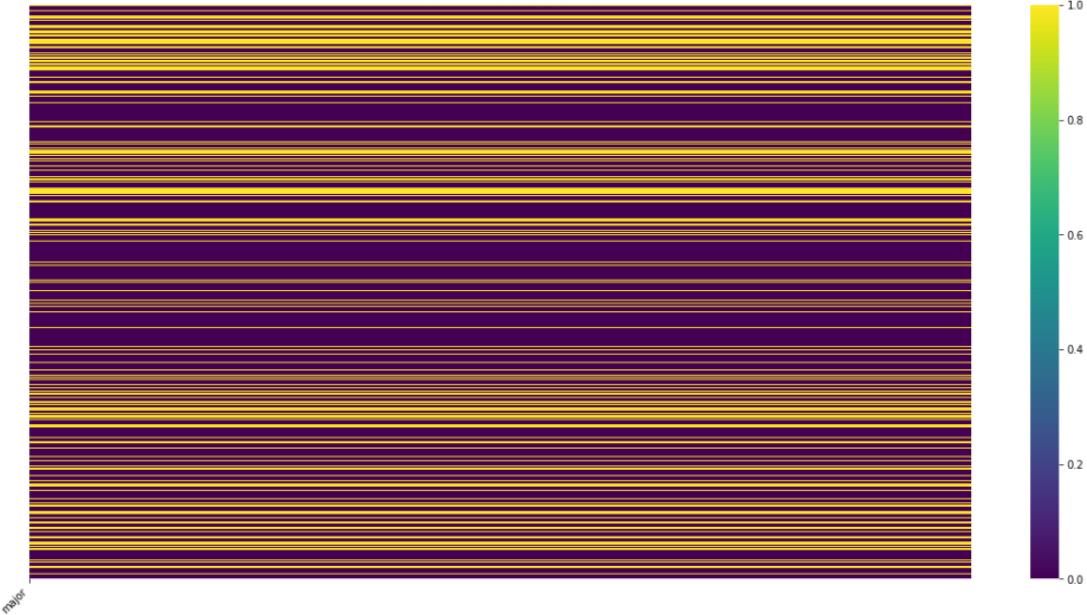
```

major
0    True
1    True
2    True
3   False
4   False
...
39770  True
39771 False
39772 False
39773 False
39774 False

```

```
[39750 rows x 1 columns]
```

Null Values Distribution in Dataframe



```
In [301]: 1 nan_values = df['major'].isnull().sum()  
2 print(f"Number of NaN values in the 'major' column: {nan_values}")
```

```
Number of NaN values in the 'major' column: 11399
```

```
In [302]: 1 df['major'].value_counts()
```

```
Out[302]: Engineering      3902  
Business/Management    3217  
I.T                  2570  
Mathematics            2362  
Psychology             1945  
...  
Mining                 1  
usa                     1  
Ophthalmology           1  
Cabin Crew              1  
Virology                1  
Name: major, Length: 158, dtype: int64
```

```
In [303]: 1 df=df.replace([np.inf, -np.inf], np.nan)  
2 df=df.dropna()
```

```
In [304]: 1 columns_to_check = df.columns[84:85]  
2 plot_null_values(df, columns_to_check)
```

```
Null values in the specified columns:  
    major  
3     False  
4     False  
6     False  
7     False  
8     False  
...  
39768  False  
39771  False  
39772  False  
39773  False  
39774  False
```

```
[28351 rows x 1 columns]
```

Null Values Distribution in Dataframe





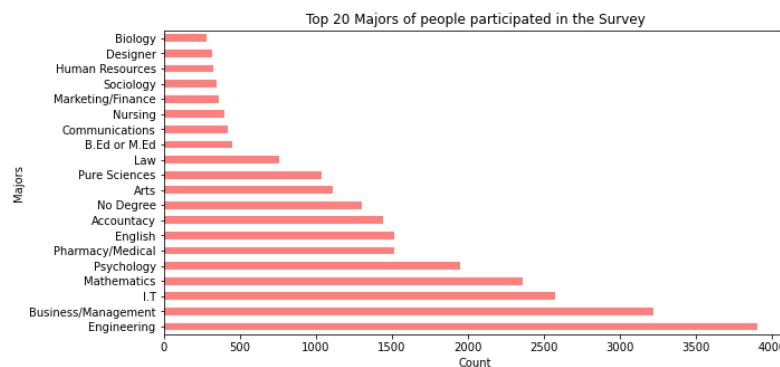
```
In [305]: 1 nan_values = df['major'].isnull().sum()
2 print(f"Number of NaN values in the 'major' column: {nan_values}")
Number of NaN values in the 'major' column: 0
```

In [306]: 1 df.shape

Out[306]: (28351, 86)

```
In [307]: 1 plt.figure(figsize=(10,5))
2 df['major'].value_counts()[:20].plot(kind='barh',color='r', alpha=0.5)
3 plt.ylabel('Majors')
4 plt.xlabel('Count')
5 plt.title('Top 20 Majors of people participated in the Survey')
```

```
Out[307]: Text(0.5, 1.0, 'Top 20 Majors of people participated in the Survey')
```



```
In [308]: 1 duplicate_rows = df.duplicated()  
2 print("Duplicate Rows except first occurrence:")  
3 duplicate_rows
```

Duplicate Rows except first occurrence:

```
Out[308]: 3      False
          4      False
          6      False
          7      False
          8      False
          ..
          39768     False
          39771     False
          39772     False
          39773     False
          39774     False
Length: 28351
```

In [309]: 1 df

Out[309]:

| Q1A   | Q2A | Q3A | Q4A | Q5A | Q6A | Q7A | Q8A | Q9A | Q10A | Q11A | Q12A | Q13A | Q14A | Q15A | Q16A | Q17A | Q18A | Q19A | Q20A | Q21A | Q22A | Q23A |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 3     | 2   | 3   | 2   | 1   | 3   | 3   | 4   | 2   | 3    | 3    | 2    | 1    | 1    | 4    | 2    | 2    | 3    | 1    | 1    | 2    | 1    | 1    |
| 4     | 2   | 2   | 3   | 4   | 4   | 2   | 4   | 4   | 4    | 3    | 2    | 4    | 4    | 4    | 4    | 3    | 4    | 4    | 4    | 4    | 3    | 3    |
| 6     | 1   | 1   | 2   | 3   | 4   | 1   | 3   | 3   | 3    | 4    | 1    | 2    | 2    | 2    | 3    | 4    | 2    | 2    | 1    | 3    | 4    | 1    |
| 7     | 1   | 1   | 1   | 1   | 1   | 3   | 2   | 2   | 1    | 1    | 1    | 2    | 1    | 1    | 2    | 2    | 1    | 1    | 2    | 1    | 1    | 1    |
| 8     | 4   | 4   | 3   | 4   | 3   | 4   | 4   | 4   | 4    | 3    | 4    | 4    | 4    | 3    | 4    | 4    | 3    | 4    | 4    | 4    | 3    | 4    |
| ...   | ... | ... | ... | ... | ... | ... | ... | ... | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  |
| 39768 | 2   | 1   | 1   | 1   | 2   | 2   | 1   | 2   | 1    | 1    | 2    | 2    | 2    | 2    | 1    | 2    | 1    | 2    | 2    | 1    | 1    | 2    |
| 39771 | 3   | 4   | 3   | 4   | 3   | 4   | 4   | 4   | 3    | 4    | 4    | 4    | 4    | 4    | 4    | 3    | 3    | 4    | 4    | 4    | 4    | 3    |
| 39772 | 2   | 1   | 2   | 1   | 1   | 1   | 1   | 2   | 1    | 1    | 1    | 2    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    |
| 39773 | 3   | 1   | 2   | 2   | 3   | 3   | 3   | 4   | 3    | 1    | 3    | 3    | 4    | 4    | 1    | 1    | 2    | 3    | 1    | 2    | 2    | 4    |
| 39774 | 2   | 1   | 2   | 1   | 4   | 2   | 1   | 1   | 1    | 1    | 3    | 1    | 3    | 1    | 1    | 4    | 4    | 3    | 4    | 2    | 1    | 1    |

28351 rows x 86 columns

```
In [310]: df = df.reset_index(drop=True)
```

In [244]:

Out[311]:

| Q1A | Q2A | Q3A | Q4A | Q5A | Q6A | Q7A | Q8A | Q9A | Q10A | Q11A | Q12A | Q13A | Q14A | Q15A | Q16A | Q17A | Q18A | Q19A | Q20A | Q21A | Q22A | Q23A |   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|------|---|
| 0   | 2   | 3   | 2   | 1   | 3   | 3   | 4   | 2   | 3    | 3    | 2    | 1    | 1    | 4    | 2    | 2    | 3    | 1    | 1    | 2    | 1    | 1    | 1 |
| 1   | 2   | 2   | 3   | 4   | 4   | 2   | 4   | 4   | 4    | 3    | 2    | 4    | 4    | 4    | 4    | 3    | 4    | 4    | 4    | 3    | 3    | 4    |   |

|       |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |   |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|
| 4     | 1   | 1   | 2   | 3   | 4   | 1   | 3   | 3   | 3   | 4   | 1   | 2   | 2   | 2   | 3   | 4   | 2   | 2   | 1   | 3   | 4   | 1   | 3   |   |
| 3     | 1   | 1   | 1   | 1   | 3   | 2   | 2   | 1   | 1   | 1   | 2   | 1   | 1   | 2   | 2   | 1   | 1   | 2   | 1   | 2   | 1   | 1   | 1   |   |
| 4     | 4   | 4   | 3   | 4   | 3   | 4   | 4   | 4   | 4   | 3   | 4   | 4   | 4   | 4   | 3   | 4   | 4   | 4   | 3   | 4   | 4   | 3   | 4   |   |
| ...   | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |   |
| 28346 | 2   | 1   | 1   | 1   | 2   | 2   | 1   | 2   | 1   | 1   | 2   | 2   | 2   | 2   | 1   | 2   | 2   | 1   | 2   | 2   | 1   | 1   | 2   | 1 |
| 28347 | 3   | 4   | 3   | 4   | 3   | 4   | 4   | 4   | 3   | 4   | 4   | 4   | 4   | 4   | 4   | 4   | 4   | 4   | 3   | 3   | 4   | 4   | 4   | 3 |
| 28348 | 2   | 1   | 2   | 1   | 1   | 1   | 1   | 2   | 1   | 1   | 1   | 2   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1 |
| 28349 | 3   | 1   | 2   | 2   | 3   | 3   | 3   | 4   | 3   | 1   | 3   | 3   | 4   | 4   | 4   | 1   | 1   | 2   | 3   | 1   | 2   | 2   | 4   | 1 |
| 28350 | 2   | 1   | 2   | 1   | 4   | 2   | 1   | 1   | 1   | 1   | 3   | 1   | 1   | 4   | 4   | 3   | 4   | 3   | 4   | 2   | 1   | 1   | 1   | 1 |

28351 rows × 86 columns



## (4) Machine Learning

In [312]: 1 dass = df.copy()

In [313]: 1 dass\_part2=df.iloc[:,42:]  
2 dass\_part2

|       | country | source | TIP11 | TIP12 | TIP13 | TIP14 | TIP15 | TIP16 | TIP17 | TIP18 | TIP19 | TIP10 | VCL1 | VCL2 | VCL3 | VCL4 | VCL5 | VCL6 | VCL7 | VCL8 | VCL9 | VCL10 |
|-------|---------|--------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|------|------|------|------|------|------|------|------|------|-------|
| 0     | US      | 2      | 1     | 1     | 7     | 4     | 6     | 4     | 6     | 1     | 6     | 1     | 1    | 0    | 0    | 1    | 1    | 0    | 0    | 0    | 0    |       |
| 1     | MY      | 2      | 2     | 5     | 3     | 6     | 5     | 5     | 5     | 6     | 3     | 3     | 1    | 1    | 0    | 1    | 1    | 0    | 0    | 1    | 0    |       |
| 2     | MX      | 2      | 2     | 5     | 6     | 5     | 3     | 2     | 6     | 3     | 5     | 5     | 1    | 1    | 1    | 1    | 1    | 0    | 1    | 0    | 0    |       |
| 3     | GB      | 2      | 7     | 6     | 4     | 5     | 3     | 2     | 6     | 3     | 5     | 2     | 1    | 1    | 0    | 1    | 1    | 1    | 0    | 1    | 0    |       |
| 4     | US      | 0      | 1     | 4     | 5     | 7     | 5     | 7     | 6     | 7     | 1     | 4     | 1    | 1    | 0    | 1    | 1    | 1    | 0    | 0    | 0    |       |
| ...   | ...     | ...    | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  |       |
| 28346 | US      | 0      | 6     | 5     | 6     | 7     | 6     | 4     | 6     | 5     | 5     | 3     | 1    | 1    | 0    | 1    | 1    | 0    | 1    | 1    | 0    |       |
| 28347 | MY      | 1      | 4     | 5     | 5     | 7     | 4     | 6     | 4     | 7     | 4     | 4     | 1    | 1    | 0    | 1    | 1    | 0    | 0    | 0    | 0    |       |
| 28348 | US      | 0      | 6     | 6     | 7     | 5     | 6     | 3     | 6     | 1     | 5     | 4     | 0    | 1    | 0    | 1    | 1    | 0    | 0    | 0    | 0    |       |
| 28349 | US      | 2      | 1     | 6     | 5     | 7     | 3     | 5     | 3     | 5     | 3     | 4     | 1    | 1    | 1    | 1    | 1    | 0    | 0    | 1    | 0    |       |
| 28350 | MY      | 1      | 6     | 2     | 3     | 5     | 6     | 3     | 5     | 5     | 1     | 2     | 1    | 0    | 0    | 1    | 1    | 0    | 0    | 0    | 0    |       |

28351 rows × 44 columns



In [314]: 1 Qs=dass.filter(regex='Q\d{1,2}A')  
2 Qs

|       | Q1A | Q2A | Q3A | Q4A | Q5A | Q6A | Q7A | Q8A | Q9A | Q10A | Q11A | Q12A | Q13A | Q14A | Q15A | Q16A | Q17A | Q18A | Q19A | Q20A | Q21A | Q22A | Q23A |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 0     | 2   | 3   | 2   | 1   | 3   | 3   | 4   | 2   | 3   | 3    | 2    | 1    | 1    | 4    | 2    | 2    | 3    | 1    | 1    | 2    | 1    | 1    | 1    |
| 1     | 2   | 2   | 3   | 4   | 4   | 2   | 4   | 4   | 4   | 3    | 2    | 4    | 4    | 4    | 4    | 3    | 4    | 4    | 4    | 3    | 3    | 4    |      |
| 2     | 1   | 1   | 2   | 3   | 4   | 1   | 3   | 3   | 3   | 4    | 1    | 2    | 2    | 2    | 3    | 4    | 2    | 2    | 1    | 3    | 4    | 1    | 3    |
| 3     | 1   | 1   | 1   | 1   | 3   | 2   | 2   | 1   | 1   | 1    | 2    | 1    | 1    | 2    | 2    | 1    | 1    | 2    | 1    | 2    | 1    | 1    | 1    |
| 4     | 4   | 4   | 3   | 4   | 3   | 4   | 4   | 4   | 4   | 3    | 4    | 4    | 4    | 4    | 3    | 4    | 4    | 4    | 4    | 3    | 4    | 4    | 4    |
| ...   | ... | ... | ... | ... | ... | ... | ... | ... | ... | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  |
| 28346 | 2   | 1   | 1   | 1   | 2   | 2   | 1   | 2   | 1   | 1    | 2    | 2    | 2    | 2    | 1    | 2    | 2    | 1    | 2    | 1    | 1    | 2    | 1    |
| 28347 | 3   | 4   | 3   | 4   | 3   | 4   | 4   | 4   | 3   | 4    | 4    | 4    | 4    | 4    | 4    | 3    | 3    | 4    | 4    | 4    | 4    | 4    | 3    |
| 28348 | 2   | 1   | 2   | 1   | 1   | 1   | 1   | 2   | 1   | 1    | 1    | 2    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    |
| 28349 | 3   | 1   | 2   | 2   | 3   | 3   | 3   | 4   | 3   | 1    | 3    | 3    | 4    | 4    | 4    | 1    | 1    | 2    | 3    | 1    | 2    | 2    | 4    |
| 28350 | 2   | 1   | 2   | 1   | 4   | 2   | 1   | 1   | 1   | 1    | 3    | 1    | 3    | 1    | 4    | 4    | 3    | 4    | 2    | 1    | 1    | 1    | 1    |

See This Photo for How This Looked 😊 !!!

0 Did not apply to me at all

1 Applied to me to some degree, or some of the time

2 Applied to me to a considerable degree, or a good part of time

3 Applied to me very much, or most of the time

In [315]: 1 def sub(Qs):  
2 return Qs.subtract(1,axis=1)

In [316]: 1 Qs = sub(Qs)

In [317]: 1 Q5

|   | Q1A | Q2A | Q3A | Q4A | Q5A | Q6A | Q7A | Q8A | Q9A | Q10A | Q11A | Q12A | Q13A | Q14A | Q15A | Q16A | Q17A | Q18A | Q19A | Q20A | Q21A | Q22A | Q23A |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 0 | 1   | 2   | 1   | 0   | 2   | 2   | 3   | 1   | 2   | 2    | 1    | 0    | 0    | 3    | 1    | 1    | 2    | 0    | 0    | 1    | 0    | 0    | 0    |
| 1 | 1   | 1   | 2   | 3   | 3   | 1   | 3   | 3   | 3   | 2    | 1    | 3    | 3    | 3    | 2    | 3    | 3    | 3    | 3    | 2    | 2    | 3    | 3    |
| 2 | 0   | 0   | 1   | 2   | 3   | 0   | 2   | 2   | 2   | 3    | 0    | 1    | 1    | 1    | 2    | 3    | 1    | 1    | 0    | 2    | 3    | 0    | 2    |

|       |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 3     | 0   | 0   | 0   | 0   | 2   | 1   | 1   | 0   | 0   | 0   | 1   | 0   | 0   | 1   | 1   | 0   | 0   | 1   | 0   | 1   | 0   | 0   | 0   |
| 4     | 3   | 3   | 2   | 3   | 2   | 3   | 3   | 3   | 3   | 2   | 3   | 3   | 3   | 2   | 3   | 3   | 2   | 3   | 3   | 2   | 3   | 3   | 3   |
| ...   | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 28346 | 1   | 0   | 0   | 0   | 1   | 1   | 0   | 1   | 0   | 0   | 1   | 1   | 1   | 1   | 0   | 1   | 0   | 1   | 1   | 0   | 0   | 1   | 0   |
| 28347 | 2   | 3   | 2   | 3   | 2   | 3   | 3   | 3   | 3   | 2   | 3   | 3   | 3   | 3   | 3   | 2   | 2   | 3   | 3   | 3   | 3   | 3   | 2   |
| 28348 | 1   | 0   | 1   | 0   | 0   | 0   | 0   | 0   | 1   | 0   | 0   | 0   | 1   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 28349 | 2   | 0   | 1   | 1   | 2   | 2   | 2   | 3   | 2   | 0   | 2   | 2   | 3   | 3   | 0   | 0   | 0   | 1   | 2   | 0   | 1   | 1   | 3   |
| 28350 | 1   | 0   | 1   | 0   | 3   | 1   | 0   | 0   | 0   | 0   | 2   | 0   | 2   | 0   | 0   | 3   | 3   | 2   | 3   | 2   | 3   | 1   | 0   |

28351 rows × 42 columns

|       |       |       |       |
|-------|-------|-------|-------|
| 1. S  | 2. A  | 3. D  | 4. A  |
| 5. D  | 6. S  | 7. A  | 8. S  |
| 9. A  | 10. D | 11. S | 12. S |
| 13. D | 14. S | 15. A | 16. D |
| 17. D | 18. S | 19. A | 20. A |
| 21. D | 22. S | 23. A | 24. D |
| 25. A | 26. D | 27. S | 28. A |
| 29. S | 30. A | 31. D | 32. S |
| 33. S | 34. D | 35. S | 36. A |
| 37. D | 38. D | 39. S | 40. A |
| 41. A | 42. D |       |       |

Scores of Depression, Anxiety and Stress are calculated by summing the scores for the relevant items.

The depression scale items are 3, 5, 10, 13, 16, 17, 21, 24, 26, 31, 34, 37, 38, 42.

The anxiety scale items are 2, 4, 7, 9, 15, 19, 20, 23, 25, 28, 30, 36, 40, 41.

The stress scale items are 1, 6, 8, 11, 12, 14, 18, 22, 27, 29, 32, 33, 35, 39.

|                  | Depression | Anxiety | Stress |
|------------------|------------|---------|--------|
| Normal           | 0-9        | 0-7     | 0-14   |
| Mild             | 10-13      | 8-9     | 15-18  |
| Moderate         | 14-20      | 10-14   | 19-25  |
| Severe           | 21-27      | 15-19   | 26-33  |
| Extremely Severe | 28+        | 20+     | 34+    |

```
In [318]: 1 DASS_keys = {'Depression': [3, 5, 10, 13, 16, 17, 21, 24, 26, 31, 34, 37, 38, 42],
 2           'Anxiety': [2, 4, 7, 9, 15, 19, 20, 23, 25, 28, 30, 36, 40, 41],
 3           'Stress': [1, 6, 8, 11, 12, 14, 18, 22, 27, 29, 32, 33, 35, 39]}
```

```
In [319]: 1 depression = []
 2 anxiety = []
 3 stress = []
```

```
In [320]: 1 for i in DASS_keys["Depression"]:
 2     depression.append('Q'+str(i)+'A')
```

```
In [321]: 1 for i in DASS_keys["Anxiety"]:
 2     anxiety.append('Q'+str(i)+'A')
```

```
In [322]: 1 for i in DASS_keys["Stress"]:
 2     stress.append('Q'+str(i)+'A')
```

```
In [323]: 1 print(depression)
 2 print(anxiety)
 3 print(stress)

['Q3A', 'Q5A', 'Q10A', 'Q13A', 'Q16A', 'Q17A', 'Q21A', 'Q24A', 'Q26A', 'Q31A', 'Q34A', 'Q37A', 'Q38A', 'Q42A']
['Q2A', 'Q4A', 'Q7A', 'Q9A', 'Q15A', 'Q19A', 'Q20A', 'Q23A', 'Q25A', 'Q28A', 'Q30A', 'Q36A', 'Q40A', 'Q41A']
['Q1A', 'Q6A', 'Q8A', 'Q11A', 'Q12A', 'Q14A', 'Q18A', 'Q22A', 'Q27A', 'Q29A', 'Q32A', 'Q33A', 'Q35A', 'Q39A']
```

```
In [324]: 1 depression= Qs.filter(depression)
 2 anxiety = Qs.filter(anxiety)
 3 stress = Qs.filter(stress)
```

```
In [325]: 1 depression
```

Out[325]:

|       | Q3A | Q5A | Q10A | Q13A | Q16A | Q17A | Q21A | Q24A | Q26A | Q31A | Q34A | Q37A | Q38A | Q42A |
|-------|-----|-----|------|------|------|------|------|------|------|------|------|------|------|------|
| 0     | 1   | 2   | 2    | 0    | 1    | 2    | 0    | 0    | 1    | 2    | 3    | 1    | 0    | 1    |
| 1     | 2   | 3   | 2    | 3    | 2    | 3    | 2    | 1    | 3    | 2    | 3    | 2    | 2    | 2    |
| 2     | 1   | 3   | 3    | 1    | 3    | 1    | 3    | 2    | 1    | 3    | 2    | 3    | 3    | 2    |
| 3     | 0   | 2   | 0    | 0    | 0    | 0    | 0    | 0    | 1    | 0    | 0    | 0    | 0    | 1    |
| 4     | 2   | 2   | 2    | 3    | 3    | 2    | 2    | 2    | 3    | 3    | 3    | 3    | 2    | 3    |
| ...   | ... | ... | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  |
| 28346 | 0   | 1   | 0    | 1    | 1    | 0    | 0    | 1    | 1    | 0    | 0    | 0    | 0    | 1    |
| 28347 | 2   | 2   | 3    | 3    | 3    | 2    | 3    | 3    | 3    | 3    | 3    | 2    | 3    | 3    |
| 28348 | 1   | 0   | 0    | 1    | 0    | 0    | 0    | 1    | 0    | 1    | 0    | 0    | 0    | 0    |
| 28349 | 1   | 2   | 0    | 3    | 0    | 1    | 1    | 1    | 2    | 1    | 1    | 0    | 1    | 2    |
| 28350 | 1   | 3   | 0    | 2    | 3    | 3    | 1    | 1    | 1    | 1    | 2    | 1    | 3    | 3    |

28351 rows × 14 columns

In [326]: 1 anxiety

Out[326]:

|       | Q2A | Q4A | Q7A | Q9A | Q15A | Q19A | Q20A | Q23A | Q25A | Q28A | Q30A | Q36A | Q40A | Q41A |
|-------|-----|-----|-----|-----|------|------|------|------|------|------|------|------|------|------|
| 0     | 2   | 0   | 3   | 2   | 1    | 0    | 1    | 0    | 0    | 0    | 2    | 0    | 3    | 3    |
| 1     | 1   | 3   | 3   | 3   | 3    | 3    | 3    | 3    | 3    | 3    | 3    | 3    | 3    | 3    |
| 2     | 0   | 2   | 2   | 2   | 2    | 0    | 2    | 2    | 1    | 1    | 2    | 1    | 1    | 1    |
| 3     | 0   | 0   | 1   | 0   | 1    | 0    | 1    | 0    | 1    | 0    | 0    | 0    | 0    | 0    |
| 4     | 3   | 3   | 3   | 3   | 3    | 3    | 3    | 3    | 1    | 3    | 2    | 3    | 3    | 3    |
| ...   | ... | ... | ... | ... | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  |
| 28346 | 0   | 0   | 0   | 0   | 0    | 1    | 0    | 0    | 0    | 0    | 1    | 0    | 0    | 0    |
| 28347 | 3   | 3   | 3   | 2   | 3    | 3    | 3    | 2    | 2    | 2    | 2    | 2    | 2    | 2    |
| 28348 | 0   | 0   | 0   | 1   | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |
| 28349 | 0   | 1   | 2   | 2   | 0    | 0    | 1    | 0    | 0    | 2    | 0    | 1    | 1    | 3    |
| 28350 | 0   | 0   | 0   | 0   | 0    | 3    | 2    | 0    | 1    | 1    | 0    | 1    | 2    | 0    |

28351 rows × 14 columns

In [327]: 1 stress

Out[327]:

|       | Q1A | Q6A | Q8A | Q11A | Q12A | Q14A | Q18A | Q22A | Q27A | Q29A | Q32A | Q33A | Q35A | Q39A |
|-------|-----|-----|-----|------|------|------|------|------|------|------|------|------|------|------|
| 0     | 1   | 2   | 1   | 1    | 0    | 3    | 0    | 0    | 3    | 2    | 0    | 1    | 0    | 2    |
| 1     | 1   | 1   | 3   | 1    | 3    | 3    | 3    | 2    | 1    | 1    | 3    | 3    | 2    | 2    |
| 2     | 0   | 0   | 2   | 0    | 1    | 1    | 1    | 0    | 0    | 1    | 1    | 3    | 1    | 3    |
| 3     | 0   | 1   | 0   | 1    | 0    | 1    | 1    | 0    | 1    | 0    | 0    | 0    | 0    | 1    |
| 4     | 3   | 3   | 3   | 3    | 3    | 2    | 3    | 3    | 1    | 3    | 1    | 3    | 1    | 1    |
| ...   | ... | ... | ... | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  |
| 28346 | 1   | 1   | 1   | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    |
| 28347 | 2   | 3   | 3   | 3    | 3    | 3    | 3    | 2    | 3    | 2    | 2    | 2    | 2    | 2    |
| 28348 | 1   | 0   | 0   | 0    | 0    | 0    | 0    | 0    | 0    | 1    | 1    | 0    | 1    | 0    |
| 28349 | 2   | 2   | 3   | 2    | 2    | 3    | 2    | 3    | 3    | 2    | 2    | 3    | 2    | 2    |
| 28350 | 1   | 1   | 0   | 2    | 0    | 0    | 2    | 1    | 2    | 2    | 1    | 1    | 1    | 2    |

28351 rows × 14 columns

In [328]: 1 def scores(source):  
2 col=list(source)  
3 source['Total\_Count']=source[col].sum(axis=1)  
4 return source

In [329]: 1 depression=scores(depression)  
2 anxiety=scores(anxiety)  
3 stress=scores(stress)

In [330]: 1 depression.head()

Out[330]:

|   | Q3A | Q5A | Q10A | Q13A | Q16A | Q17A | Q21A | Q24A | Q26A | Q31A | Q34A | Q37A | Q42A | Total_Count |    |
|---|-----|-----|------|------|------|------|------|------|------|------|------|------|------|-------------|----|
| 0 | 1   | 2   | 2    | 0    | 1    | 2    | 0    | 0    | 1    | 2    | 3    | 1    | 0    | 1           | 16 |
| 1 | 2   | 3   | 2    | 3    | 2    | 3    | 2    | 1    | 3    | 2    | 3    | 2    | 2    | 2           | 32 |
| 2 | 1   | 3   | 3    | 1    | 3    | 1    | 3    | 2    | 1    | 3    | 2    | 3    | 3    | 2           | 31 |
| 3 | 0   | 2   | 0    | 0    | 0    | 0    | 0    | 0    | 1    | 0    | 0    | 0    | 0    | 1           | 4  |
| 4 | 2   | 2   | 2    | 3    | 3    | 2    | 2    | 2    | 3    | 3    | 3    | 3    | 2    | 3           | 35 |

In [331]: 1 anxiety.head()

Out[331]:

|   | Q2A | Q4A | Q7A | Q9A | Q15A | Q19A | Q20A | Q23A | Q25A | Q28A | Q30A | Q36A | Q40A | Q41A | Total_Count |
|---|-----|-----|-----|-----|------|------|------|------|------|------|------|------|------|------|-------------|
| 0 | 2   | 0   | 3   | 2   | 1    | 0    | 1    | 0    | 0    | 0    | 2    | 0    | 3    | 3    | 17          |
| 1 | 1   | 3   | 3   | 3   | 3    | 3    | 3    | 3    | 3    | 3    | 3    | 3    | 3    | 3    | 40          |
| 2 | 0   | 2   | 2   | 2   | 2    | 0    | 2    | 2    | 1    | 1    | 2    | 1    | 1    | 1    | 19          |
| 3 | 0   | 0   | 1   | 0   | 1    | 0    | 1    | 0    | 1    | 0    | 0    | 0    | 0    | 0    | 4           |
| 4 | 3   | 3   | 3   | 3   | 3    | 3    | 3    | 3    | 1    | 3    | 2    | 3    | 3    | 3    | 39          |

In [332]: 1 stress.head()

Out[332]:

|   | Q1A | Q6A | Q8A | Q11A | Q12A | Q14A | Q18A | Q22A | Q27A | Q29A | Q32A | Q33A | Q35A | Q39A | Total_Count |
|---|-----|-----|-----|------|------|------|------|------|------|------|------|------|------|------|-------------|
| 0 | 1   | 2   | 1   | 1    | 0    | 3    | 0    | 0    | 3    | 2    | 0    | 1    | 0    | 2    | 16          |
| 1 | 1   | 1   | 3   | 1    | 3    | 3    | 3    | 2    | 1    | 1    | 3    | 3    | 2    | 2    | 29          |
| 2 | 0   | 0   | 2   | 0    | 1    | 1    | 1    | 0    | 0    | 1    | 1    | 3    | 1    | 3    | 14          |
| 3 | 0   | 1   | 0   | 1    | 0    | 1    | 1    | 0    | 1    | 0    | 0    | 0    | 0    | 1    | 6           |
| 4 | 3   | 3   | 3   | 3    | 3    | 2    | 3    | 3    | 1    | 3    | 2    | 3    | 3    | 1    | 33          |

in Depression Set

```
In [333]: 1 Depression=pd.merge(depression,dass_part2,how='left',left_index=True,right_index=True)
2 Depression.head()
```

Out[333]:

|   | Q3A | Q5A | Q10A | Q13A | Q16A | Q17A | Q21A | Q24A | Q26A | Q31A | Q34A | Q37A | Q38A | Q42A | Total_Count | country | source | TIP11 | TIP12 | TIP13 | TIP14 |
|---|-----|-----|------|------|------|------|------|------|------|------|------|------|------|------|-------------|---------|--------|-------|-------|-------|-------|
| 0 | 1   | 2   | 2    | 0    | 1    | 2    | 0    | 0    | 1    | 2    | 3    | 1    | 0    | 1    | 16          | US      | 2      | 1     | 1     | 7     | 4     |
| 1 | 2   | 3   | 2    | 3    | 2    | 3    | 2    | 1    | 3    | 2    | 3    | 2    | 2    | 2    | 32          | MY      | 2      | 2     | 5     | 3     | 6     |
| 2 | 1   | 3   | 3    | 1    | 3    | 1    | 3    | 2    | 1    | 3    | 2    | 3    | 3    | 2    | 31          | MX      | 2      | 2     | 5     | 6     | 5     |
| 3 | 0   | 2   | 0    | 0    | 0    | 0    | 0    | 0    | 1    | 0    | 0    | 0    | 0    | 1    | 4           | GB      | 2      | 7     | 6     | 4     | 5     |
| 4 | 2   | 2   | 2    | 3    | 3    | 2    | 2    | 2    | 3    | 3    | 3    | 3    | 2    | 3    | 35          | US      | 0      | 1     | 4     | 5     | 7     |

```
In [334]: 1 def predict_depression(x):
2     if x<=9:
3         return 'Normal'
4     if 10<=x<=13:
5         return 'Mild'
6     if 14<=x<=20:
7         return 'Moderate'
8     if 21<=x<=27:
9         return 'Severe'
10    if x>28:
11        return 'Extremely Severe'
```

```
In [335]: 1 Depression['predict']=Depression['Total_Count'].apply(predict_depression)
```

```
In [336]: 1 Depression
```

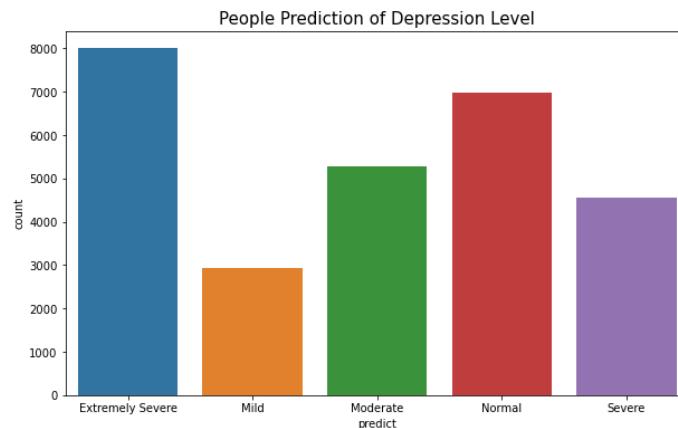
Out[336]:

|       | Q3A | Q5A | Q10A | Q13A | Q16A | Q17A | Q21A | Q24A | Q26A | Q31A | Q34A | Q37A | Q38A | Q42A | Total_Count | country | source | TIP11 | TIP12 | TIP13 | TIP14 |   |
|-------|-----|-----|------|------|------|------|------|------|------|------|------|------|------|------|-------------|---------|--------|-------|-------|-------|-------|---|
| 0     | 1   | 2   | 2    | 0    | 1    | 2    | 0    | 0    | 1    | 2    | 3    | 1    | 0    | 1    | 16          | US      | 2      | 1     | 1     | 7     | 4     |   |
| 1     | 2   | 3   | 2    | 3    | 2    | 3    | 2    | 1    | 3    | 2    | 3    | 2    | 2    | 2    | 32          | MY      | 2      | 2     | 5     | 3     | 6     |   |
| 2     | 1   | 3   | 3    | 1    | 3    | 1    | 3    | 2    | 1    | 3    | 2    | 3    | 3    | 2    | 31          | MX      | 2      | 2     | 5     | 6     | 5     |   |
| 3     | 0   | 2   | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 1    | 0    | 0    | 0    | 0    | 1           | 4       | GB     | 2     | 7     | 6     | 4     | 5 |
| 4     | 2   | 2   | 2    | 3    | 3    | 2    | 2    | 2    | 3    | 3    | 3    | 3    | 2    | 3    | 35          | US      | 0      | 1     | 4     | 5     | 7     |   |
| ...   | ... | ... | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...         | ...     | ...    | ...   | ...   | ...   | ...   |   |
| 28346 | 0   | 1   | 0    | 1    | 1    | 0    | 0    | 1    | 1    | 0    | 0    | 0    | 0    | 1    | 6           | US      | 0      | 6     | 5     | 6     | 7     |   |
| 28347 | 2   | 2   | 3    | 3    | 3    | 2    | 3    | 3    | 3    | 3    | 3    | 3    | 2    | 3    | 3           | 38      | MY     | 1     | 4     | 5     | 5     | 7 |
| 28348 | 1   | 0   | 0    | 1    | 0    | 0    | 0    | 1    | 0    | 1    | 0    | 0    | 0    | 0    | 4           | US      | 0      | 6     | 6     | 7     | 5     |   |
| 28349 | 1   | 2   | 0    | 3    | 0    | 1    | 1    | 1    | 2    | 1    | 1    | 0    | 1    | 2    | 16          | US      | 2      | 1     | 6     | 5     | 7     |   |
| 28350 | 1   | 3   | 0    | 2    | 3    | 3    | 3    | 1    | 1    | 1    | 1    | 2    | 1    | 3    | 3           | 27      | MY     | 1     | 6     | 2     | 3     | 5 |

28351 rows × 60 columns

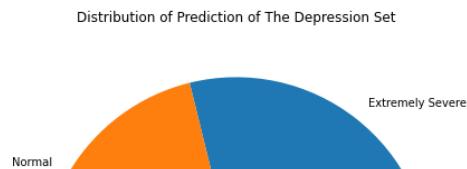
```
In [337]: 1 plt.figure(figsize=(10,6))
2 sns.countplot(data=Depression, x=Depression.sort_values('predict').predict)
3 plt.title('People Prediction of Depression Level', fontsize=15)
```

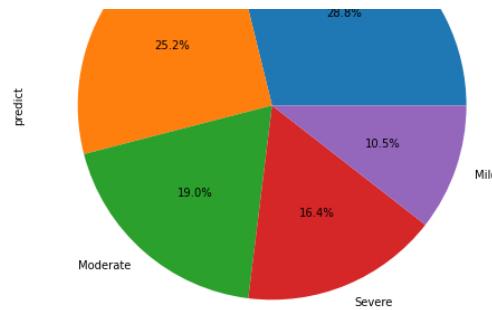
```
Out[337]: Text(0.5, 1.0, 'People Prediction of Depression Level')
```



```
In [338]: 1 value_counts = Depression['predict'].value_counts().sort_index()
```

```
In [339]: 1 plt.figure(figsize=(8, 8))
2 Depression['predict'].value_counts().plot.pie(autopct='%1.1f%%', startangle=0)
3 plt.title('Distribution of Prediction of The Depression Set')
4 plt.show()
```





## in Anxiety Set

```
In [340]: 1 Anxiety=pd.merge(anxiety, dass_part2,how='left',left_index=True,right_index=True)
2 Anxiety.head()
```

Out[340]:

|   | Q2A | Q4A | Q7A | Q9A | Q15A | Q19A | Q20A | Q23A | Q25A | Q28A | Q30A | Q36A | Q40A | Q41A | Total_Count | country | source | TIP1 | TIP12 | TIP13 | TIP14 | TIP15 |
|---|-----|-----|-----|-----|------|------|------|------|------|------|------|------|------|------|-------------|---------|--------|------|-------|-------|-------|-------|
| 0 | 2   | 0   | 3   | 2   | 1    | 0    | 1    | 0    | 0    | 0    | 2    | 0    | 3    | 3    | 17          | US      | 2      | 1    | 1     | 7     | 4     | 6     |
| 1 | 1   | 3   | 3   | 3   | 3    | 3    | 3    | 3    | 3    | 3    | 3    | 3    | 3    | 3    | 40          | MY      | 2      | 2    | 5     | 3     | 6     | 5     |
| 2 | 0   | 2   | 2   | 2   | 2    | 0    | 2    | 2    | 1    | 1    | 2    | 1    | 1    | 1    | 19          | MX      | 2      | 2    | 5     | 6     | 5     | 3     |
| 3 | 0   | 0   | 1   | 0   | 1    | 0    | 1    | 0    | 1    | 0    | 0    | 0    | 0    | 0    | 4           | GB      | 2      | 7    | 6     | 4     | 5     | 3     |
| 4 | 3   | 3   | 3   | 3   | 3    | 3    | 3    | 3    | 1    | 3    | 2    | 3    | 3    | 3    | 39          | US      | 0      | 1    | 4     | 5     | 7     | 5     |

```
In [341]: 1 def predict_anxiety(x):
2     if x<=7:
3         return 'Normal'
4     if 8<x<=9:
5         return 'Mild'
6     if 10<x<=14:
7         return 'Moderate'
8     if 15<x<=19:
9         return 'Severe'
10    if x>19:
11        return 'Extremely Severe'
```

```
In [342]: 1 Anxiety['predict']=Anxiety['Total_Count'].apply(predict_anxiety)
```

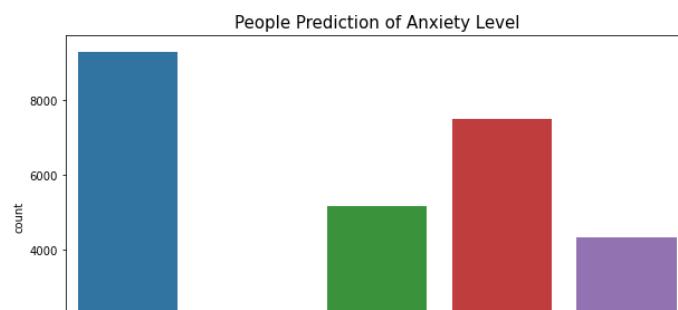
In [343]: 1 Anxiety

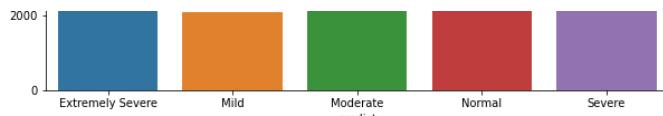
|       | Q2A | Q4A | Q7A | Q9A | Q15A | Q19A | Q20A | Q23A | Q25A | Q28A | Q30A | Q36A | Q40A | Q41A | Total_Count | country | source | TIP1 | TIP12 | TIP13 | TIP14 | T   |
|-------|-----|-----|-----|-----|------|------|------|------|------|------|------|------|------|------|-------------|---------|--------|------|-------|-------|-------|-----|
| 0     | 2   | 0   | 3   | 2   | 1    | 0    | 1    | 0    | 0    | 0    | 2    | 0    | 3    | 3    | 17          | US      | 2      | 1    | 1     | 7     | 4     | 6   |
| 1     | 1   | 3   | 3   | 3   | 3    | 3    | 3    | 3    | 3    | 3    | 3    | 3    | 3    | 3    | 40          | MY      | 2      | 2    | 5     | 3     | 6     | 5   |
| 2     | 0   | 2   | 2   | 2   | 2    | 0    | 2    | 2    | 1    | 1    | 2    | 1    | 1    | 1    | 19          | MX      | 2      | 2    | 5     | 6     | 5     | 3   |
| 3     | 0   | 0   | 1   | 0   | 1    | 0    | 1    | 0    | 1    | 0    | 0    | 0    | 0    | 0    | 4           | GB      | 2      | 7    | 6     | 4     | 5     | 3   |
| 4     | 3   | 3   | 3   | 3   | 3    | 3    | 3    | 3    | 1    | 3    | 2    | 3    | 3    | 3    | 39          | US      | 0      | 1    | 4     | 5     | 7     | 5   |
| ...   | ... | ... | ... | ... | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...         | ...     | ...    | ...  | ...   | ...   | ...   | ... |
| 28346 | 0   | 0   | 0   | 0   | 0    | 1    | 0    | 0    | 0    | 0    | 1    | 0    | 0    | 0    | 2           | US      | 0      | 6    | 5     | 6     | 7     |     |
| 28347 | 3   | 3   | 3   | 2   | 3    | 3    | 3    | 2    | 2    | 2    | 2    | 2    | 2    | 2    | 34          | MY      | 1      | 4    | 5     | 5     | 7     |     |
| 28348 | 0   | 0   | 0   | 1   | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 1           | US      | 0      | 6    | 6     | 7     | 5     |     |
| 28349 | 0   | 1   | 2   | 2   | 0    | 0    | 1    | 0    | 0    | 2    | 0    | 1    | 1    | 3    | 13          | US      | 2      | 1    | 6     | 5     | 7     |     |
| 28350 | 0   | 0   | 0   | 0   | 0    | 3    | 2    | 0    | 1    | 1    | 0    | 1    | 2    | 0    | 10          | MY      | 1      | 6    | 2     | 3     | 5     |     |

28351 rows × 60 columns

```
In [344]: 1 plt.figure(figsize=(10,6))
2 sns.countplot(data=Anxiety, x=Anxiety.sort_values('predict').predict)
3 plt.title('People Prediction of Anxiety Level', fontsize=15)
```

Out[344]: Text(0.5, 1.0, 'People Prediction of Anxiety Level')

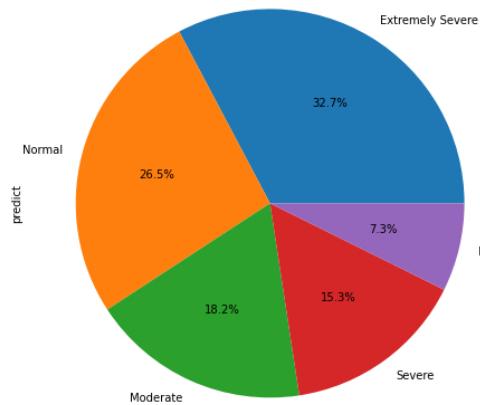




```
In [345]: 1 value_counts = Anxiety['predict'].value_counts().sort_index()
```

```
In [346]: 1 plt.figure(figsize=(8, 8))
2 Anxiety['predict'].value_counts().plot.pie(autopct='%1.1f%%', startangle=0)
3 plt.title('Distribution of Prediction of The Anxiety Set')
4 plt.show()
```

Distribution of Prediction of The Anxiety Set



## in Stress Set

```
In [347]: 1 Stress=pd.merge(stress,dass_part2,how='left',left_index=True,right_index=True)
2 Stress.head()
```

```
Out[347]:
```

|   | Q1A | Q6A | Q8A | Q11A | Q12A | Q14A | Q18A | Q22A | Q27A | Q29A | Q32A | Q33A | Q35A | Q39A | Total_Count | country | source | TIP11 | TIP12 | TIP13 | TIP14 | TIP15 |
|---|-----|-----|-----|------|------|------|------|------|------|------|------|------|------|------|-------------|---------|--------|-------|-------|-------|-------|-------|
| 0 | 1   | 2   | 1   | 1    | 0    | 3    | 0    | 0    | 3    | 2    | 0    | 1    | 0    | 2    | 16          | US      | 2      | 1     | 1     | 7     | 4     | 6     |
| 1 | 1   | 1   | 3   | 1    | 3    | 3    | 3    | 2    | 1    | 1    | 3    | 3    | 2    | 2    | 29          | MY      | 2      | 2     | 5     | 3     | 6     | 5     |
| 2 | 0   | 0   | 2   | 0    | 1    | 1    | 1    | 0    | 0    | 1    | 1    | 3    | 1    | 3    | 14          | MX      | 2      | 2     | 5     | 6     | 5     | 3     |
| 3 | 0   | 1   | 0   | 1    | 0    | 1    | 1    | 0    | 1    | 0    | 0    | 0    | 0    | 1    | 6           | GB      | 2      | 7     | 6     | 4     | 5     | 3     |
| 4 | 3   | 3   | 3   | 3    | 3    | 2    | 3    | 3    | 1    | 3    | 1    | 3    | 1    | 1    | 33          | US      | 0      | 1     | 4     | 5     | 7     | 5     |

```
In [348]: 1 def predict_stress(x):
2     if x<=14:
3         return 'Normal'
4     if 15<=x<=18:
5         return 'Mild'
6     if 19<=x<=25:
7         return 'Moderate'
8     if 26<=x<=33:
9         return 'Severe'
10    if x>=34:
11        return 'Extremely Severe'
```

```
In [349]: 1 Stress['predict']=Stress['Total_Count'].apply(predict_stress)
```

```
In [350]: 1 Stress
```

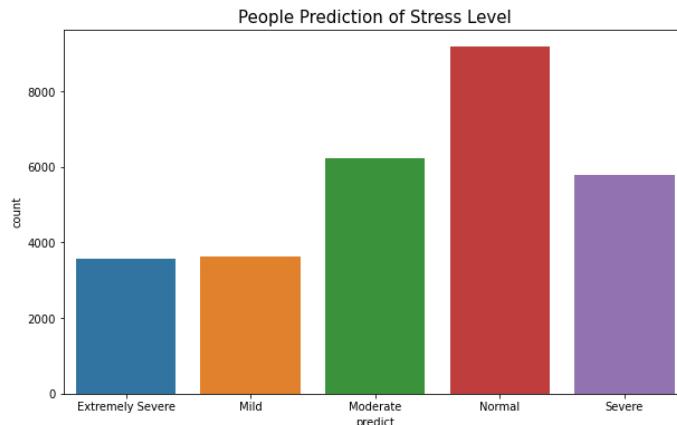
```
Out[350]:
```

|       | Q1A | Q6A | Q8A | Q11A | Q12A | Q14A | Q18A | Q22A | Q27A | Q29A | Q32A | Q33A | Q35A | Q39A | Total_Count | country | source | TIP11 | TIP12 | TIP13 | TIP14 | TIP15 |
|-------|-----|-----|-----|------|------|------|------|------|------|------|------|------|------|------|-------------|---------|--------|-------|-------|-------|-------|-------|
| 0     | 1   | 2   | 1   | 1    | 0    | 3    | 0    | 0    | 3    | 2    | 0    | 1    | 0    | 2    | 16          | US      | 2      | 1     | 1     | 7     | 4     | 6     |
| 1     | 1   | 1   | 3   | 1    | 3    | 3    | 3    | 2    | 1    | 1    | 3    | 3    | 2    | 2    | 29          | MY      | 2      | 2     | 5     | 3     | 6     | 5     |
| 2     | 0   | 0   | 2   | 0    | 1    | 1    | 1    | 0    | 0    | 1    | 1    | 3    | 1    | 3    | 14          | MX      | 2      | 2     | 5     | 6     | 5     | 5     |
| 3     | 0   | 1   | 0   | 1    | 0    | 1    | 1    | 1    | 0    | 1    | 0    | 0    | 0    | 1    | 6           | GB      | 2      | 7     | 6     | 4     | 5     | 5     |
| 4     | 3   | 3   | 3   | 3    | 3    | 2    | 3    | 3    | 1    | 3    | 1    | 3    | 1    | 1    | 33          | US      | 0      | 1     | 4     | 5     | 7     | 5     |
| ...   | ... | ... | ... | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...         | ...     | ...    | ...   | ...   | ...   | ...   | ...   |
| 28346 | 1   | 1   | 1   | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 14          | US      | 0      | 6     | 5     | 6     | 7     | 7     |
| 28347 | 2   | 3   | 3   | 3    | 3    | 3    | 2    | 3    | 2    | 2    | 2    | 2    | 2    | 2    | 34          | MY      | 1      | 4     | 5     | 5     | 7     | 7     |
| 28348 | 1   | 0   | 0   | 0    | 0    | 0    | 0    | 0    | 1    | 1    | 0    | 1    | 0    | 1    | 5           | US      | 0      | 6     | 6     | 7     | 5     | 5     |
| 28349 | 2   | 2   | 3   | 2    | 2    | 3    | 2    | 3    | 3    | 2    | 2    | 3    | 2    | 2    | 33          | US      | 2      | 1     | 6     | 5     | 7     | 7     |
| 28350 | 1   | 1   | 0   | 2    | 0    | 0    | 2    | 1    | 2    | 2    | 1    | 1    | 1    | 2    | 16          | MY      | 1      | 6     | 2     | 3     | 5     | 5     |

28351 rows × 60 columns

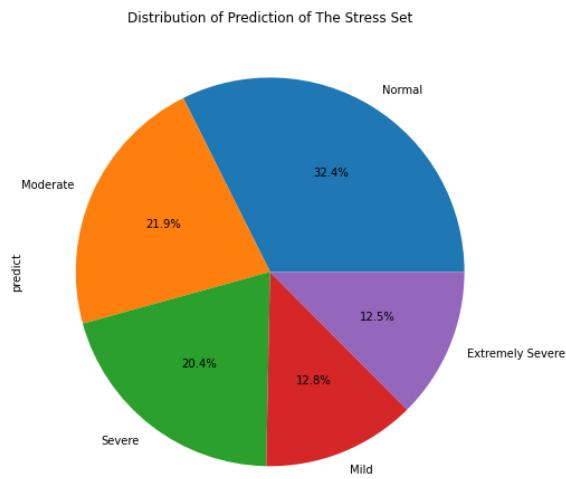
```
In [351]: 1 plt.figure(figsize=(10,6))
2 sns.countplot(data=Depression, x=Depression.sort_values('predict').predict)
3 plt.title('People Prediction of Stress Level', fontsize=15)
```

Out[351]: Text(0.5, 1.0, 'People Prediction of Stress Level')



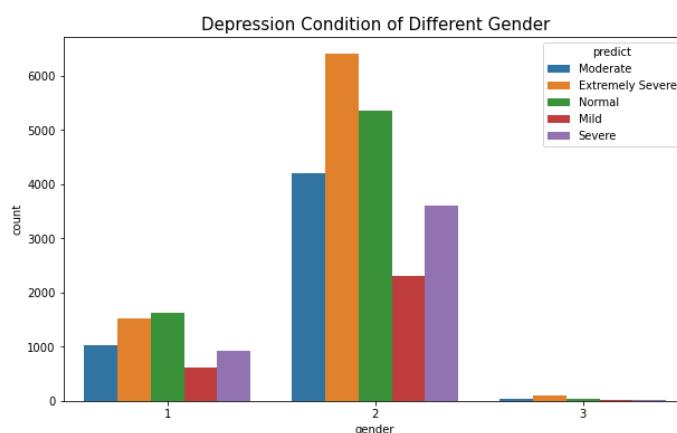
```
In [352]: 1 value_counts = Stress['predict'].value_counts().sort_index()
```

```
In [353]: 1 plt.figure(figsize=(8,8))
2 Stress['predict'].value_counts().plot.pie(autopct='%.1f%%', startangle=0)
3 plt.title('Distribution of Prediction of The Stress Set')
4 plt.show()
```



```
In [354]: 1 plt.figure(figsize=(10,6))
2 sns.countplot(data=Depression, x=Depression.sort_values('gender').gender,hue=Depression['predict'])
3 plt.title('Depression Condition of Different Gender', fontsize=15)
```

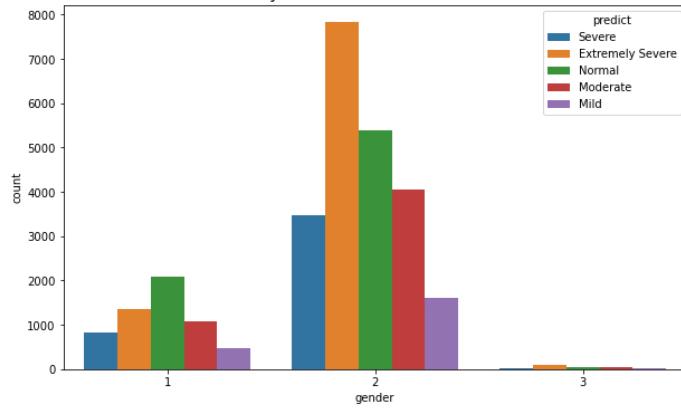
Out[354]: Text(0.5, 1.0, 'Depression Condition of Different Gender')



```
In [355]: 1 plt.figure(figsize=(10,6))
2 sns.countplot(data=Anxiety, x=Anxiety.sort_values('gender').gender,hue=Anxiety['predict'])
3 plt.title('Anxiety Condition of Different Gender', fontsize=15)
```

Out[355]: Text(0.5, 1.0, 'Anxiety Condition of Different Gender')

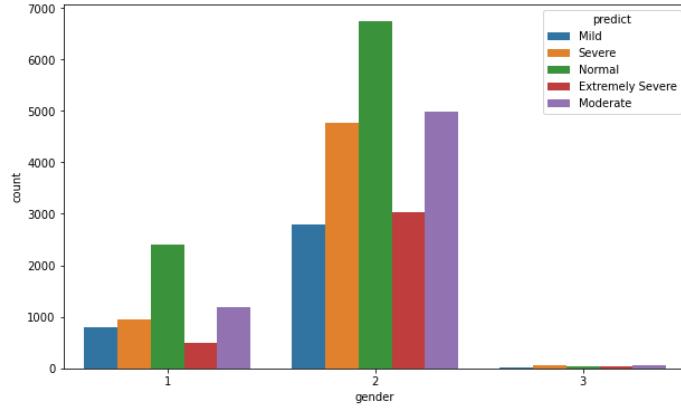
Anxiety Condition of Different Gender



```
In [356]: 1 plt.figure(figsize=(10,6))
2 sns.countplot(data=Stress, x=Stress.sort_values('gender').gender,hue=Stress['predict'])
3 plt.title('Stress Condition of Different Gender',fontsize=15)
```

Out[356]: Text(0.5, 1.0, 'Stress Condition of Different Gender')

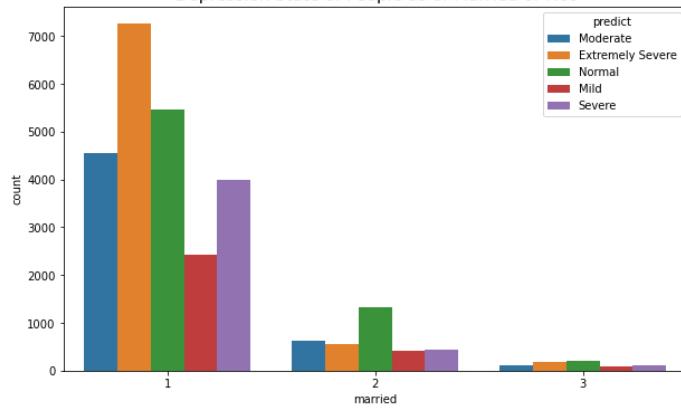
Stress Condition of Different Gender



```
In [357]: 1 plt.figure(figsize=(10,6))
2 sns.countplot(data=Depression, x=Depression.sort_values('married').married,hue=Depression['predict'])
3 plt.title('Depression State of People as of Married or Not',fontsize=15)
```

Out[357]: Text(0.5, 1.0, 'Depression State of People as of Married or Not')

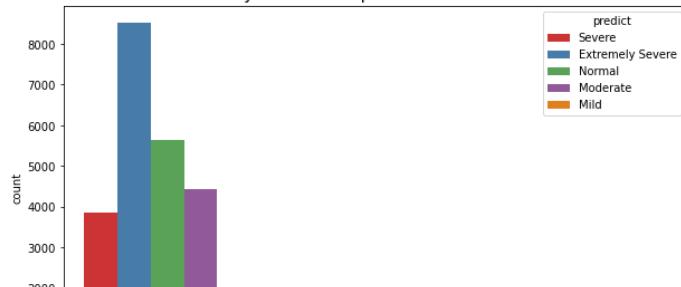
Depression State of People as of Married or Not

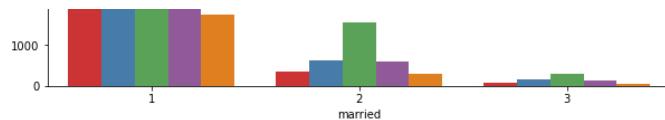


```
In [358]: 1 plt.figure(figsize=(10,6))
2 sns.countplot(data=Anxiety, x=Anxiety.sort_values('married').married,hue=Anxiety['predict'],palette='Set1')
3 plt.title('Anxiety State of People as of Married or Not',fontsize=15)
```

Out[358]: Text(0.5, 1.0, 'Anxiety State of People as of Married or Not')

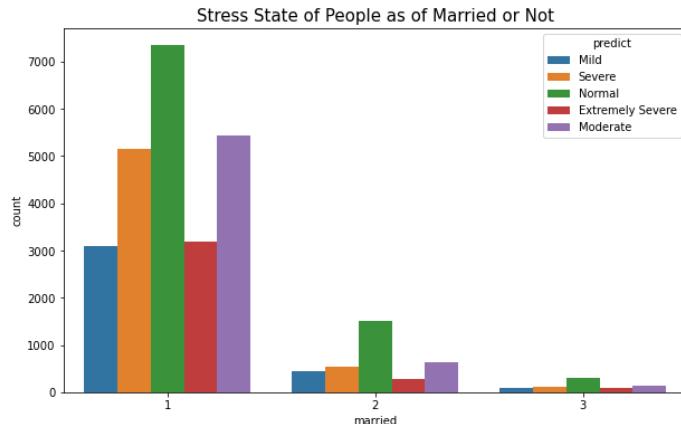
Anxiety State of People as of Married or Not





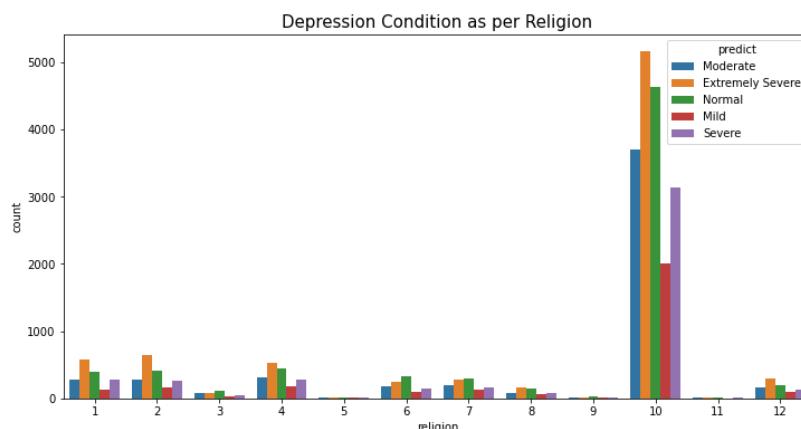
```
In [359]: 1 plt.figure(figsize=(10,6))
2 sns.countplot(data=Stress, x=Stress.sort_values('married').married,hue=Stress['predict'])
3 plt.title('Stress State of People as of Married or Not',fontsize=15)
```

Out[359]: Text(0.5, 1.0, 'Stress State of People as of Married or Not')



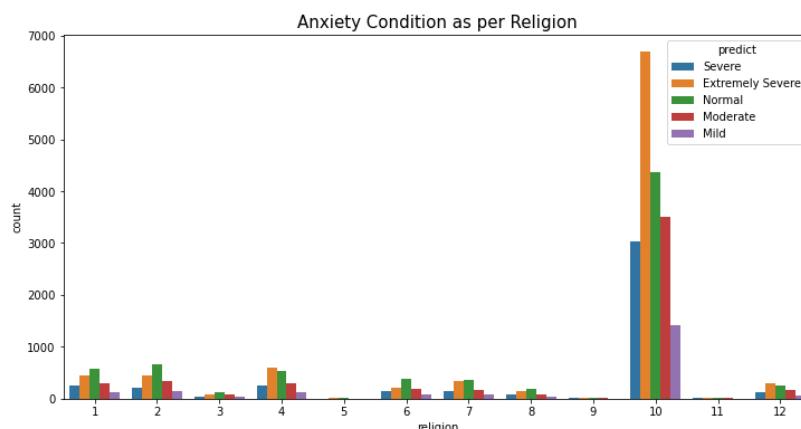
```
In [360]: 1 plt.figure(figsize=(12,6))
2 a=sns.countplot(data=Depression, x=Depression.sort_values('religion').religion,hue=Depression['predict'])
3 plt.title('Depression Condition as per Religion',fontsize=15)
```

Out[360]: Text(0.5, 1.0, 'Depression Condition as per Religion')



```
In [361]: 1 plt.figure(figsize=(12,6))
2 sns.countplot(data=Anxiety, x=Anxiety.sort_values('religion').religion,hue=Anxiety['predict'])
3 plt.title('Anxiety Condition as per Religion',fontsize=15)
```

Out[361]: Text(0.5, 1.0, 'Anxiety Condition as per Religion')

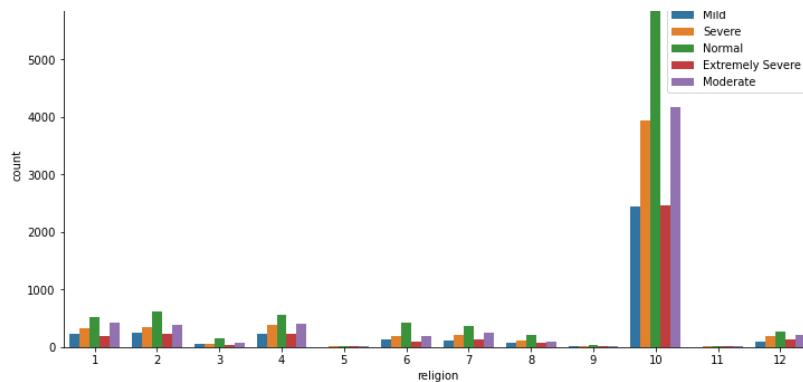


```
In [362]: 1 plt.figure(figsize=(12,6))
2 sns.countplot(data=Stress, x=Stress.sort_values('religion').religion,hue=Stress['predict'])
3 plt.title('Stress Condition as per Religion',fontsize=15)
```

Out[362]: Text(0.5, 1.0, 'Stress Condition as per Religion')

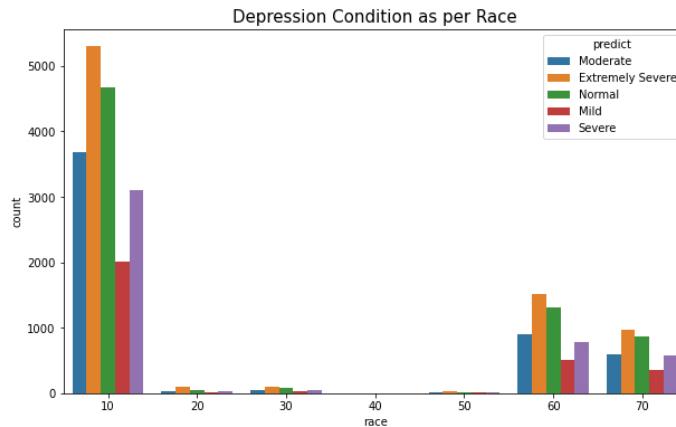
Stress Condition as per Religion





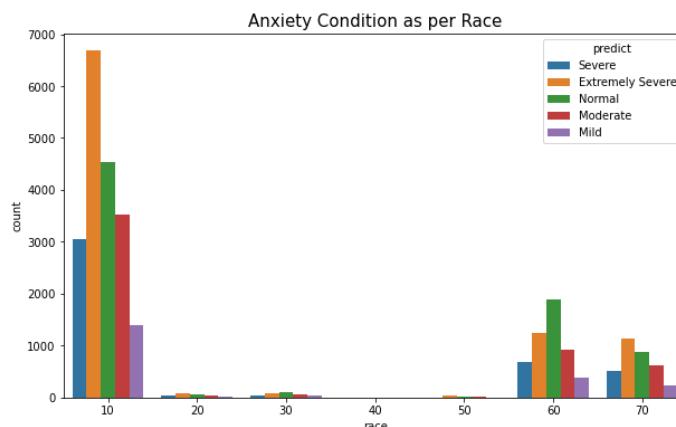
```
In [363]: 1 plt.figure(figsize=(10,6))
2 sns.countplot(data=Depression, x=Depression.sort_values('race').race,hue=Depression['predict'])
3 plt.title('Depression Condition as per Race',fontsize=15)
```

Out[363]: Text(0.5, 1.0, 'Depression Condition as per Race')



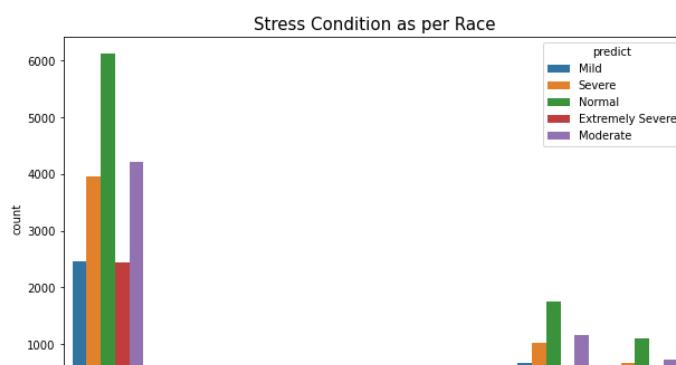
```
In [364]: 1 plt.figure(figsize=(10,6))
2 sns.countplot(data=Anxiety, x=Anxiety.sort_values('race').race,hue=Anxiety['predict'])
3 plt.title('Anxiety Condition as per Race',fontsize=15)
```

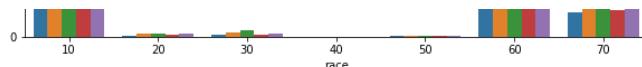
Out[364]: Text(0.5, 1.0, 'Anxiety Condition as per Race')



```
In [365]: 1 plt.figure(figsize=(10,6))
2 sns.countplot(data=Stress, x=Stress.sort_values('race').race,hue=Stress['predict'])
3 plt.title('Stress Condition as per Race',fontsize=15)
```

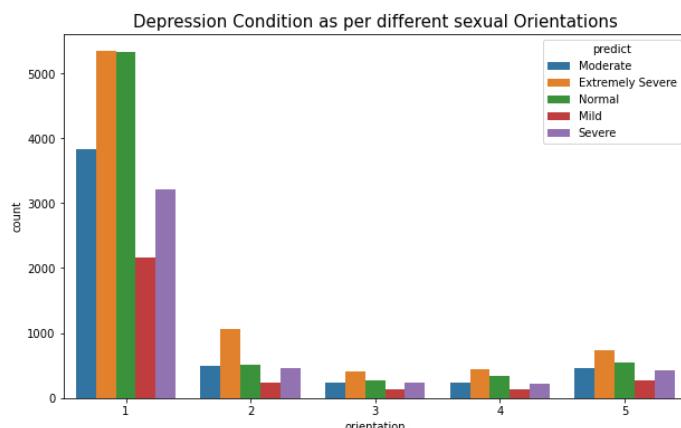
Out[365]: Text(0.5, 1.0, 'Stress Condition as per Race')





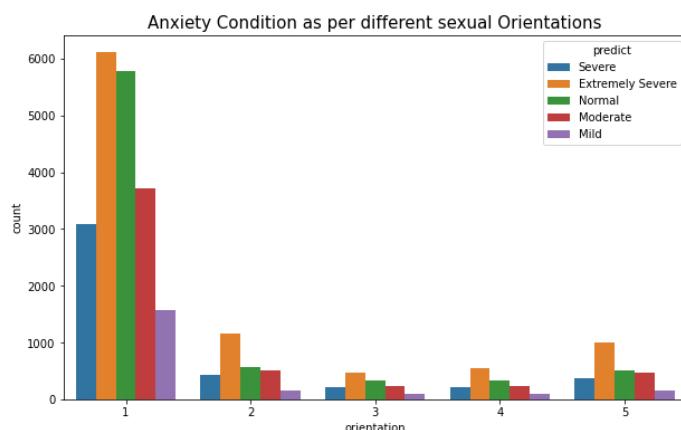
```
In [366]: 1 plt.figure(figsize=(10,6))
2 sns.countplot(data=Depression, x=Depression.sort_values('orientation').orientation,hue=Depression['predict'])
3 plt.title('Depression Condition as per different sexual Orientations',fontsize=15)
```

Out[366]: Text(0.5, 1.0, 'Depression Condition as per different sexual Orientations')



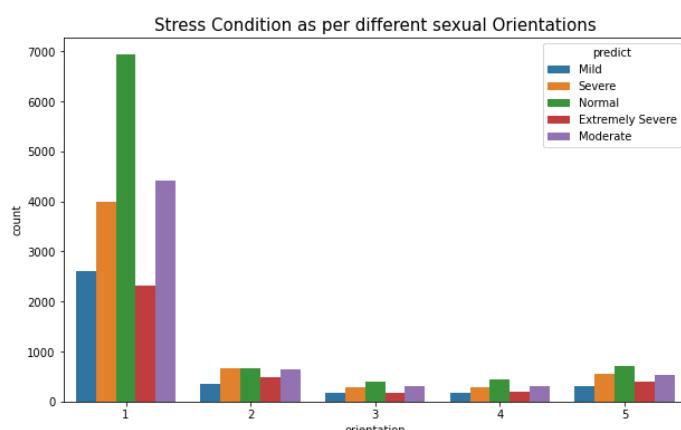
```
In [367]: 1 plt.figure(figsize=(10,6))
2 sns.countplot(data=Anxiety, x=Anxiety.sort_values('orientation').orientation,hue=Anxiety['predict'])
3 plt.title('Anxiety Condition as per different sexual Orientations',fontsize=15)
```

Out[367]: Text(0.5, 1.0, 'Anxiety Condition as per different sexual Orientations')



```
In [368]: 1 plt.figure(figsize=(10,6))
2 sns.countplot(data=Stress, x=Stress.sort_values('orientation').orientation,hue=Stress['predict'])
3 plt.title('Stress Condition as per different sexual Orientations',fontsize=15)
```

Out[368]: Text(0.5, 1.0, 'Stress Condition as per different sexual Orientations')



In [369]: 1 Depression

Out[369]:

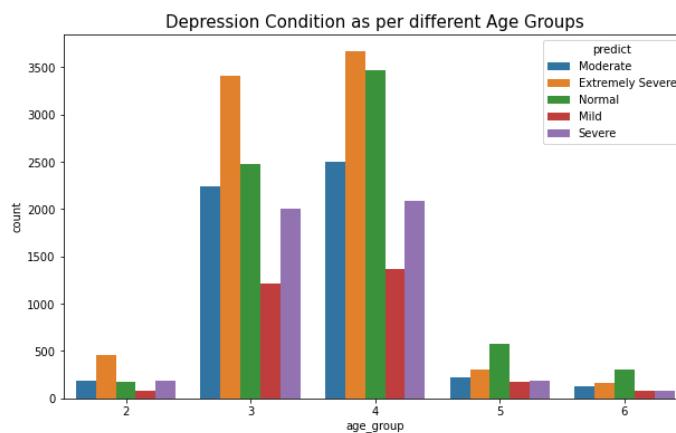
| Q3A | Q5A | Q10A | Q13A | Q16A | Q17A | Q21A | Q24A | Q26A | Q31A | Q34A | Q37A | Q38A | Q42A | Total_Count | country | source | TIP1 | TIP12 | TIP13 | TIP14 |   |
|-----|-----|------|------|------|------|------|------|------|------|------|------|------|------|-------------|---------|--------|------|-------|-------|-------|---|
| 0   | 1   | 2    | 2    | 0    | 1    | 2    | 0    | 0    | 1    | 2    | 3    | 1    | 0    | 1           | 16      | US     | 2    | 1     | 1     | 7     | 4 |
| 1   | 2   | 3    | 2    | 3    | 2    | 3    | 2    | 1    | 3    | 2    | 3    | 2    | 2    | 2           | 32      | MY     | 2    | 2     | 5     | 3     | 6 |
| 2   | 1   | 3    | 3    | 1    | 3    | 1    | 3    | 2    | 1    | 3    | 2    | 3    | 3    | 2           | 31      | MX     | 2    | 2     | 5     | 6     | 5 |
| 3   | 0   | 2    | 0    | 0    | 0    | 0    | 0    | 0    | 1    | 0    | 0    | 0    | 0    | 1           | 4       | GB     | 2    | 7     | 6     | 4     | 5 |

|       | 4 | 2 | 2 | 2 | 3 | 3 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 2 | 3  | 35 | US | 0 | 1 | 4 | 5 | 7 |
|-------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|----|----|----|---|---|---|---|---|
| 28346 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1  | 6  | US | 0 | 6 | 5 | 6 | 7 |
| 28347 | 2 | 2 | 3 | 3 | 3 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 3 | 3  | 38 | MY | 1 | 4 | 5 | 5 | 7 |
| 28348 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0  | 4  | US | 0 | 6 | 6 | 7 | 5 |
| 28349 | 1 | 2 | 0 | 3 | 0 | 1 | 1 | 1 | 2 | 1 | 1 | 0 | 1 | 2 | 16 | US | 2  | 1 | 6 | 5 | 7 |   |
| 28350 | 1 | 3 | 0 | 2 | 3 | 3 | 3 | 1 | 1 | 1 | 1 | 2 | 1 | 3 | 3  | 27 | MY | 1 | 6 | 2 | 3 | 5 |

28351 rows × 60 columns

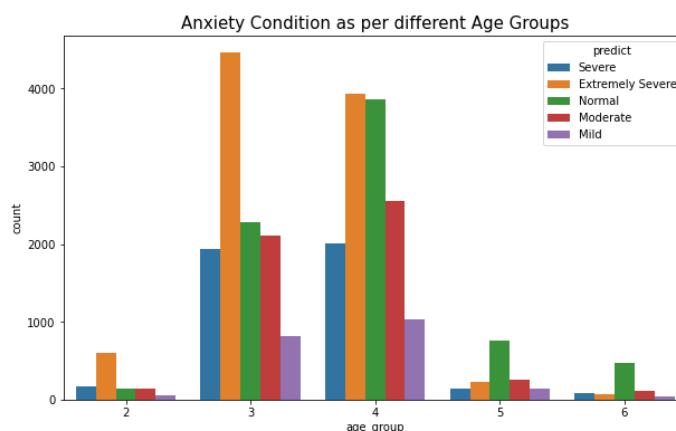
```
In [370]: 1 plt.figure(figsize=(10,6))
2 sns.countplot(data=Depression, x=Depression.sort_values('age_group').age_group,hue=Depression['predict'])
3 plt.title('Depression Condition as per different Age Groups',fontsize=15)
```

Out[370]: Text(0.5, 1.0, 'Depression Condition as per different Age Groups')



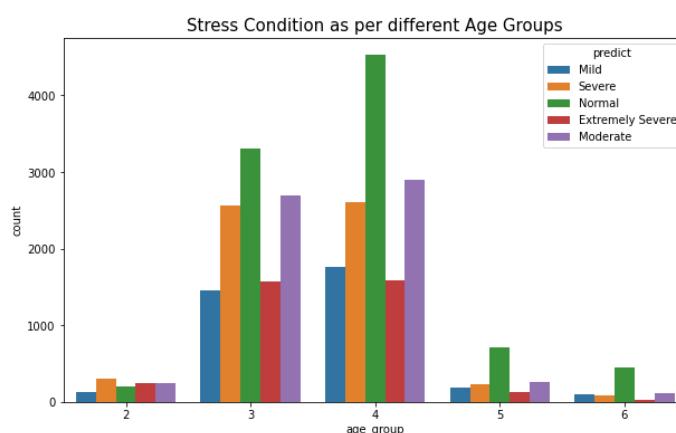
```
In [371]: 1 plt.figure(figsize=(10,6))
2 sns.countplot(data=Anxiety, x=Anxiety.sort_values('age_group').age_group,hue=Anxiety['predict'])
3 plt.title('Anxiety Condition as per different Age Groups',fontsize=15)
```

Out[371]: Text(0.5, 1.0, 'Anxiety Condition as per different Age Groups')



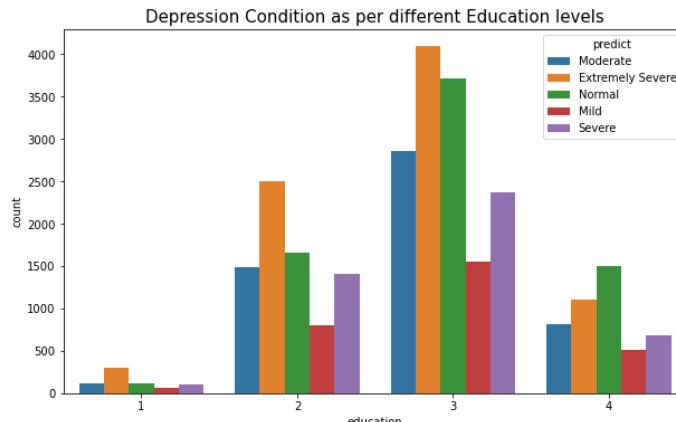
```
In [372]: 1 plt.figure(figsize=(10,6))
2 sns.countplot(data=Stress, x=Stress.sort_values('age_group').age_group,hue=Stress['predict'])
3 plt.title('Stress Condition as per different Age Groups',fontsize=15)
```

Out[372]: Text(0.5, 1.0, 'Stress Condition as per different Age Groups')



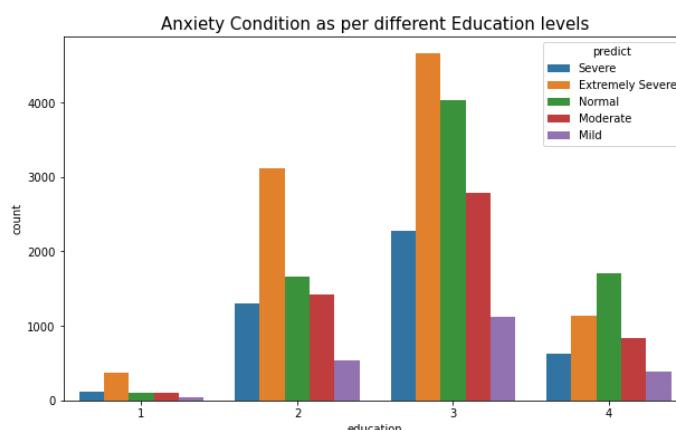
```
In [373]: 1 plt.figure(figsize=(10,6))
2 sns.countplot(data=Depression, x=Depression.sort_values('education').education,hue=Depression['predict'])
3 plt.title("Depression Condition as per different Education levels",fontsize=15)
```

```
Out[373]: Text(0.5, 1.0, 'Depression Condition as per different Education levels')
```



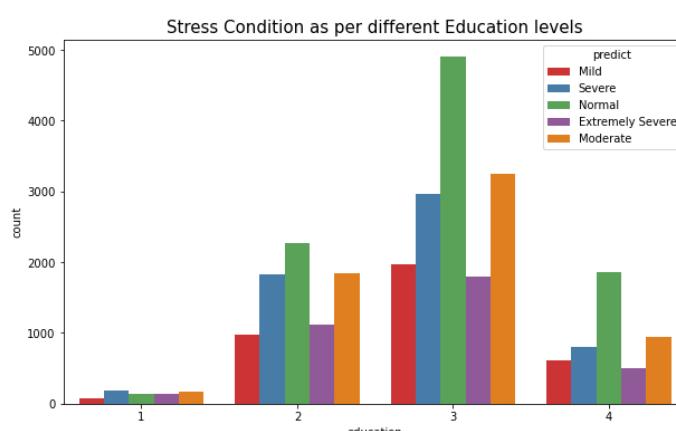
```
In [374]: 1 plt.figure(figsize=(10,6))
2 sns.countplot(data=Anxiety, x=Anxiety.sort_values('education').education,hue=Anxiety['predict'])
3 plt.title('Anxiety Condition as per different Education levels'.fontsize=15)
```

```
Out[374]: Text(0.5, 1.0, 'Anxiety Condition as per different Education levels')
```



```
In [375]: plt.figure(figsize=(10,6))
sns.countplot(data=Stress, x=Stress.sort_values('education').education,hue=Stress['predict'],palette='Set1')
plt.title('Stress Condition as per different Education levels'.fontsize=15)
```

```
Out[375]: Text(0.5 1.0 'Stress Condition as per different Education levels')
```



```
In [376]: 1 columns_to_check = Depression.columns[0::]
2 plot_null_values(Deression, columns_to_check)
```

```

28350 False False
          Q34A Q37A Q38A Q42A Total_Count country source TIPI1 TIPI2 \
0 False False
1 False False
2 False False
3 False False
4 False False
...
...
...
...
...
...
28346 False False
28347 False False
28348 False False
28349 False False
28350 False False

          TIPI3 TIPI4 TIPI5 TIPI6 TIPI7 TIPI8 TIPI9 TIPI10 VCL1 VCL2 \
0 False False
1 False False
2 False False
3 False False
4 False False
...
...
...
...
...
28346 False False
28347 False False
28348 False False
28349 False False
28350 False False

          VCL3 VCL4 VCL5 VCL6 VCL7 VCL8 VCL9 VCL10 VCL11 VCL12 \
0 False False
1 False False
2 False False
3 False False
4 False False
...
...
...
...
...
28346 False False
28347 False False
28348 False False
28349 False False
28350 False False

          VCL13 VCL14 VCL15 VCL16 education urban gender engnat age \
0 False False
1 False False
2 False False
3 False False
4 False False
...
...
...
...
...
28346 False False
28347 False False
28348 False False
28349 False False
28350 False False

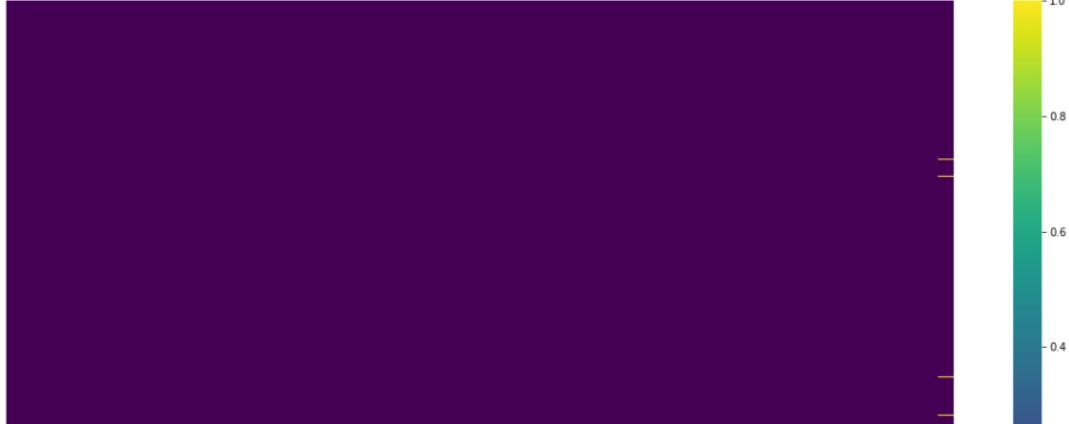
          screensize unquenetworklocation hand religion orientation race \
0 False False
1 False False
2 False False
3 False False
4 False False
...
...
...
...
...
28346 False False
28347 False False
28348 False False
28349 False False
28350 False False

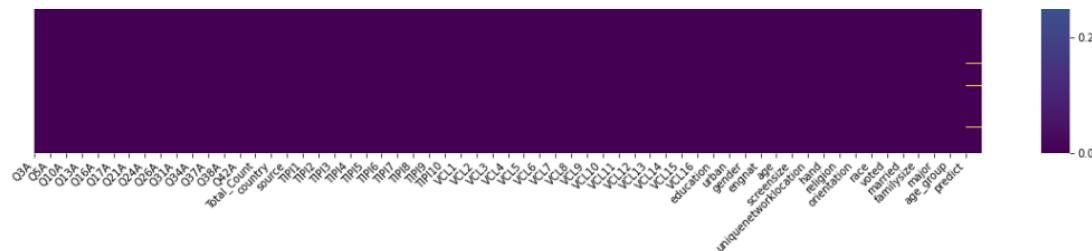
          voted married familysize major age_group predict
0 False False False False False False False False False False
1 False False False False False False False False False False
2 False False False False False False False False False False
3 False False False False False False False False False False
4 False False False False False False False False False False
...
...
...
...
...
28346 False False False False False False False False False False
28347 False False False False False False False False False False
28348 False False False False False False False False False False
28349 False False False False False False False False False False
28350 False False False False False False False False False False

```

[28351 rows x 60 columns]

Null Values Distribution in Dataframe





```
In [377]: 1 Depression.isna().sum()
```

```
Out[377]: Q3A          0
Q5A          0
Q10A         0
Q13A         0
Q16A         0
Q17A         0
Q21A         0
Q24A         0
Q26A         0
Q31A         0
Q34A         0
Q37A         0
Q38A         0
Q42A         0
Total_Count   0
country       0
source        0
TIPI1         0
TIPI2         0
```

```
In [378]: 1 Anxiety.isna().sum()
```

```
Out[378]: Q2A          0
Q4A          0
Q7A          0
Q9A          0
Q15A         0
Q19A         0
Q20A         0
Q23A         0
Q25A         0
Q28A         0
Q30A         0
Q36A         0
Q40A         0
Q41A         0
Total_Count   0
country       0
source        0
TIPI1         0
TIPI2         0
```

```
In [379]: 1 Stress.isna().sum()
```

```
Out[379]: Q1A          0
Q6A          0
Q8A          0
Q11A         0
Q12A         0
Q14A         0
Q18A         0
Q22A         0
Q27A         0
Q29A         0
Q32A         0
Q33A         0
Q35A         0
Q39A         0
Total_Count   0
country       0
source        0
TIPI1         0
TIPI2         0
```

```
In [380]: 1 Depression=Depression.dropna()
```

```
In [381]: 1 Depression.isna().sum()
```

```
Out[381]: Q3A          0
Q5A          0
Q10A         0
Q13A         0
Q16A         0
Q17A         0
Q21A         0
Q24A         0
Q26A         0
Q31A         0
Q34A         0
Q37A         0
Q38A         0
Q42A         0
Total_Count   0
country       0
source        0
TIPI1         0
TIPI2         0
```

```
In [382]: 1 depression.columns
Out[382]: Index(['Q3A', 'Q5A', 'Q10A', 'Q13A', 'Q16A', 'Q17A', 'Q21A', 'Q24A', 'Q26A',
   'Q31A', 'Q34A', 'Q37A', 'Q38A', 'Q42A', 'Total_Count'],
  dtype='object')

In [383]: 1 Depression=Depression.drop(columns=['Total_Count','age', 'engnat', 'screensize', 'uniquenetworklocation', 'hand', 'voted', 'major']
2 Anxiety=Anxiety.drop(columns=['Total_Count','age', 'engnat', 'screensize', 'uniquenetworklocation', 'hand', 'voted', 'major']
3 Stress=Stress.drop(columns=['Total_Count','age', 'engnat', 'screensize', 'uniquenetworklocation', 'hand', 'voted', 'major'],
   ...)
```

```
In [384]: 1 Depression.head()
Out[384]:
   Q3A  Q5A  Q10A  Q13A  Q16A  Q17A  Q21A  Q24A  Q26A  Q31A  Q34A  Q37A  Q38A  Q42A  TIPI1  TIPI2  TIPI3  TIPI4  TIPI5  TIPI6  TIPI7  TIPI8  TIPI9  TI
0    1    2     2     0     1     2     0     0     1     2     3     1     0     1     1     1     1     7     4     6     4     6     1     6
1    2    3     2     3     2     3     2     1     3     2     3     2     2     2     2     2     5     3     6     5     5     5     6     3
2    1    3     3     1     3     1     3     2     1     3     2     3     3     2     2     5     6     5     3     2     6     3     5
3    0    2     0     0     0     0     0     0     1     0     0     0     0     0     1     7     6     4     5     3     2     6     3     5
4    2    2     2     3     3     2     2     2     3     3     3     3     2     3     1     4     5     7     5     7     6     7     1
```

```
In [385]: 1 Anxiety.head()
Out[385]:
   Q2A  Q4A  Q7A  Q9A  Q15A  Q19A  Q20A  Q23A  Q25A  Q28A  Q30A  Q36A  Q40A  Q41A  TIPI1  TIPI2  TIPI3  TIPI4  TIPI5  TIPI6  TIPI7  TIPI8  TIPI9  TI
0    2    0     3     2     1     0     1     0     0     0     2     0     3     3     3     1     1     7     4     6     4     6     1     6
1    1    3     3     3     3     3     3     3     3     3     3     3     3     3     3     2     5     3     6     5     5     5     6     3
2    0    2     2     2     2     0     2     2     1     1     2     1     1     1     1     2     5     6     5     3     2     6     3     5
3    0    0     1     0     1     0     1     0     1     0     0     0     0     0     0     0     7     6     4     5     3     2     6     3     5
4    3    3     3     3     3     3     3     3     3     1     3     2     3     3     3     1     4     5     7     5     7     6     7     1
```

```
In [386]: 1 Stress.head()
Out[386]:
   Q1A  Q6A  Q8A  Q11A  Q12A  Q14A  Q18A  Q22A  Q27A  Q29A  Q32A  Q33A  Q35A  Q39A  TIPI1  TIPI2  TIPI3  TIPI4  TIPI5  TIPI6  TIPI7  TIPI8  TIPI9  TI
0    1    2     1     1     0     3     0     0     3     2     0     1     0     2     1     1     1     7     4     6     4     6     1     6
1    1    1     3     1     3     3     3     3     2     1     1     3     3     2     2     2     5     3     6     5     5     5     6     3
2    0    0     2     0     1     1     1     0     0     1     1     3     1     1     2     5     6     5     3     2     6     3     5
3    0    1    0     1     0     1     1     0     1     0     0     0     0     0     0     1     7     6     4     5     3     2     6     3     5
4    3    3     3     3     3     3     2     3     3     1     3     1     3     1     1     1     4     5     7     5     7     6     7     1
```

## 2.1 Model for Anxiety

```
In [387]: 1 X=Depression[['Q3A', 'Q5A', 'Q10A', 'Q13A', 'Q16A', 'Q17A', 'Q21A', 'Q24A', 'Q26A',
2   'Q31A', 'Q34A', 'Q37A', 'Q38A', 'Q42A', 'TIPI1', 'TIPI2', 'TIPI3', 'TIPI4', 'TIPI5', 'TIPI6', 'TIPI7', 'TIPI8', 'TIPI9', 'TI
   ...]
```

```
In [388]: 1 y=Depression[['predict']].values
2 y = y.flatten()
```

```
In [389]: 1 X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.35,random_state=0)
2 X_train_scaled = scaler.fit_transform(X_train)
3 X_test_scaled = scaler.transform(X_test)
```

```
In [390]: 1 print('Training Set:',X_train.shape,y_train.shape)
2 print('Test Set:',X_test.shape,y_test.shape)

Training Set: (18040, 33) (18040,
Test Set: (9714, 33) (9714,
```

```
In [391]: 1 clf = LazyClassifier(verbose=0,ignore_warnings=True, custom_metric=None)
2 models,predictions = clf.fit(X_train, X_test, y_train, y_test)
3
4 print(models)
97% | 28/29 [09:32<00:07, 7.02s/it]

[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.005769 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 217
[LightGBM] [Info] Number of data points in the train set: 18040, number of used features: 33
[LightGBM] [Info] Start training from score -1.244895
[LightGBM] [Info] Start training from score -2.233001
[LightGBM] [Info] Start training from score -1.656539
[LightGBM] [Info] Start training from score -1.374392
[LightGBM] [Info] Start training from score -1.826158
```

```
100% | 29/29 [09:37<00:00, 19.91s/it]

          Accuracy  Balanced Accuracy  ROC AUC  F1 Score  \
Model      LogisticRegression        1.00        1.00    None     1.00
```

|                               |      |      |      |      |
|-------------------------------|------|------|------|------|
| SVC                           | 0.99 | 0.98 | None | 0.99 |
| NuSVC                         | 0.98 | 0.97 | None | 0.98 |
| QuadraticDiscriminantAnalysis | 0.95 | 0.95 | None | 0.95 |
| LinearDiscriminantAnalysis    | 0.94 | 0.94 | None | 0.94 |
| LGBMClassifier                | 0.95 | 0.93 | None | 0.95 |
| ExtraTreesClassifier          | 0.93 | 0.90 | None | 0.93 |
| RandomForestClassifier        | 0.92 | 0.88 | None | 0.92 |
| GaussianNB                    | 0.87 | 0.87 | None | 0.88 |
| NearestCentroid               | 0.86 | 0.86 | None | 0.86 |
| BaggingClassifier             | 0.85 | 0.81 | None | 0.85 |
| KNeighborsClassifier          | 0.82 | 0.76 | None | 0.81 |
| AdaBoostClassifier            | 0.73 | 0.73 | None | 0.73 |
| DecisionTreeClassifier        | 0.77 | 0.72 | None | 0.77 |
| BernoulliNB                   | 0.78 | 0.72 | None | 0.78 |
| LabelPropagation              | 0.76 | 0.70 | None | 0.76 |
| LabelSpreading                | 0.76 | 0.70 | None | 0.76 |
| ExtraTreeClassifier           | 0.75 | 0.70 | None | 0.75 |
| LinearSVC                     | 0.79 | 0.68 | None | 0.76 |
| CalibratedClassifierCV        | 0.79 | 0.68 | None | 0.76 |
| SGDClassifier                 | 0.73 | 0.64 | None | 0.73 |
| Perceptron                    | 0.73 | 0.64 | None | 0.73 |
| PassiveAggressiveClassifier   | 0.73 | 0.63 | None | 0.72 |
| RidgeClassifier               | 0.59 | 0.46 | None | 0.48 |
| RidgeClassifierCV             | 0.59 | 0.46 | None | 0.48 |
| DummyClassifier               | 0.29 | 0.20 | None | 0.13 |

| Model                         | Time Taken |
|-------------------------------|------------|
| LogisticRegression            | 3.05       |
| SVC                           | 28.12      |
| NuSVC                         | 136.14     |
| QuadraticDiscriminantAnalysis | 0.61       |
| LinearDiscriminantAnalysis    | 1.93       |
| LGBMClassifier                | 4.43       |
| ExtraTreesClassifier          | 5.98       |
| RandomForestClassifier        | 5.89       |
| GaussianNB                    | 0.46       |
| NearestCentroid               | 0.48       |
| BaggingClassifier             | 2.91       |
| KNeighborsClassifier          | 9.52       |
| AdaBoostClassifier            | 5.77       |
| DecisionTreeClassifier        | 0.68       |
| BernoulliNB                   | 0.50       |
| LabelPropagation              | 145.38     |
| LabelSpreading                | 112.09     |
| ExtraTreeClassifier           | 0.50       |
| LinearSVC                     | 23.10      |
| CalibratedClassifierCV        | 83.66      |
| SGDClassifier                 | 1.59       |
| Perceptron                    | 0.76       |
| PassiveAggressiveClassifier   | 0.93       |
| RidgeClassifier               | 0.57       |
| RidgeClassifierCV             | 0.53       |
| DummyClassifier               | 0.25       |

## Algorithm (1): K Nearest Neighbors

```
In [392]: 1 knn_depression = KNeighborsClassifier(n_neighbors=15).fit(X_train_scaled,y_train)

In [393]: 1 accuracy_knn = round(accuracy_score(y_test,knn_depression.predict(X_test_scaled)),3)
2 precision_knn = round(precision_score(y_test,knn_depression.predict(X_test_scaled),average='weighted'),3)
3 recall_knn = round(recall_score(y_test,knn_depression.predict(X_test_scaled),average='weighted'),3)
4 f1_score_knn = round(f1_score(y_test,knn_depression.predict(X_test_scaled),average='weighted'),3)
5 cross_validation_knn = round(np.mean(cross_val_score(knn_depression, X_train_scaled, y_train, cv = 6)),3)

In [394]: 1 print('Accuracy:', accuracy_knn)
2 print('Precision:', precision_knn)
3 print('Recall:', recall_knn)
4 print('F1_Score:', f1_score_knn)
5 print('Cross Validation:', cross_validation_knn)

Accuracy: 0.883
Precision: 0.885
Recall: 0.883
F1_Score: 0.878
Cross Validation: 0.88

In [395]: 1 classification = classification_report(digits=4, y_true=y_test, y_pred=knn_depression.predict(X_test_scaled))
2 print(classification)

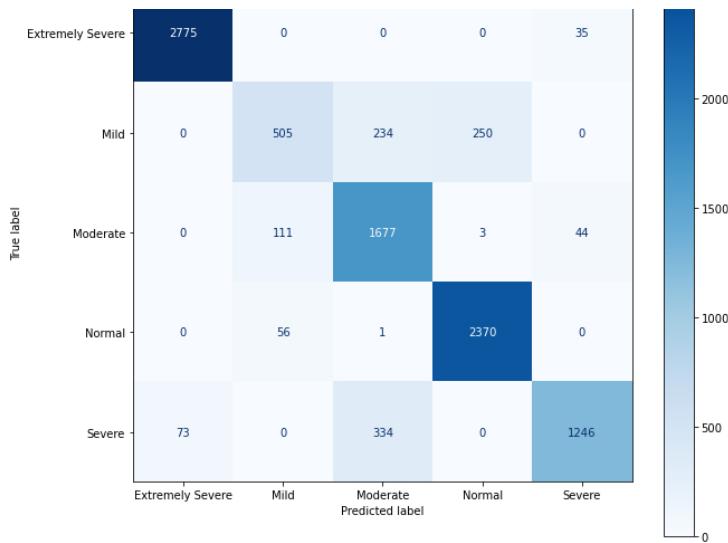
          precision    recall  f1-score   support

  Extremely Severe       0.9744    0.9875    0.9809     2810
        Mild           0.7515    0.5106    0.6081      989
      Moderate       0.7467    0.9139    0.8219     1835
       Normal        0.9035    0.9765    0.9386     2427
      Severe         0.9404    0.7538    0.8368     1653

   accuracy                           0.8825    9714
  macro avg       0.8633    0.8285    0.8373    9714
weighted avg       0.8852    0.8825    0.8778    9714

In [396]: 1 fig, ax = plt.subplots(figsize=(10, 10))
2 plot_confusion_matrix(knn_depression, X_test_scaled, y_test, ax=ax, cmap='Blues')

Out[396]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x14bbf6d2b50>
```



## Algorithm (2): Decision Tree

```
In [397]: 1 decision_tree_depression = DecisionTreeClassifier(criterion='entropy', splitter='best', min_samples_split=9, min_samples_leaf=2)
```

```
In [398]: 1 accuracy_dt = round(accuracy_score(y_test,decision_tree_depression.predict(X_test_scaled)),3)
2 precision_dt = round(precision_score(y_test,decision_tree_depression.predict(X_test_scaled),average='weighted'),3)
3 recall_dt = round(recall_score(y_test,decision_tree_depression.predict(X_test_scaled),average='weighted'),3)
4 f1_score_dt = round(f1_score(y_test,decision_tree_depression.predict(X_test_scaled),average='weighted'),3)
5 cross_validation_dt = round(np.mean(cross_val_score(decision_tree_depression, X_train_scaled, y_train, cv = 6)),3)
```

```
In [399]: 1 print('Accuracy:', accuracy_dt)
2 print('Precision:', precision_dt)
3 print('Recall:', recall_dt)
4 print('F1_Score:', f1_score_dt)
5 print('Cross Validation:', cross_validation_dt)
```

```
Accuracy: 0.788  
Precision: 0.789  
Recall: 0.788  
F1_Score: 0.788  
Cross Validation: 0.787
```

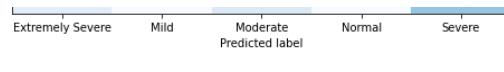
```
In [400]: 1 classification = classification_report(digits=4, y_true=y_test, y_pred=decision_tree_depression.predict(X_test_scaled))
2 print(classification)
```

|                  | precision | recall | f1-score | support |
|------------------|-----------|--------|----------|---------|
| Extremely Severe | 0.9034    | 0.9320 | 0.9175   | 2810    |
| Mild             | 0.5317    | 0.5844 | 0.5568   | 989     |
| Moderate         | 0.6694    | 0.6741 | 0.6717   | 1835    |
| Normal           | 0.9222    | 0.8982 | 0.9100   | 2427    |
| Severe           | 0.6840    | 0.6273 | 0.6545   | 1653    |
| accuracy         |           |        | 0.7876   | 9714    |
| macro avg        | 0.7421    | 0.7432 | 0.7421   | 9714    |
| weighted avg     | 0.7887    | 0.7876 | 0.7877   | 9714    |

```
In [401]: 1 fig, ax = plt.subplots(figsize=(10, 10))
2 plot_confusion_matrix(decision_tree_depression, X_test_scaled, y_test, ax=ax, cmap='Blues')
```

```
Out[401]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x149c1886be0>
```





### Algorithm (3): Random Forest

```
In [402]: 1 random_forest_depression = RandomForestClassifier(n_estimators=200, min_samples_split=3, min_samples_leaf=1, max_depth=160,
```

```
In [403]: 1 accuracy_rf = round(accuracy_score(y_test, random_forest_depression.predict(X_test_scaled)),3)
2 precision_rf = round(precision_score(y_test, random_forest_depression.predict(X_test_scaled),average='weighted'), 3)
3 recall_rf = round(recall_score(y_test, random_forest_depression.predict(X_test_scaled), average='weighted'), 3)
4 f1_Score_rf = round(f1_score(y_test, random_forest_depression.predict(X_test_scaled), average='weighted'), 3)
5 cross_validation_rf = round(np.mean(cross_val_score(random_forest_depression, X_train_scaled, y_train, cv = 6)), 3)
```

```
In [404]: 1 print('Accuracy:', accuracy_rf)
2 print('Precision:', precision_rf)
3 print('Recall:', recall_rf)
4 print('F1_Score:', f1_Score_rf)
5 print('Cross Validation:', cross_validation_rf)
```

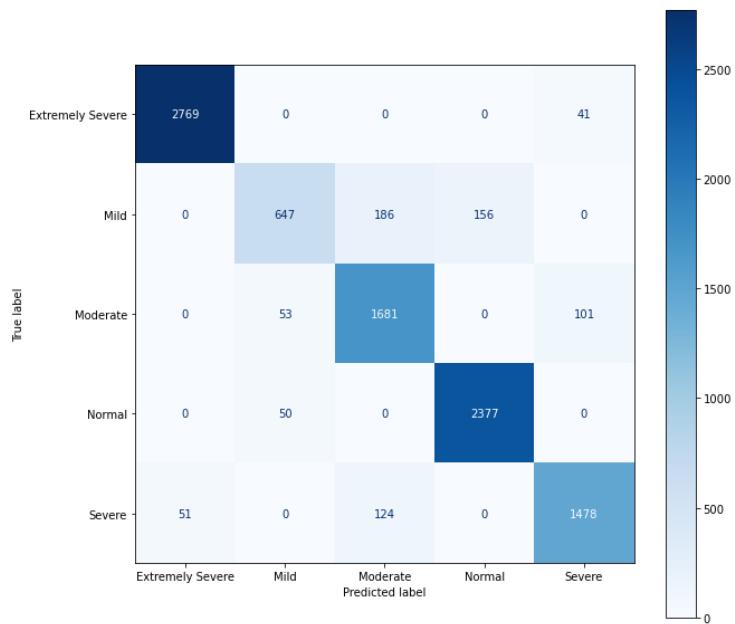
```
Accuracy: 0.922
Precision: 0.921
Recall: 0.922
F1_Score: 0.919
Cross Validation: 0.924
```

```
In [405]: 1 classification=classification_report(digits=4, y_true=y_test, y_pred=random_forest_depression.predict(X_test_scaled))
2 print(classification)
```

|                  | precision | recall | f1-score | support |
|------------------|-----------|--------|----------|---------|
| Extremely Severe | 0.9819    | 0.9854 | 0.9837   | 2810    |
| Mild             | 0.8627    | 0.6542 | 0.7441   | 989     |
| Moderate         | 0.8443    | 0.9161 | 0.8787   | 1835    |
| Normal           | 0.9384    | 0.9794 | 0.9585   | 2427    |
| Severe           | 0.9123    | 0.8941 | 0.9031   | 1653    |
| accuracy         |           |        | 0.9216   | 9714    |
| macro avg        | 0.9079    | 0.8858 | 0.8936   | 9714    |
| weighted avg     | 0.9211    | 0.9216 | 0.9195   | 9714    |

```
In [406]: 1 fig, ax = plt.subplots(figsize=(10, 10))
2 plot_confusion_matrix(random_forest_depression,X_test_scaled,y_test,ax=ax, cmap='Blues')
```

Out[406]: <sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x14bc0d357c0>



### Algorithm (4): Naive Bayes

```
In [407]: 1 gaussian_nb_depression = GaussianNB().fit(X_train_scaled, y_train)
```

```
In [408]: 1 accuracy_nb = round(accuracy_score(y_test, gaussian_nb_depression.predict(X_test_scaled)), 3)
2 precision_nb = round(precision_score(y_test, gaussian_nb_depression.predict(X_test_scaled), average='weighted'), 3)
3 recall_nb = round(recall_score(y_test, gaussian_nb_depression.predict(X_test_scaled), average='weighted'), 3)
4 f1_score_nb = round(f1_score(y_test, gaussian_nb_depression.predict(X_test_scaled), average='weighted'), 3)
5 cross_validation_nb = round(np.mean(cross_val_score(gaussian_nb_depression, X_train_scaled, y_train, cv = 6)), 3)
```

```
In [409]: 1 print('Accuracy:', accuracy_nb)
2 print('Precision:', precision_nb)
3 print('Recall:', recall_nb)
4 print('F1_Score:', f1_score_nb)
5 print('Cross Validation:', cross_validation_nb)
```

Accuracy: 0.872

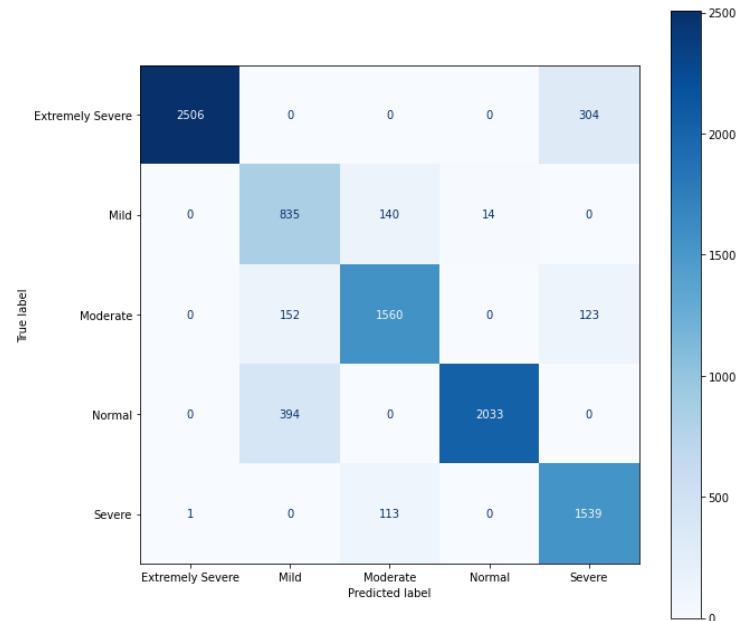
```
Precision: 0.895
Recall: 0.872
F1_Score: 0.878
Cross Validation: 0.873
```

```
In [410]: 1 classification=classification_report(digits=4, y_true=y_test, y_pred=gaussian_nb_depression.predict(X_test_scaled))
2 print(classification)
```

|                  | precision | recall | f1-score | support |
|------------------|-----------|--------|----------|---------|
| Extremely Severe | 0.9996    | 0.8918 | 0.9426   | 2810    |
| Mild             | 0.6046    | 0.8443 | 0.7046   | 989     |
| Moderate         | 0.8605    | 0.8501 | 0.8553   | 1835    |
| Normal           | 0.9932    | 0.8377 | 0.9088   | 2427    |
| Severe           | 0.7828    | 0.9310 | 0.8595   | 1653    |
| accuracy         |           |        | 0.8722   | 9714    |
| macro avg        | 0.8481    | 0.8710 | 0.8524   | 9714    |
| weighted avg     | 0.8946    | 0.8722 | 0.8778   | 9714    |

```
In [411]: 1 fig, ax = plt.subplots(figsize=(10, 10))
2 plot_confusion_matrix(gaussian_nb_depression, X_test_scaled, y_test, ax=ax, cmap='Blues')
```

Out[411]: <sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x14bbee0a90>



## Algorithm (5): Logistic Regression

```
In [412]: 1 logistic_regression_depression = LogisticRegression().fit(X_train_scaled, y_train)
```

```
In [413]: 1 accuracy_lr = round(accuracy_score(y_test, logistic_regression_depression.predict(X_test_scaled)), 3)
2 precision_lr = round(precision_score(y_test, logistic_regression_depression.predict(X_test_scaled), average='weighted'), 3)
3 recall_lr = round(recall_score(y_test, logistic_regression_depression.predict(X_test_scaled), average='weighted'), 3)
4 f1_score_lr = round(f1_score(y_test, logistic_regression_depression.predict(X_test_scaled), average='weighted'), 3)
5 cross_validation_lr = round(np.mean(cross_val_score(logistic_regression_depression, X_train_scaled, y_train, cv=6)), 3)
```

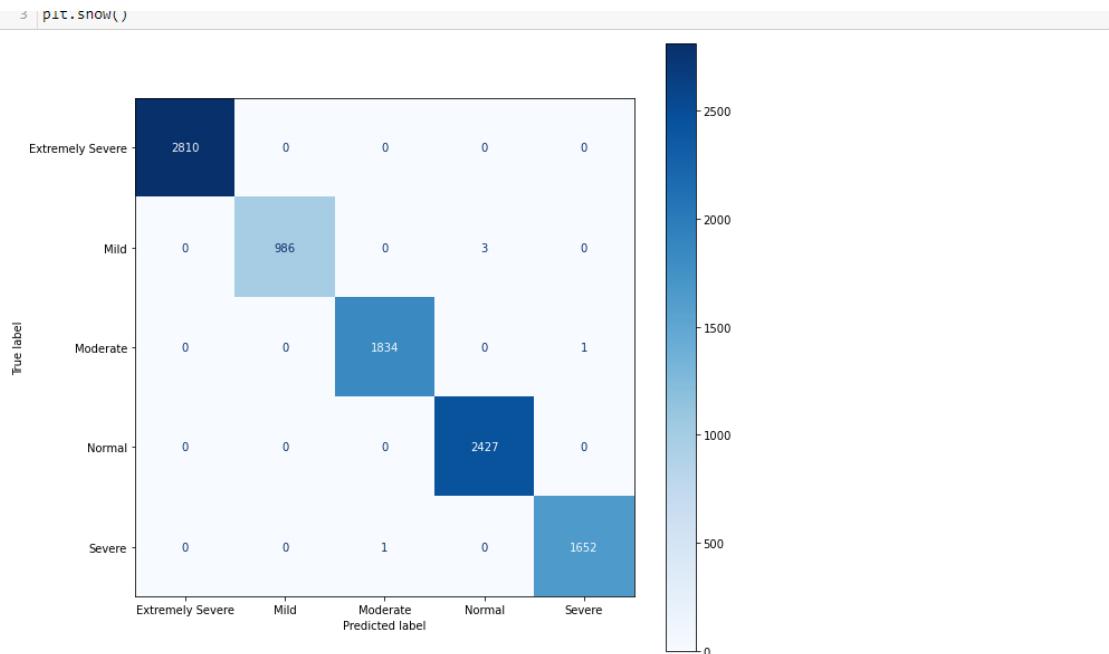
```
In [414]: 1 print('Accuracy:', accuracy_lr)
2 print('F1_Score:', f1_score_lr)
3 print('Recall:', recall_lr)
4 print('Precision:', precision_lr)
5 print('Cross Validation:', cross_validation_lr)
```

```
Accuracy: 0.999
F1_Score: 0.999
Recall: 0.999
Precision: 0.999
Cross Validation: 0.998
```

```
In [415]: 1 classification = classification_report(digits=4, y_true=y_test, y_pred=logistic_regression_depression.predict(X_test_scaled))
2 print(classification)
```

|                  | precision | recall | f1-score | support |
|------------------|-----------|--------|----------|---------|
| Extremely Severe | 1.0000    | 1.0000 | 1.0000   | 2810    |
| Mild             | 1.0000    | 0.9970 | 0.9985   | 989     |
| Moderate         | 0.9995    | 0.9995 | 0.9995   | 1835    |
| Normal           | 0.9988    | 1.0000 | 0.9994   | 2427    |
| Severe           | 0.9994    | 0.9994 | 0.9994   | 1653    |
| accuracy         |           |        | 0.9995   | 9714    |
| macro avg        | 0.9995    | 0.9992 | 0.9993   | 9714    |
| weighted avg     | 0.9995    | 0.9995 | 0.9995   | 9714    |

```
In [416]: 1 fig, ax = plt.subplots(figsize=(10, 10))
2 plot_confusion_matrix(logistic_regression_depression, X_test_scaled, y_test, ax=ax, cmap='Blues')
```



### Algorithm (5): Support Vector Classifier

```
In [417]: 1 svm_depression = SVC(C=100, gamma=0.1, kernel='rbf').fit(X_train_scaled,y_train)

In [418]: 1 accuracy_svm = round(accuracy_score(y_test, svm_depression.predict(X_test_scaled)), 3)
2 precision_svm = round(precision_score(y_test, svm_depression.predict(X_test_scaled), average='weighted'), 3)
3 recall_svm = round(recall_score(y_test, svm_depression.predict(X_test_scaled), average='weighted'), 3)
4 f1_score_svm = round(f1_score(y_test, svm_depression.predict(X_test_scaled), average='weighted'), 3)
5 cross_validation_svm = round(np.mean(cross_val_score(svm_depression, X_train_scaled, y_train, cv = 6)), 3)

In [419]: 1 print('Accuracy:', accuracy_svm)
2 print('F1_Score:', f1_score_svm)
3 print('Recall_Score:', recall_svm)
4 print('Precision_Score:', precision_svm)
5 print('Cross Validation Score:', cross_validation_svm)

Accuracy: 1.0
F1_Score: 1.0
Recall_Score: 1.0
Precision_Score: 1.0
Cross Validation Score: 1.0

In [420]: 1 classification=classification_report(digits=4, y_true=y_test, y_pred=svm_depression.predict(X_test_scaled))
2 print(classification)

          precision    recall  f1-score   support

Extremely Severe      1.0000    1.0000    1.0000     2810
Mild                  1.0000    1.0000    1.0000      989
Moderate               1.0000    1.0000    1.0000     1835
Normal                 1.0000    1.0000    1.0000     2427
Severe                 1.0000    1.0000    1.0000     1653

   accuracy                           1.0000      9714
  macro avg       1.0000    1.0000    1.0000      9714
weighted avg        1.0000    1.0000    1.0000      9714

In [421]: 1 fig, ax = plt.subplots(figsize=(6, 6))
2 plot_confusion_matrix(svm_depression, X_test_scaled, y_test,ax=ax, cmap='Blues')

Out[421]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x14bb7815a90>
```

|            |                  | Extremely Severe | Mild | Moderate | Normal | Severe |
|------------|------------------|------------------|------|----------|--------|--------|
| True label | Extremely Severe | 2810             | 0    | 0        | 0      | 0      |
|            | Mild             | 0                | 989  | 0        | 0      | 0      |
| Moderate   | 0                | 0                | 1835 | 0        | 0      |        |
|            | Normal           | 0                | 0    | 0        | 2427   | 0      |
| Severe     | 0                | 0                | 0    | 0        | 1653   |        |

```
In [422]: 1 Result={'Model':['Random_Forest', 'Decision_Tree', 'Naive_Baise', 'K_Nearest_Neighbors', 'Logistic_Regression', 'SVM'],
2           'Accuracy(%)':[accuracy_rf*100, accuracy_dt*100, accuracy_nb*100, accuracy_knn*100, accuracy_lr*100, accuracy_svm*100]}
```

```
3     'Precision(%)': [precision_rf*100, precision_dt*100, precision_nb*100, precision_knn*100, precision_lr*100, precision_svm*100],
4     'Recall(%)': [recall_rf*100, recall_dt*100, recall_nb*100, recall_knn*100, recall_lr*100, recall_svm*100],
5     'F1_Score(%)': [f1_Score_rf*100, f1_score_dt*100, f1_score_nb*100, f1_score_knn*100, f1_score_lr*100, f1_score_svm*100]
```

```
In [423]: 1 Result_Depression=pd.DataFrame(Result)
2 Result_Depression.to_csv('D:/COURSES/Programming_Language/python/data_science_track/projects/NTI/Graduation_Project/Datasets/')

In [424]: 1 Result_Depression
```

```
Out[424]:
```

|   | Model               | Accuracy(%) | Precision(%) | Recall(%) | F1_Score(%) |
|---|---------------------|-------------|--------------|-----------|-------------|
| 0 | Random_Forest       | 92.20       | 92.10        | 92.20     | 91.90       |
| 1 | Decision_Tree       | 78.80       | 78.90        | 78.80     | 78.80       |
| 2 | Naive_Baise         | 87.20       | 89.50        | 87.20     | 87.80       |
| 3 | K_Nearest_Neighbors | 88.30       | 88.50        | 88.30     | 87.80       |
| 4 | Logistic_Regression | 99.90       | 99.90        | 99.90     | 99.90       |
| 5 | SVM                 | 100.00      | 100.00       | 100.00    | 100.00      |

## Test by My Self

```
In [425]: 1 Depression
```

```
Out[425]:
```

|       | Q3A | Q5A | Q10A | Q13A | Q16A | Q17A | Q21A | Q24A | Q26A | Q31A | Q34A | Q37A | Q38A | Q42A | TIP1 | TIP12 | TIP13 | TIP14 | TIP15 | TIP16 | TIP17 | TIP18 | TIP |
|-------|-----|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|-------|-------|-------|-------|-------|-------|-------|-----|
| 0     | 1   | 2   | 2    | 0    | 1    | 2    | 0    | 0    | 1    | 2    | 3    | 1    | 2    | 0    | 1    | 1     | 1     | 7     | 4     | 6     | 4     | 6     | 1   |
| 1     | 2   | 3   | 2    | 3    | 2    | 3    | 2    | 1    | 3    | 2    | 3    | 2    | 2    | 2    | 2    | 5     | 3     | 6     | 5     | 5     | 5     | 6     |     |
| 2     | 1   | 3   | 3    | 1    | 3    | 1    | 3    | 2    | 1    | 3    | 2    | 3    | 3    | 2    | 2    | 2     | 5     | 6     | 5     | 3     | 2     | 6     | 3   |
| 3     | 0   | 2   | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 1    | 0    | 0    | 0    | 0    | 1    | 7     | 6     | 4     | 5     | 3     | 2     | 6     | 3   |
| 4     | 2   | 2   | 2    | 3    | 3    | 2    | 2    | 2    | 3    | 3    | 3    | 3    | 2    | 3    | 1    | 4     | 5     | 7     | 5     | 7     | 6     | 7     |     |
| ...   | ... | ... | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...   | ...   | ...   | ...   | ...   | ...   | ...   |     |
| 28346 | 0   | 1   | 0    | 1    | 1    | 0    | 0    | 1    | 1    | 0    | 0    | 0    | 0    | 0    | 1    | 6     | 5     | 6     | 7     | 6     | 4     | 6     | 5   |
| 28347 | 2   | 2   | 3    | 3    | 3    | 2    | 3    | 3    | 3    | 3    | 3    | 2    | 3    | 3    | 4    | 5     | 5     | 7     | 4     | 6     | 4     | 7     |     |
| 28348 | 1   | 0   | 0    | 1    | 0    | 0    | 0    | 1    | 0    | 1    | 0    | 0    | 0    | 0    | 0    | 6     | 6     | 7     | 5     | 6     | 3     | 6     | 1   |
| 28349 | 1   | 2   | 0    | 3    | 0    | 1    | 1    | 1    | 2    | 1    | 1    | 0    | 1    | 2    | 1    | 6     | 5     | 7     | 3     | 5     | 3     | 5     |     |
| 28350 | 1   | 3   | 0    | 2    | 3    | 3    | 3    | 1    | 1    | 1    | 2    | 1    | 3    | 3    | 6    | 2     | 3     | 5     | 6     | 3     | 5     | 5     |     |

27754 rows × 34 columns

```
In [426]: 1 Depression[Depression['predict'] == 'Mild']
```

```
Out[426]:
```

|       | Q3A | Q5A | Q10A | Q13A | Q16A | Q17A | Q21A | Q24A | Q26A | Q31A | Q34A | Q37A | Q38A | Q42A | TIP1 | TIP12 | TIP13 | TIP14 | TIP15 | TIP16 | TIP17 | TIP18 | TIP |
|-------|-----|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|-------|-------|-------|-------|-------|-------|-------|-----|
| 9     | 1   | 0   | 0    | 2    | 0    | 2    | 0    | 1    | 3    | 0    | 0    | 0    | 0    | 0    | 2    | 1     | 7     | 5     | 7     | 1     | 6     | 5     | 4   |
| 22    | 0   | 2   | 1    | 1    | 1    | 0    | 1    | 1    | 1    | 3    | 0    | 0    | 0    | 1    | 0    | 1     | 4     | 4     | 3     | 5     | 6     | 1     | 3   |
| 27    | 0   | 1   | 0    | 1    | 1    | 0    | 1    | 1    | 0    | 1    | 1    | 0    | 0    | 0    | 3    | 4     | 6     | 5     | 6     | 6     | 4     | 4     | 2   |
| 31    | 1   | 1   | 2    | 1    | 1    | 1    | 0    | 1    | 0    | 1    | 0    | 1    | 0    | 0    | 3    | 2     | 4     | 1     | 5     | 6     | 4     | 3     | 6   |
| 61    | 1   | 3   | 0    | 1    | 0    | 0    | 1    | 0    | 1    | 0    | 0    | 0    | 1    | 0    | 3    | 3     | 5     | 6     | 6     | 7     | 2     | 6     | 6   |
| ...   | ... | ... | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...   | ...   | ...   | ...   | ...   | ...   | ...   |     |
| 28308 | 0   | 1   | 1    | 2    | 0    | 1    | 1    | 1    | 1    | 1    | 1    | 0    | 1    | 1    | 1    | 6     | 5     | 2     | 6     | 7     | 5     | 5     |     |
| 28324 | 1   | 0   | 0    | 0    | 2    | 0    | 0    | 1    | 1    | 2    | 0    | 2    | 0    | 2    | 5    | 5     | 4     | 4     | 4     | 5     | 6     | 6     |     |
| 28338 | 1   | 1   | 1    | 2    | 0    | 2    | 0    | 0    | 1    | 1    | 2    | 1    | 0    | 0    | 5    | 6     | 7     | 4     | 6     | 4     | 5     | 5     |     |
| 28340 | 0   | 1   | 0    | 2    | 1    | 1    | 0    | 0    | 3    | 2    | 0    | 0    | 0    | 1    | 1    | 2     | 7     | 2     | 7     | 4     | 6     | 1     |     |
| 28342 | 0   | 1   | 1    | 1    | 1    | 0    | 1    | 1    | 1    | 0    | 1    | 1    | 2    | 3    | 2    | 6     | 2     | 4     | 6     | 6     | 3     | 5     |     |

2923 rows × 34 columns

```
In [427]: 1 Depression.columns
```

```
Out[427]:
```

Index(['Q3A', 'Q5A', 'Q10A', 'Q13A', 'Q16A', 'Q17A', 'Q21A', 'Q24A', 'Q26A', 'Q31A', 'Q34A', 'Q37A', 'Q38A', 'Q42A', 'TIP1', 'TIP12', 'TIP13', 'TIP14', 'TIP15', 'TIP16', 'TIP17', 'TIP18', 'TIP', 'education', 'urban', 'gender', 'religion', 'orientation', 'race', 'married', 'family\_size', 'age\_group', 'predict'], dtype='object')

```
In [428]: 1 input_data_Moderate = [1, 2, 2, 0, 1, 2, 0, 0, 1, 2, 3, 1, 0, 1, 1, 1, 7, 4, 6, 4, 6, 1, 6, 1, 1, 3, 2, 4, 5, 70, 1, 5, 2]
2 input_data_scaled = scaler.transform([input_data_Moderate])
```

```
In [429]: 1 prediction = svm_depression.predict(input_data_scaled)
2 print("Prediction:", prediction[0])
```

Prediction: Moderate

```
In [430]: 1 input_data_Mild = [1, 0, 0, 2, 0, 1, 3, 0, 0, 0, 2, 1, 7, 5, 7, 1, 6, 5, 4, 1, 3, 2, 2, 7, 1, 60, 1, 2, 3]
2 input_data_scaled = scaler.transform([input_data_Mild])
```

```
In [431]: 1 prediction = svm_depression.predict(input_data_scaled)
2 print("Prediction:", prediction[0])
```

Prediction: Mild

```
In [432]: 1 input_data_Exremely_Severe = [2, 2, 3, 3, 3, 2, 3, 3, 3, 3, 3, 2, 3, 3, 4, 5, 5, 7, 4, 6, 4, 7, 4, 4, 3, 2, 1, 1, 10, 1,  
2 input_data_scaled = scaler.transform([input_data_Exremely_Severe])
```

```
In [433]: 1 prediction = svm_depression.predict(input_data_scaled)  
2 print("Prediction:", prediction[0])
```

Prediction: Extremely Severe

```
In [434]: 1 input_data_Normal = [0, 2, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 7, 6, 4, 5, 3, 2, 6, 3, 5, 2, 4, 2, 2, 2, 60, 1, 2, 4]  
2 input_data_scaled = scaler.transform([input_data_Normal])
```

```
In [435]: 1 prediction = svm_depression.predict(input_data_scaled)  
2 print("Prediction:", prediction[0])
```

Prediction: Normal

```
In [436]: 1 input_data_Severe = [1, 3, 0, 2, 3, 3, 3, 1, 1, 1, 2, 1, 3, 3, 6, 2, 3, 5, 6, 3, 5, 5, 1, 2, 3, 3, 2, 10, 1, 10, 1, 4, 4]  
2 input_data_scaled = scaler.transform([input_data_Severe])
```

```
In [437]: 1 prediction = svm_depression.predict(input_data_scaled)  
2 print("Prediction:", prediction[0])
```

Prediction: Severe

```
In [438]: 1 column_scales = {  
2     'Q3A': list(range(4)),  
3     'Q5A': list(range(4)),  
4     'Q10A': list(range(4)),  
5     'Q13A': list(range(4)),  
6     'Q16A': list(range(4)),  
7     'Q17A': list(range(4)),  
8     'Q21A': list(range(4)),  
9     'Q24A': list(range(4)),  
10    'Q26A': list(range(4)),  
11    'Q31A': list(range(4)),  
12    'Q34A': list(range(4)),  
13    'Q37A': list(range(4)),  
14    'Q38A': list(range(4)),  
15    'Q42A': list(range(4)),  
16    'TIPI1': list(range(1, 8)),  
17    'TIPI2': list(range(1, 8)),  
18    'TIPI3': list(range(1, 8)),  
19    'TIPI4': list(range(1, 8)),  
20    'TIPI5': list(range(1, 8)),  
21    'TIPI6': list(range(1, 8)),  
22    'TIPI7': list(range(1, 8)),  
23    'TIPI8': list(range(1, 8)),  
24    'TIPI9': list(range(1, 8)),  
25    'TIPI10': list(range(1, 8)),  
26    'education': list(range(1, 5)),  
27    'urban': list(range(1, 4)),  
28    'gender': list(range(1, 4)),  
29    'religion': list(range(1, 13)),  
30    'orientation': list(range(1, 6)),  
31    'race': [10, 20, 30, 40, 50, 60, 70],  
32    'married': list(range(1, 4)),  
33    'familysize': list(range(1, 101)),  
34    'age_group': list(range(1, 7))  
35 }
```

```
In [439]: 1 random_lists = []  
2 for i in range(30):  
3     random_list = [  
4         random.choice(column_scales[column]) for column in [  
5             'Q3A', 'Q5A', 'Q10A', 'Q13A', 'Q16A', 'Q17A', 'Q21A', 'Q24A', 'Q26A',  
6             'Q31A', 'Q34A', 'Q37A', 'Q38A', 'Q42A', 'TIPI1', 'TIPI2', 'TIPI3',  
7             'TIPI4', 'TIPI5', 'TIPI6', 'TIPI7', 'TIPI8', 'TIPI9', 'TIPI10',  
8             'education', 'urban', 'gender', 'religion', 'orientation', 'race',  
9             'married', 'familysize', 'age_group'  
10        ]  
11    ]  
12    random_lists.append(random_list)
```

```
In [440]: 1 for i, random_list in enumerate(random_lists, start=1):  
2     print(f"random_values{i} = {random_list}")
```

```
random_values1 = [3, 1, 1, 3, 0, 1, 1, 0, 2, 2, 3, 1, 0, 4, 7, 2, 6, 3, 2, 3, 5, 7, 4, 2, 1, 3, 11, 4, 40, 3, 27, 2]  
random_values2 = [0, 0, 2, 3, 0, 2, 3, 2, 0, 0, 0, 1, 3, 1, 3, 3, 4, 3, 6, 7, 6, 3, 5, 4, 2, 3, 3, 4, 5, 50, 1, 93, 5]  
random_values3 = [0, 2, 1, 0, 0, 3, 1, 1, 2, 1, 0, 3, 2, 1, 5, 1, 3, 3, 6, 6, 6, 2, 1, 6, 3, 2, 2, 3, 11, 5, 60, 3, 21, 6]  
random_values4 = [0, 0, 3, 1, 0, 3, 1, 2, 0, 2, 0, 2, 1, 2, 1, 6, 3, 1, 4, 7, 5, 2, 1, 3, 1, 1, 3, 4, 3, 40, 1, 36, 3]  
random_values5 = [1, 0, 3, 3, 0, 3, 2, 3, 0, 3, 3, 0, 2, 0, 5, 4, 7, 4, 6, 2, 1, 4, 6, 5, 2, 1, 3, 6, 3, 30, 3, 92, 6]  
random_values6 = [0, 0, 1, 3, 2, 0, 1, 3, 0, 1, 0, 1, 1, 7, 6, 3, 7, 1, 1, 4, 1, 7, 1, 4, 1, 1, 5, 2, 40, 3, 94, 5]  
random_values7 = [0, 3, 1, 1, 2, 3, 2, 2, 0, 3, 2, 2, 1, 5, 6, 6, 3, 2, 7, 3, 3, 4, 5, 3, 3, 3, 2, 1, 10, 2, 67, 4]  
random_values8 = [3, 1, 2, 1, 3, 2, 1, 0, 2, 3, 0, 3, 0, 0, 6, 4, 4, 7, 1, 4, 5, 2, 3, 1, 3, 1, 3, 5, 2, 10, 2, 68, 2]  
random_values9 = [2, 1, 1, 2, 3, 1, 3, 1, 2, 3, 2, 3, 1, 2, 7, 4, 6, 3, 7, 4, 6, 1, 6, 1, 2, 1, 8, 3, 70, 1, 59, 3]  
random_values10 = [0, 3, 0, 1, 2, 0, 2, 2, 1, 3, 3, 2, 0, 3, 2, 4, 5, 1, 4, 4, 1, 7, 6, 5, 3, 1, 2, 9, 5, 10, 3, 25, 2]  
random_values11 = [0, 2, 3, 1, 0, 1, 2, 0, 1, 2, 1, 0, 3, 0, 3, 6, 3, 7, 2, 7, 1, 3, 3, 2, 3, 3, 6, 1, 18, 1, 90, 4]  
random_values12 = [1, 0, 1, 3, 3, 3, 0, 1, 2, 3, 2, 1, 2, 0, 6, 1, 5, 4, 1, 5, 1, 5, 4, 3, 2, 1, 1, 9, 2, 60, 1, 56, 5]  
random_values13 = [3, 2, 3, 3, 3, 2, 1, 1, 0, 0, 2, 3, 0, 2, 6, 5, 2, 6, 6, 4, 4, 2, 3, 4, 1, 1, 8, 4, 60, 1, 12, 3]  
random_values14 = [0, 1, 3, 0, 1, 3, 0, 0, 0, 3, 0, 3, 2, 2, 7, 2, 2, 1, 6, 2, 6, 1, 2, 1, 2, 3, 4, 1, 40, 3, 77, 3]  
random_values15 = [0, 1, 2, 1, 0, 0, 1, 0, 3, 1, 2, 2, 0, 7, 4, 7, 2, 7, 2, 4, 3, 6, 6, 1, 1, 2, 10, 5, 50, 3, 13, 2]  
random_values16 = [0, 1, 0, 0, 3, 2, 0, 0, 2, 0, 1, 0, 0, 2, 5, 1, 6, 3, 6, 4, 5, 3, 3, 1, 3, 6, 2, 30, 3, 77, 6]  
random_values17 = [0, 0, 3, 2, 1, 0, 1, 0, 3, 0, 2, 0, 2, 3, 1, 2, 6, 3, 4, 5, 6, 2, 5, 1, 3, 1, 3, 4, 70, 2, 31, 4]  
random_values18 = [1, 3, 2, 0, 0, 2, 2, 3, 2, 0, 2, 2, 1, 0, 5, 7, 1, 6, 4, 5, 7, 7, 1, 1, 3, 1, 1, 12, 5, 60, 1, 92, 5]  
random_values19 = [0, 1, 2, 3, 3, 2, 2, 3, 3, 1, 0, 3, 3, 7, 4, 2, 3, 3, 4, 2, 6, 1, 2, 1, 1, 4, 5, 60, 1, 13, 5]  
random_values20 = [2, 2, 2, 2, 1, 1, 2, 3, 2, 1, 1, 2, 2, 6, 2, 1, 2, 4, 3, 4, 3, 1, 1, 4, 2, 2, 1, 1, 20, 2, 95, 4]  
random_values21 = [2, 1, 1, 2, 0, 3, 1, 0, 0, 0, 3, 0, 3, 1, 7, 1, 6, 4, 1, 3, 1, 3, 3, 7, 3, 2, 3, 11, 5, 50, 2, 30, 6]  
random_values22 = [2, 1, 1, 1, 3, 2, 1, 2, 0, 0, 0, 1, 0, 4, 5, 4, 5, 7, 5, 7, 4, 1, 2, 2, 3, 6, 3, 60, 2, 12, 2]  
random_values23 = [2, 2, 2, 0, 1, 2, 3, 1, 0, 2, 3, 3, 1, 1, 2, 2, 5, 5, 6, 4, 4, 1, 2, 6, 1, 1, 2, 5, 3, 30, 3, 99, 1]
```

```

random_values24 = [0, 3, 3, 0, 4, 3, 3, 0, 4, 3, 3, 0, 4, 4, /, 1, 1, 4, 3, 3, 11, 1, 50, 1, 38, 1]
random_values25 = [1, 2, 3, 1, 0, 0, 3, 1, 0, 1, 1, 2, 3, 0, 5, 3, 1, 3, 2, 1, 7, 3, 4, 3, 4, 3, 2, 6, 2, 50, 1, 44, 1]
random_values26 = [2, 2, 1, 2, 2, 2, 1, 2, 2, 0, 3, 1, 2, 5, 6, 1, 2, 6, 1, 1, 6, 7, 1, 2, 3, 8, 3, 20, 1, 14, 1]
random_values27 = [0, 2, 2, 3, 1, 3, 1, 2, 2, 2, 3, 3, 3, 1, 7, 6, 1, 1, 4, 3, 3, 1, 2, 4, 3, 2, 1, 8, 4, 20, 3, 100, 4]
random_values28 = [3, 0, 1, 3, 1, 3, 1, 2, 3, 0, 1, 1, 0, 3, 4, 3, 5, 6, 7, 1, 1, 7, 1, 1, 2, 3, 8, 1, 10, 2, 53, 2]
random_values29 = [2, 1, 2, 2, 2, 1, 2, 3, 2, 2, 3, 0, 2, 1, 4, 1, 4, 6, 4, 6, 7, 7, 7, 4, 3, 3, 6, 5, 10, 3, 26, 1]
random_values30 = [1, 0, 1, 2, 1, 1, 2, 3, 2, 1, 1, 0, 3, 0, 3, 3, 4, 2, 6, 4, 5, 6, 1, 1, 3, 12, 4, 30, 2, 87, 5]

```

```

In [441]: 1 random_values1 = [0, 0, 3, 0, 0, 0, 3, 0, 0, 0, 2, 1, 2, 3, 3, 4, 7, 1, 6, 2, 3, 7, 3, 2, 1, 4, 2, 60, 1, 5, 4]
2 random_values2 = [1, 0, 0, 0, 1, 2, 0, 3, 1, 0, 2, 3, 1, 0, 5, 3, 2, 4, 7, 1, 6, 2, 3, 7, 1, 3, 2, 6, 4, 30, 2, 3, 1]
3 random_values3 = [0, 0, 0, 1, 3, 0, 2, 1, 3, 2, 0, 1, 3, 2, 3, 5, 2, 7, 4, 1, 6, 2, 2, 7, 3, 4, 1, 3, 8, 1, 40, 3, 7, 2]
4 random_values4 = [0, 0, 1, 0, 0, 3, 1, 2, 0, 1, 3, 2, 0, 1, 7, 1, 5, 3, 6, 4, 2, 7, 3, 6, 2, 4, 1, 10, 5, 50, 1, 4, 3]
5 random_values5 = [0, 1, 3, 0, 0, 3, 1, 0, 2, 1, 3, 0, 2, 3, 2, 6, 4, 5, 1, 7, 3, 2, 5, 1, 3, 2, 1, 4, 2, 70, 2, 6, 2]
6 random_values6 = [2, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 2, 4, 1, 6, 3, 5, 2, 7, 4, 3, 6, 4, 1, 3, 8, 1, 10, 3, 3, 4]
7 random_values7 = [3, 0, 0, 1, 3, 2, 0, 1, 3, 0, 2, 1, 3, 0, 6, 5, 2, 7, 4, 1, 6, 2, 7, 6, 4, 1, 3, 8, 1, 20, 1, 7, 1]
8 random_values8 = [0, 3, 1, 0, 0, 1, 3, 0, 0, 2, 1, 3, 2, 1, 2, 7, 1, 5, 3, 6, 4, 2, 7, 3, 3, 5, 2, 7, 4, 30, 2, 4, 5]
9 random_values9 = [2, 1, 3, 0, 0, 2, 3, 1, 0, 2, 1, 3, 0, 2, 1, 4, 2, 6, 1, 5, 4, 1, 3, 2, 6, 4, 40, 3, 6, 3]
10 random_values10 = [1, 2, 0, 3, 1, 0, 2, 3, 1, 0, 0, 3, 1, 2, 6, 4, 1, 3, 5, 2, 7, 6, 3, 5, 3, 5, 2, 7, 4, 50, 1, 3, 4]
11 random_values11 = [1, 0, 2, 0, 1, 0, 1, 0, 1, 0, 2, 3, 5, 2, 7, 4, 1, 6, 2, 7, 3, 4, 1, 3, 8, 1, 40, 3, 7, 2]
12 random_values12 = [0, 3, 1, 2, 0, 3, 1, 0, 0, 1, 3, 0, 0, 1, 7, 1, 5, 3, 6, 4, 2, 7, 3, 6, 2, 4, 1, 10, 5, 50, 1, 4, 3]
13 random_values13 = [2, 1, 3, 0, 2, 3, 1, 0, 0, 1, 3, 0, 0, 3, 2, 6, 4, 5, 1, 7, 3, 2, 5, 1, 3, 2, 1, 4, 2, 70, 2, 6, 2]
14 random_values14 = [1, 2, 0, 3, 1, 0, 2, 3, 1, 0, 0, 3, 1, 0, 4, 1, 6, 3, 5, 2, 7, 4, 3, 6, 4, 1, 3, 8, 1, 10, 3, 3, 4]
15 random_values15 = [0, 3, 0, 2, 1, 3, 2, 0, 1, 3, 0, 0, 1, 3, 0, 0, 3, 5, 2, 7, 4, 1, 6, 2, 7, 6, 4, 1, 3, 1, 20, 1, 7, 1]
16 random_values16 = [1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 2, 1, 2, 7, 1, 5, 3, 6, 4, 2, 7, 3, 3, 5, 2, 7, 4, 30, 2, 4, 5]
17 random_values17 = [2, 1, 3, 0, 2, 2, 3, 1, 0, 2, 1, 3, 0, 2, 1, 3, 2, 1, 4, 2, 6, 1, 5, 4, 1, 3, 2, 6, 4, 40, 3, 6, 3]
18 random_values18 = [1, 2, 0, 3, 1, 0, 2, 3, 1, 2, 0, 3, 0, 2, 6, 4, 1, 3, 5, 2, 7, 6, 3, 5, 3, 5, 2, 7, 4, 50, 1, 3, 4]
19 random_values19 = [0, 0, 2, 1, 3, 0, 0, 1, 0, 0, 0, 0, 2, 3, 5, 2, 7, 4, 1, 6, 2, 7, 3, 4, 1, 3, 8, 1, 40, 3, 7, 2]
20 random_values20 = [0, 3, 1, 2, 0, 3, 1, 0, 2, 0, 1, 0, 2, 0, 1, 7, 1, 5, 3, 6, 4, 2, 7, 3, 6, 2, 4, 1, 10, 5, 50, 1, 4, 3]
21 random_values21 = [0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 1, 0, 1, 2, 1, 2, 3, 3, 4, 7, 1, 6, 2, 3, 7, 3, 2, 1, 4, 2, 60, 1, 5, 4]
22 random_values22 = [1, 2, 0, 3, 1, 2, 0, 3, 0, 0, 2, 3, 1, 0, 5, 3, 2, 4, 7, 1, 6, 2, 3, 7, 1, 3, 2, 6, 4, 30, 2, 3, 1]
23 random_values23 = [3, 0, 2, 1, 3, 0, 2, 0, 3, 2, 0, 1, 3, 2, 3, 5, 2, 7, 4, 1, 6, 2, 7, 3, 4, 1, 3, 8, 1, 40, 3, 7, 2]
24 random_values24 = [0, 3, 1, 2, 0, 3, 0, 2, 0, 1, 3, 2, 0, 1, 7, 1, 5, 3, 6, 4, 2, 7, 3, 6, 2, 4, 1, 10, 5, 50, 1, 4, 3]
25 random_values25 = [3, 1, 3, 3, 2, 0, 3, 0, 2, 1, 3, 3, 2, 3, 2, 6, 4, 5, 1, 7, 3, 2, 5, 1, 3, 2, 1, 4, 2, 70, 2, 6, 2]
26 random_values26 = [1, 0, 3, 0, 0, 0, 0, 0, 1, 0, 1, 0, 2, 2, 4, 1, 6, 3, 5, 2, 7, 4, 3, 6, 4, 1, 3, 8, 1, 10, 3, 3, 4]
27 random_values27 = [3, 0, 2, 0, 3, 2, 0, 1, 3, 0, 2, 1, 3, 0, 6, 3, 5, 2, 7, 4, 6, 2, 7, 6, 4, 1, 3, 8, 1, 20, 1, 7, 1]
28 random_values28 = [3, 1, 3, 3, 2, 0, 3, 3, 2, 1, 3, 2, 3, 1, 2, 7, 1, 5, 3, 6, 4, 2, 7, 3, 3, 5, 2, 7, 4, 30, 2, 4, 5]
29 random_values29 = [2, 0, 3, 0, 2, 2, 3, 1, 0, 2, 1, 3, 0, 2, 1, 3, 2, 1, 4, 2, 6, 1, 5, 4, 1, 3, 2, 6, 4, 40, 3, 6, 3]
30 random_values30 = [0, 2, 0, 3, 1, 0, 2, 3, 1, 2, 0, 3, 1, 2, 6, 4, 1, 3, 5, 2, 7, 6, 3, 5, 3, 2, 7, 4, 50, 1, 3, 4]

```

```

In [442]: 1 input_data_scaled = scaler.transform([random_values1])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

```

Prediction: Normal

```

In [443]: 1 input_data_scaled = scaler.transform([random_values2])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

```

Prediction: Moderate

```

In [444]: 1 input_data_scaled = scaler.transform([random_values3])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

```

Prediction: Severe

```

In [445]: 1 input_data_scaled = scaler.transform([random_values4])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

```

Prediction: Moderate

```

In [446]: 1 input_data_scaled = scaler.transform([random_values5])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

```

Prediction: Moderate

```

In [447]: 1 input_data_scaled = scaler.transform([random_values6])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

```

Prediction: Normal

```

In [448]: 1 input_data_scaled = scaler.transform([random_values7])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

```

Prediction: Moderate

```

In [449]: 1 input_data_scaled = scaler.transform([random_values8])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

```

Prediction: Moderate

```

In [450]: 1 input_data_scaled = scaler.transform([random_values9])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

```

Prediction: Moderate

```

In [451]: 1 input_data_scaled = scaler.transform([random_values10])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

```

Prediction: Moderate

```

In [452]: 1 input_data_scaled = scaler.transform([random_values11])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

```

```
Prediction: Mild

In [453]: 1 input_data_scaled = scaler.transform([random_values12])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Moderate

In [454]: 1 input_data_scaled = scaler.transform([random_values13])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Moderate

In [455]: 1 input_data_scaled = scaler.transform([random_values14])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Moderate

In [456]: 1 input_data_scaled = scaler.transform([random_values15])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Moderate

In [457]: 1 input_data_scaled = scaler.transform([random_values16])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Mild

In [458]: 1 input_data_scaled = scaler.transform([random_values17])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Severe

In [459]: 1 input_data_scaled = scaler.transform([random_values18])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Moderate

In [460]: 1 input_data_scaled = scaler.transform([random_values19])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Normal

In [461]: 1 input_data_scaled = scaler.transform([random_values20])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Moderate

In [462]: 1 input_data_scaled = scaler.transform([random_values21])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Normal

In [463]: 1 input_data_scaled = scaler.transform([random_values22])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Moderate

In [464]: 1 input_data_scaled = scaler.transform([random_values23])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Severe

In [465]: 1 input_data_scaled = scaler.transform([random_values24])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Moderate

In [466]: 1 input_data_scaled = scaler.transform([random_values25])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Extremely Severe

In [467]: 1 input_data_scaled = scaler.transform([random_values26])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Mild

In [468]: 1 input_data_scaled = scaler.transform([random_values27])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Moderate

In [469]: 1 input_data_scaled = scaler.transform([random_values28])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])
```

```

Prediction: Extremely Severe

In [470]: 1 input_data_scaled = scaler.transform([random_values29])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Severe

In [471]: 1 input_data_scaled = scaler.transform([random_values30])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Moderate

```

## 2.1 Model for Anxiety

```

In [472]: 1 Anxiety.columns

Out[472]: Index(['Q2A', 'Q4A', 'Q7A', 'Q9A', 'Q15A', 'Q19A', 'Q20A', 'Q23A', 'Q25A',
   'Q28A', 'Q30A', 'Q36A', 'Q40A', 'Q41A', 'TIPI1', 'TIPI2', 'TIPI3',
   'TIPI4', 'TIPI5', 'TIPI6', 'TIPI7', 'TIPI8', 'TIPI9', 'TIPI10',
   'education', 'urban', 'gender', 'religion', 'orientation', 'race',
   'married', 'familysize', 'age_group', 'predict'],
  dtype='object')

In [473]: 1 X=Anxiety[['Q2A', 'Q4A', 'Q7A', 'Q9A', 'Q15A', 'Q19A', 'Q20A', 'Q23A', 'Q25A',
2   'Q28A', 'Q30A', 'Q36A', 'Q40A', 'Q41A', 'TIPI1', 'TIPI2', 'TIPI3',
3   'TIPI4', 'TIPI5', 'TIPI6', 'TIPI7', 'TIPI8', 'TIPI9', 'TIPI10',
4   'education', 'urban', 'gender', 'religion', 'orientation', 'race',
5   'married', 'familysize', 'age_group']]

In [474]: 1 y=Anxiety[['predict']].values
2 y = y.flatten()

In [475]: 1 X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.35,random_state=0)
2 X_train_scaled = scaler.fit_transform(X_train)
3 X_test_scaled = scaler.transform(X_test)

In [476]: 1 print('Training Set:',X_train.shape,y_train.shape)
2 print('Test Set:',X_test.shape,y_test.shape)

Training Set: (18428, 33) (18428,)
Test Set: (9923, 33) (9923,)

In [477]: 1 clf = LazyClassifier(verbose=0,ignore_warnings=True, custom_metric=None)
2 models, predictions = clf.fit(X_train, X_test, y_train, y_test)
3 print(models)

97% | 28/29 [07:19<00:06, 6.73s/it]
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.006827 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 217
[LightGBM] [Info] Number of data points in the train set: 18428, number of used features: 33
[LightGBM] [Info] Start training from score -1.119615
[LightGBM] [Info] Start training from score -2.618221
[LightGBM] [Info] Start training from score -1.696291
[LightGBM] [Info] Start training from score -1.323820
[LightGBM] [Info] Start training from score -1.889264
100% | 29/29 [07:23<00:00, 15.28s/it]

          Accuracy  Balanced Accuracy  ROC AUC  F1 Score  \
Model
LogisticRegression      1.00        1.00    None     1.00
SVC                     0.98        0.96    None     0.98
QuadraticDiscriminantAnalysis  0.96        0.95    None     0.96
LGBMClassifier          0.90        0.83    None     0.90
GaussianNB              0.81        0.77    None     0.82
NearestCentroid          0.75        0.74    None     0.77
ExtraTreesClassifier     0.86        0.74    None     0.84
LinearDiscriminantAnalysis  0.84        0.73    None     0.81
RandomForestClassifier    0.84        0.71    None     0.81
AdaBoostClassifier       0.71        0.70    None     0.72
BaggingClassifier         0.79        0.69    None     0.78
LinearSVC                 0.82        0.66    None     0.79
BernoulliNB               0.77        0.66    None     0.76
CalibratedClassifierCV    0.80        0.64    None     0.76
KNeighborsClassifier      0.75        0.63    None     0.73
SGDClassifier             0.77        0.62    None     0.76
DecisionTreeClassifier    0.71        0.61    None     0.71
PassiveAggressiveClassifier  0.74        0.59    None     0.73
LabelSpreading             0.70        0.59    None     0.69
LabelPropagation           0.70        0.59    None     0.69
ExtraTreeClassifier        0.68        0.58    None     0.68
Perceptron                  0.73        0.57    None     0.71
RidgeClassifier            0.63        0.44    None     0.52
RidgeClassifierCV           0.63        0.44    None     0.52
DummyClassifier             0.33        0.20    None     0.16

          Time Taken
Model
LogisticRegression           2.61

```

|                               |        |
|-------------------------------|--------|
| SVC                           | 34.73  |
| QuadraticDiscriminantAnalysis | 0.78   |
| LGBMClassifier                | 3.70   |
| GaussianNB                    | 0.34   |
| NearestCentroid               | 0.38   |
| ExtraTreesClassifier          | 5.81   |
| LinearDiscriminantAnalysis    | 0.91   |
| RandomForestClassifier        | 8.76   |
| AdaBoostClassifier            | 6.04   |
| BaggingClassifier             | 3.04   |
| LinearSVC                     | 21.75  |
| BernoulliNB                   | 0.53   |
| CalibratedClassifierCV        | 84.96  |
| KNeighborsClassifier          | 7.91   |
| SGDClassifier                 | 1.12   |
| DecisionTreeClassifier        | 0.67   |
| PassiveAggressiveClassifier   | 0.80   |
| LabelSpreading                | 140.44 |
| LabelPropagation              | 114.24 |
| ExtraTreeClassifier           | 0.40   |
| Perceptron                    | 0.87   |
| RidgeClassifier               | 0.50   |
| RidgeClassifierCV             | 0.53   |
| DummyClassifier               | 0.30   |

### Algorithm (1): K Nearest Neighbors

```
In [478]: 1 knn_anxiety = KNeighborsClassifier(n_neighbors=15).fit(X_train_scaled, y_train)

In [479]: 1 accuracy_knn = round(accuracy_score(y_test, knn_anxiety.predict(X_test_scaled)), 3)
2 precision_knn = round(precision_score(y_test, knn_anxiety.predict(X_test_scaled), average='weighted'), 3)
3 recall_knn = round(recall_score(y_test, knn_anxiety.predict(X_test_scaled), average='weighted'), 3)
4 f1_score_knn = round(f1_score(y_test, knn_anxiety.predict(X_test_scaled), average='weighted'), 3)
5 cross_validation_knn = round(np.mean(cross_val_score(knn_anxiety, X_train_scaled, y_train, cv = 6)), 3)
```

```
In [480]: 1 print('Accuracy:', accuracy_knn)
2 print('Precision:', precision_knn)
3 print('Recall:', recall_knn)
4 print('F1_Score:', f1_score_knn)
5 print('Cross Validation:', cross_validation_knn)
```

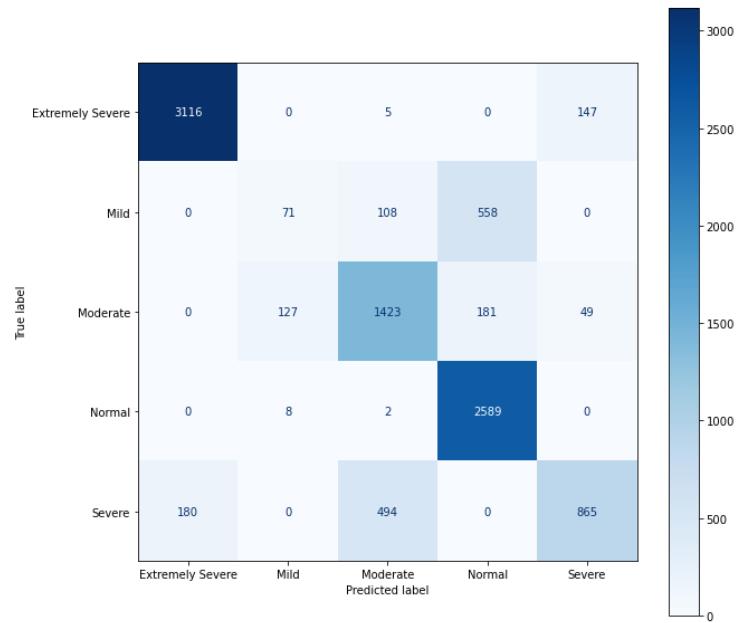
Accuracy: 0.813  
Precision: 0.793  
Recall: 0.813  
F1\_Score: 0.79  
Cross Validation: 0.804

```
In [481]: 1 classification = classification_report(digits=4, y_true=y_test, y_pred=knn_anxiety.predict(X_test_scaled))
2 print(classification)
```

|                  | precision | recall | f1-score | support |
|------------------|-----------|--------|----------|---------|
| Extremely Severe | 0.9454    | 0.9535 | 0.9494   | 3268    |
| Mild             | 0.3447    | 0.0963 | 0.1506   | 737     |
| Moderate         | 0.7003    | 0.7994 | 0.7466   | 1780    |
| Normal           | 0.7779    | 0.9962 | 0.8736   | 2599    |
| Severe           | 0.8153    | 0.5621 | 0.6654   | 1539    |
| accuracy         |           |        | 0.8127   | 9923    |
| macro avg        | 0.7167    | 0.6815 | 0.6771   | 9923    |
| weighted avg     | 0.7928    | 0.8127 | 0.7898   | 9923    |

```
In [482]: 1 fig, ax = plt.subplots(figsize=(10, 10))
2 plot_confusion_matrix(knn_anxiety, X_test_scaled, y_test, ax=ax, cmap='Blues')
```

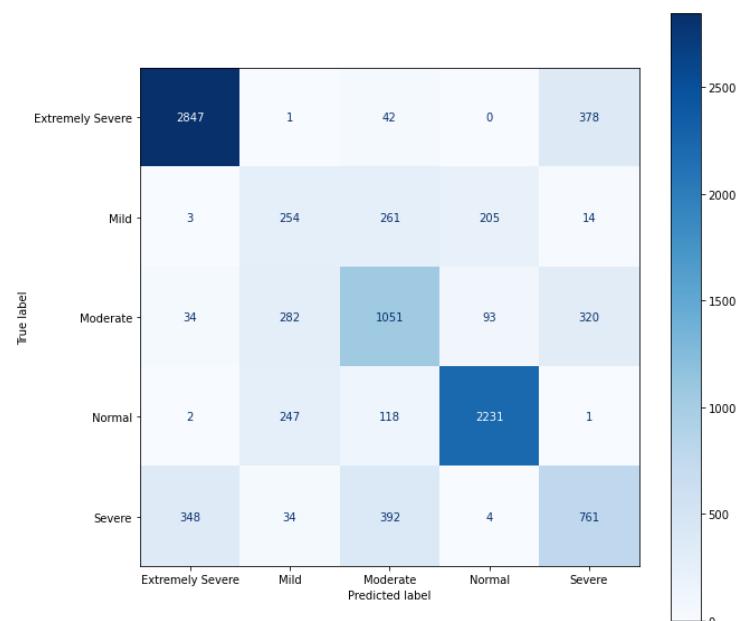
Out[482]: <sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x14baf28c130>



## Algorithm (2): Decision Tree

```
In [483]: 1 decision_tree_anxiety = DecisionTreeClassifier(criterion='entropy', splitter='best', min_samples_split=9, min_samples_leaf=2, ◀ ▶)  
  
In [484]: 1 accuracy_dt = round(accuracy_score(y_test, decision_tree_anxiety.predict(X_test_scaled)), 3)  
2 precision_dt = round(precision_score(y_test, decision_tree_anxiety.predict(X_test_scaled), average='weighted'), 3)  
3 recall_dt = round(recall_score(y_test, decision_tree_anxiety.predict(X_test_scaled), average='weighted'), 3)  
4 f1_score_dt = round(f1_score(y_test, decision_tree_anxiety.predict(X_test_scaled), average='weighted'), 3)  
5 cross_validation_dt = round(np.mean(cross_val_score(decision_tree_anxiety, X_train_scaled, y_train, cv = 6)), 3)  
  
In [485]: 1 print('Accuracy:', accuracy_dt)  
2 print('Precision:', precision_dt)  
3 print('Recall:', recall_dt)  
4 print('F1_Score:', f1_score_dt)  
5 print('Cross Validation:', cross_validation_dt)  
  
Accuracy: 0.72  
Precision: 0.725  
Recall: 0.72  
F1_Score: 0.722  
Cross Validation: 0.714  
  
In [486]: 1 classification = classification_report(digits=4, y_true=y_test, y_pred=decision_tree_anxiety.predict(X_test_scaled))  
2 print(classification)  
  
precision recall f1-score support  
  
Extremely Severe 0.8803 0.8712 0.8757 3268  
Mild 0.3105 0.3446 0.3267 737  
Moderate 0.5638 0.5904 0.5768 1780  
Normal 0.8808 0.8584 0.8694 2599  
Severe 0.5163 0.4945 0.5051 1539  
  
accuracy 0.7199 9923  
macro avg 0.6303 0.6318 0.6308 9923  
weighted avg 0.7249 0.7199 0.7222 9923
```

```
In [487]: 1 fig, ax = plt.subplots(figsize=(10, 10))  
2 plot_confusion_matrix(decision_tree_anxiety, X_test_scaled, y_test, ax=ax, cmap='Blues')  
Out[487]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x14bb7828a30>
```



## Algorithm (3): Random Forest

```
In [488]: 1 random_forest_anxiety = RandomForestClassifier(n_estimators=200, min_samples_split=3, min_samples_leaf=1, max_depth=160, max_ ◀ ▶)  
  
In [489]: 1 accuracy_rf = round(accuracy_score(y_test, random_forest_anxiety.predict(X_test_scaled)), 3)  
2 precision_rf = round(precision_score(y_test, random_forest_anxiety.predict(X_test_scaled), average='weighted'), 3)  
3 recall_rf = round(recall_score(y_test, random_forest_anxiety.predict(X_test_scaled), average='weighted'), 3)  
4 f1_Score_rf = round(f1_score(y_test, random_forest_anxiety.predict(X_test_scaled), average='weighted'), 3)  
5 cross_validation_rf = round(np.mean(cross_val_score(random_forest_anxiety, X_train_scaled, y_train, cv = 6)), 3)  
  
In [490]: 1 print('Accuracy:', accuracy_rf)  
2 print('Precision:', precision_rf)  
3 print('Recall:', recall_rf)  
4 print('F1_Score:', f1_Score_rf)  
5 print('Cross Validation:', cross_validation_rf)  
  
Accuracy: 0.852  
Precision: 0.845  
Recall: 0.852  
F1_Score: 0.826  
Cross Validation: 0.852
```

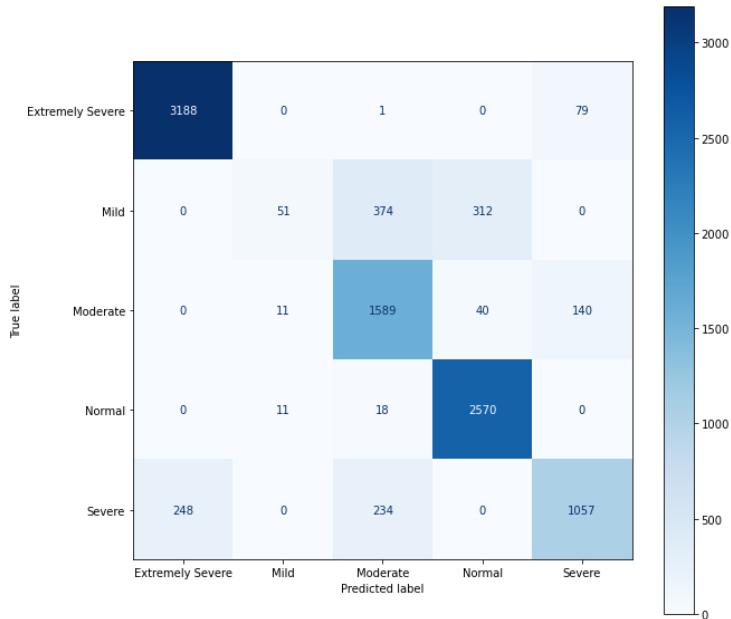
```
In [491]: 1 classification=classification_report(digits=4, y_true=y_test, y_pred=random_forest_anxiety.predict(X_test_scaled))
2 print(classification)

      precision    recall   f1-score  support

Extremely Severe    0.9278   0.9755   0.9511    3268
Mild             0.6986   0.0692   0.1259     737
Moderate          0.7171   0.8927   0.7953    1780
Normal            0.8795   0.9888   0.9310    2599
Severe            0.8284   0.6868   0.7510    1539

accuracy           0.8521
macro avg        0.8103   0.7226   0.7109    9923
weighted avg     0.8449   0.8521   0.8256    9923
```

```
In [492]: 1 fig, ax = plt.subplots(figsize=(10, 10))
2 plot_confusion_matrix(random_forest_anxiety,X_test_scaled,y_test,ax=ax, cmap='Blues')
Out[492]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x14bb6ed5820>
```



## Algorithm (4): Naive Bayes

```
In [493]: 1 naive_bayes_anxiety = GaussianNB().fit(X_train_scaled,y_train)

In [494]: 1 accuracy_nb = round(accuracy_score(y_test, naive_bayes_anxiety.predict(X_test_scaled)), 3)
2 precision_nb = round(precision_score(y_test, naive_bayes_anxiety.predict(X_test_scaled), average='weighted'), 3)
3 recall_nb = round(recall_score(y_test, naive_bayes_anxiety.predict(X_test_scaled), average='weighted'), 3)
4 f1_score_nb = round(f1_score(y_test, naive_bayes_anxiety.predict(X_test_scaled), average='weighted'), 3)
5 cross_validation_nb = round(np.mean(cross_val_score(naive_bayes_anxiety, X_train_scaled, y_train, cv = 6)), 3)
```

```
In [495]: 1 print('Accuracy:', accuracy_nb)
2 print('Precision:', precision_nb)
3 print('Recall:', recall_nb)
4 print('F1_Score:', f1_score_nb)
5 print('Cross Validation:', cross_validation_nb)
```

Accuracy: 0.812  
Precision: 0.841  
Recall 0.812  
F1\_Score: 0.821  
Cross Validation: 0.813

```
In [496]: 1 classification=classification_report(digits=4, y_true=y_test, y_pred=naive_bayes_anxiety.predict(X_test_scaled))
2 print(classification)

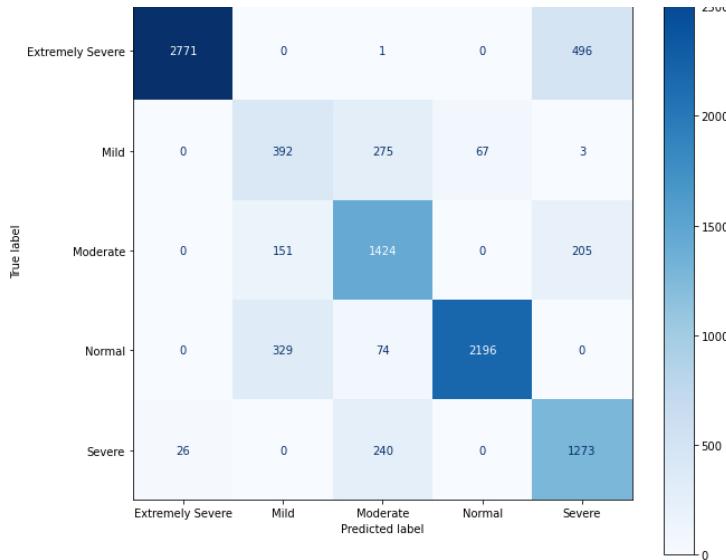
      precision    recall   f1-score  support

Extremely Severe    0.9907   0.8479   0.9138    3268
Mild             0.4495   0.5319   0.4873     737
Moderate          0.7071   0.8000   0.7507    1780
Normal            0.9704   0.8449   0.9033    2599
Severe            0.6439   0.8272   0.7241    1539

accuracy           0.8119
macro avg        0.7523   0.7704   0.7558    9923
weighted avg     0.8405   0.8119   0.8207    9923
```

```
In [497]: 1 fig, ax = plt.subplots(figsize=(10, 10))
2 plot_confusion_matrix(naive_bayes_anxiety,X_test_scaled,y_test,ax=ax, cmap='Blues')
Out[497]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x14bb85666a0>
```





### Algorithm (5): Logistic Regression

```
In [498]: 1 logistic_regression_anxiety = LogisticRegression().fit(X_train_scaled, y_train)

In [499]: 1 accuracy_lr = round(accuracy_score(y_test, logistic_regression_anxiety.predict(X_test_scaled)), 3)
2 precision_lr = round(precision_score(y_test, logistic_regression_anxiety.predict(X_test_scaled), average='weighted'), 3)
3 recall_lr = round(recall_score(y_test, logistic_regression_anxiety.predict(X_test_scaled), average='weighted'), 3)
4 f1_score_lr = round(f1_score(y_test, logistic_regression_anxiety.predict(X_test_scaled), average='weighted'), 3)
5 cross_validation_lr = round(np.mean(cross_val_score(logistic_regression_anxiety, X_train_scaled, y_train, cv=6)), 3)

In [500]: 1 print('Accuracy:', accuracy_lr)
2 print('F1_Score:', f1_score_lr)
3 print('Recall:', recall_lr)
4 print('Precision:', precision_lr)
5 print('Cross Validation:', cross_validation_lr)

Accuracy: 1.0
F1_Score: 1.0
Recall: 1.0
Precision: 1.0
Cross Validation: 0.997

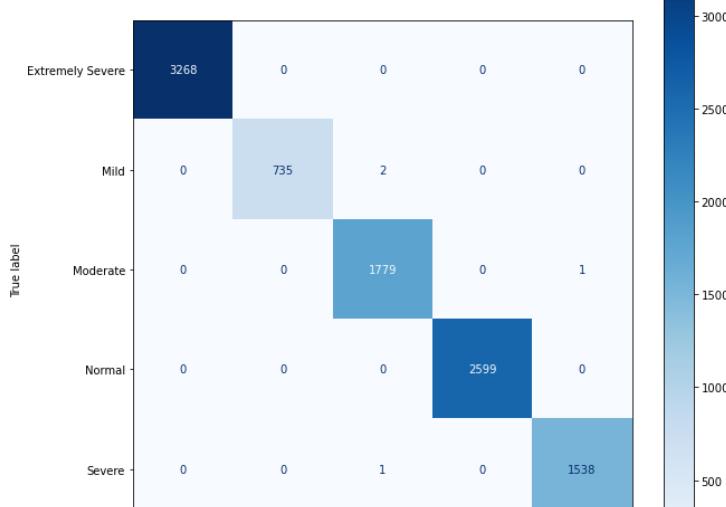
In [501]: 1 classification = classification_report(digits=4, y_true=y_test, y_pred=logistic_regression_anxiety.predict(X_test_scaled))
2 print(classification)

          precision    recall  f1-score   support

Extremely Severe    1.0000    1.0000    1.0000     3268
      Mild           1.0000    0.9973    0.9986      737
      Moderate        0.9983    0.9994    0.9989     1780
      Normal          1.0000    1.0000    1.0000     2599
      Severe          0.9994    0.9994    0.9994     1539

   accuracy         0.9996
macro avg       0.9995    0.9992    0.9994     9923
weighted avg     0.9996    0.9996    0.9996     9923
```

```
In [502]: 1 fig, ax = plt.subplots(figsize=(10, 10))
2 plot_confusion_matrix(logistic_regression_anxiety, X_test_scaled, y_test, ax=ax, cmap='Blues')
3 plt.show()
```



| Predicted label | Extremely Severe | Mild | Moderate | Normal | Severe |   |
|-----------------|------------------|------|----------|--------|--------|---|
|                 |                  |      |          |        |        | 0 |

## Algorithm (5):Support Vector Classifier

```
In [503]: 1 svm_anxiety = SVC(C=100, gamma=0.1, kernel='rbf').fit(X_train_scaled,y_train)
```

```
In [504]: 1 accuracy_svm = round(accuracy_score(y_test, svm_anxiety.predict(X_test_scaled)), 3)
2 precision_svm = round(precision_score(y_test, svm_anxiety.predict(X_test_scaled), average='weighted'), 3)
3 recall_svm = round(recall_score(y_test, svm_anxiety.predict(X_test_scaled), average='weighted'), 3)
4 f1_score_svm = round(f1_score(y_test, svm_anxiety.predict(X_test_scaled), average='weighted'), 3)
5 cross_validation_svm = round(np.mean(cross_val_score(svm_anxiety, X_train_scaled, y_train, cv = 6)), 3)
```

```
In [505]: 1 print('Accuracy:', accuracy_svm)
2 print('Precision:', precision_svm)
3 print('Recall:', recall_svm)
4 print('F1_Score:', f1_score_svm)
5 print('Cross Validation:', cross_validation_svm)
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1_Score: 1.0
Cross Validation: 1.0
```

```
In [506]: 1 classification=classification_report(digits=4, y_true=y_test, y_pred = svm_anxiety.predict(X_test_scaled))
2 print(classification)
```

|                  | precision | recall | f1-score | support |
|------------------|-----------|--------|----------|---------|
| Extremely Severe | 1.0000    | 1.0000 | 1.0000   | 3268    |
| Mild             | 1.0000    | 1.0000 | 1.0000   | 737     |
| Moderate         | 1.0000    | 1.0000 | 1.0000   | 1780    |
| Normal           | 1.0000    | 1.0000 | 1.0000   | 2599    |
| Severe           | 1.0000    | 1.0000 | 1.0000   | 1539    |
| accuracy         |           |        | 1.0000   | 9923    |
| macro avg        | 1.0000    | 1.0000 | 1.0000   | 9923    |
| weighted avg     | 1.0000    | 1.0000 | 1.0000   | 9923    |

```
In [507]: 1 fig, ax = plt.subplots(figsize=(6, 6))
2 plot_confusion_matrix(svm_anxiety, X_test_scaled, y_test,ax=ax, cmap='Blues')
```

```
Out[507]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x14bc063c550>
```

```
In [508]: 1 Result={'Model':['Random_Forest','Decision_Tree','Naive_Baise','K_Nearest_Neighbors','Logistic_Regression', 'SVM'],
2 'Accuracy%':[accuracy_rf*100,accuracy_dt*100,accuracy_nb*100,accuracy_knn*100,accuracy_lr*100, accuracy_svm*100],
3 'Precision%': [precision_rf*100,precision_dt*100,precision_nb*100,precision_knn*100,precision_lr*100, precision_svm*100],
4 'Recall%': [recall_rf*100,recall_dt*100,recall_nb*100,recall_knn*100,recall_lr*100, recall_svm*100],
5 'F1_Score%': [f1_Score_rf*100,f1_score_dt*100,f1_score_nb*100,f1_score_knn*100,f1_score_lr*100,f1_score_svm*100]}
```

```
In [509]: 1 Result_Depression=pd.DataFrame(Result)
2 Result_Depression.to_csv('D:/COURSES/Programming Language/python/data_scince_track/projects/NTI/Graduation_Project/Datasets/')

In [510]: 1 Result_Depression
```

```
Out[510]:
```

|   | Model               | Accuracy(%) | Precision(%) | Recall(%) | F1_Score(%) |
|---|---------------------|-------------|--------------|-----------|-------------|
| 0 | Random_Forest       | 85.20       | 84.50        | 85.20     | 82.60       |
| 1 | Decision_Tree       | 72.00       | 72.50        | 72.00     | 72.20       |
| 2 | Naive_Baise         | 81.20       | 84.10        | 81.20     | 82.10       |
| 3 | K_Nearest_Neighbors | 81.30       | 79.30        | 81.30     | 79.00       |
| 4 | Logistic_Regression | 100.00      | 100.00       | 100.00    | 100.00      |
| 5 | SVM                 | 100.00      | 100.00       | 100.00    | 100.00      |

## Test by My Self

```
In [511]: 1 Anxiety
```

```
Out[511]: Q2A Q4A Q7A Q9A Q15A Q19A Q20A Q23A Q25A Q28A Q30A Q36A Q40A Q41A TIPI1 TIPI2 TIPI3 TIPI4 TIPI5 TIPI6 TIPI7 TIPI8 TIPI9
```

|       |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |   |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|
| 0     | 2   | 0   | 3   | 2   | 1   | 0   | 1   | 0   | 0   | 0   | 2   | 0   | 3   | 3   | 1   | 1   | 7   | 4   | 6   | 4   | 6   | 1   |   |
| 1     | 1   | 3   | 3   | 3   | 3   | 3   | 3   | 3   | 3   | 3   | 3   | 3   | 3   | 3   | 2   | 5   | 3   | 6   | 5   | 5   | 5   | 6   |   |
| 2     | 0   | 2   | 2   | 2   | 2   | 0   | 2   | 2   | 1   | 1   | 2   | 1   | 1   | 1   | 2   | 5   | 6   | 5   | 3   | 2   | 6   | 3   |   |
| 3     | 0   | 0   | 1   | 0   | 1   | 0   | 1   | 0   | 1   | 0   | 0   | 0   | 0   | 0   | 0   | 7   | 6   | 4   | 5   | 3   | 2   | 6   | 3 |
| 4     | 3   | 3   | 3   | 3   | 3   | 3   | 3   | 3   | 3   | 1   | 3   | 2   | 3   | 3   | 3   | 1   | 4   | 5   | 7   | 5   | 7   | 6   | 7 |
| ...   | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |   |
| 28346 | 0   | 0   | 0   | 0   | 0   | 1   | 0   | 0   | 0   | 0   | 1   | 0   | 0   | 0   | 6   | 5   | 6   | 7   | 6   | 4   | 6   | 5   |   |
| 28347 | 3   | 3   | 3   | 2   | 3   | 3   | 3   | 2   | 2   | 2   | 2   | 2   | 2   | 4   | 5   | 5   | 7   | 4   | 6   | 4   | 7   |     |   |
| 28348 | 0   | 0   | 0   | 1   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 6   | 6   | 7   | 5   | 6   | 3   | 6   | 1   |   |

In [512]: 1 Anxiety[Anxiety['predict'] == 'Mild']

Out[512]:

|       | Q2A | Q4A | Q7A | Q9A | Q15A | Q19A | Q20A | Q23A | Q25A | Q28A | Q30A | Q36A | Q40A | Q41A | TIP1 | TIP12 | TIP13 | TIP14 | TIP15 | TIP16 | TIP17 | TIP18 | TIP19 |
|-------|-----|-----|-----|-----|------|------|------|------|------|------|------|------|------|------|------|-------|-------|-------|-------|-------|-------|-------|-------|
| 12    | 1   | 0   | 0   | 2   | 0    | 1    | 3    | 0    | 0    | 0    | 1    | 0    | 0    | 0    | 2    | 2     | 5     | 4     | 6     | 7     | 4     | 6     | 3     |
| 18    | 2   | 0   | 1   | 2   | 0    | 0    | 1    | 1    | 0    | 0    | 0    | 1    | 1    | 0    | 5    | 2     | 6     | 5     | 7     | 7     | 5     | 2     | 1     |
| 28    | 0   | 2   | 0   | 0   | 0    | 0    | 1    | 1    | 1    | 0    | 2    | 1    | 1    | 0    | 3    | 6     | 3     | 4     | 4     | 7     | 3     | 4     | 3     |
| 33    | 0   | 0   | 1   | 1   | 1    | 0    | 1    | 0    | 1    | 0    | 1    | 1    | 0    | 1    | 4    | 5     | 6     | 5     | 7     | 5     | 3     | 6     | 6     |
| 35    | 0   | 0   | 0   | 2   | 0    | 0    | 0    | 0    | 3    | 0    | 3    | 0    | 0    | 1    | 5    | 7     | 2     | 6     | 6     | 6     | 2     | 7     | 1     |
| ...   | ... | ... | ... | ... | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...   | ...   | ...   | ...   | ...   | ...   | ...   |       |
| 28265 | 1   | 0   | 0   | 2   | 1    | 0    | 1    | 0    | 0    | 1    | 0    | 1    | 1    | 0    | 6    | 3     | 5     | 7     | 5     | 5     | 7     | 6     | 1     |
| 28277 | 1   | 0   | 0   | 1   | 1    | 0    | 1    | 0    | 0    | 1    | 1    | 1    | 2    | 0    | 1    | 7     | 5     | 1     | 4     | 6     | 2     | 3     | 3     |
| 28295 | 0   | 2   | 1   | 1   | 0    | 0    | 0    | 0    | 2    | 0    | 1    | 1    | 1    | 0    | 2    | 2     | 5     | 6     | 7     | 7     | 4     | 3     | 5     |
| 28308 | 0   | 2   | 0   | 2   | 0    | 0    | 0    | 0    | 2    | 1    | 0    | 0    | 2    | 0    | 1    | 6     | 5     | 2     | 6     | 7     | 5     | 5     | 6     |
| 28316 | 2   | 0   | 0   | 1   | 0    | 0    | 1    | 0    | 0    | 0    | 2    | 0    | 3    | 0    | 4    | 4     | 6     | 5     | 2     | 5     | 4     | 6     | 4     |

2081 rows × 34 columns

In [513]: 1 Anxiety.columns

Out[513]: Index(['Q2A', 'Q4A', 'Q7A', 'Q9A', 'Q15A', 'Q19A', 'Q20A', 'Q23A', 'Q25A', 'Q28A', 'Q30A', 'Q36A', 'Q40A', 'Q41A', 'TIP1', 'TIP12', 'TIP13', 'TIP14', 'TIP15', 'TIP16', 'TIP17', 'TIP18', 'TIP19', 'education', 'urban', 'gender', 'religion', 'orientation', 'race', 'married', 'familysize', 'age\_group', 'predict'], dtype='object')

In [514]: 1 input\_data\_Severe = [2, 0, 3, 2, 1, 0, 1, 0, 0, 0, 2, 0, 3, 3, 1, 1, 7, 4, 6, 4, 6, 1, 6, 1, 1, 3, 2, 4, 5, 70, 1, 5, 2]  
2 input\_data\_scaled = scaler.transform([input\_data\_Severe])

In [515]: 1 prediction = svm\_anxiety.predict(input\_data\_scaled)  
2 print("Prediction:", prediction[0])

Prediction: Severe

In [516]: 1 input\_data\_Exremely\_Severe = [1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 5, 3, 6, 5, 5, 6, 3, 3, 3, 2, 2, 10, 1, 10, 1]  
2 input\_data\_scaled = scaler.transform([input\_data\_Exremely\_Severe])

In [517]: 1 prediction = svm\_anxiety.predict(input\_data\_scaled)  
2 print("Prediction:", prediction[0])

Prediction: Extremely Severe

In [518]: 1 input\_data\_Normal = [0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 7, 6, 4, 5, 3, 2, 6, 3, 5, 2, 4, 2, 2, 2, 60, 1, 2, 4]  
2 input\_data\_scaled = scaler.transform([input\_data\_Normal])

In [519]: 1 prediction = svm\_anxiety.predict(input\_data\_scaled)  
2 print("Prediction:", prediction[0])

Prediction: Normal

In [520]: 1 input\_data\_Moderate = [0, 1, 2, 2, 0, 1, 0, 0, 2, 0, 1, 1, 3, 1, 6, 5, 7, 3, 5, 3, 5, 3, 4, 3, 2, 2, 6, 1, 60, 1, 2, 3]  
2 input\_data\_scaled = scaler.transform([input\_data\_Moderate])

In [521]: 1 prediction = svm\_anxiety.predict(input\_data\_scaled)  
2 print("Prediction:", prediction[0])

Prediction: Moderate

In [522]: 1 input\_data\_Mild = [1, 0, 0, 2, 0, 1, 3, 0, 0, 1, 0, 0, 0, 2, 2, 5, 4, 6, 7, 4, 6, 3, 6, 2, 3, 2, 4, 2, 50, 1, 2, 3]  
2 input\_data\_scaled = scaler.transform([input\_data\_Mild])

In [523]: 1 prediction = svm\_anxiety.predict(input\_data\_scaled)  
2 print("Prediction:", prediction[0])

Prediction: Mild

In [524]: 1 column\_scales = {  
2 'Q2A': list(range(4)),  
3 'Q4A': list(range(4)),  
4 'Q7A': list(range(4)),  
5 'Q9A': list(range(4)),  
6 'Q15A': list(range(4)),  
7 'Q19A': list(range(4)),  
8 'Q20A': list(range(4))}

```

9   'Q23A': list(range(4)),
10  'Q25A': list(range(4)),
11  'Q28A': list(range(4)),
12  'Q30A': list(range(4)),
13  'Q36A': list(range(4)),
14  'Q40A': list(range(4)),
15  'Q41A': list(range(4)),
16  'TIPI1': list(range(1, 8)),
17  'TIPI2': list(range(1, 8)),
18  'TIPI3': list(range(1, 8)),
19  'TIPI4': list(range(1, 8)),
20  'TIPI5': list(range(1, 8)),
21  'TIPI6': list(range(1, 8)),
22  'TIPI7': list(range(1, 8)),
23  'TIPI8': list(range(1, 8)),
24  'TIPI9': list(range(1, 8)),
25  'TIPI10': list(range(1, 8)),
26  'education': list(range(1, 5)),
27  'urban': list(range(1, 4)),
28  'gender': list(range(1, 4)),
29  'religion': list(range(1, 13)),
30  'orientation': list(range(1, 6)),
31  'race': [10, 20, 30, 40, 50, 60, 70],
32  'married': list(range(1, 4)),
33  'familysize': list(range(1, 101)),
34  'age_group': list(range(1, 7))
35 }

```

```

In [525]: 1 random_lists = []
2 for i in range(30):
3     random_list = [
4         random.choice(column_scales[column]) for column in [
5             'Q2A', 'Q4A', 'Q7A', 'Q9A', 'Q15A', 'Q19A', 'Q20A', 'Q23A', 'Q25A',
6             'Q28A', 'Q30A', 'Q36A', 'Q40A', 'Q41A', 'TIPI1', 'TIPI2', 'TIPI3',
7             'TIPI4', 'TIPI5', 'TIPI6', 'TIPI7', 'TIPI8', 'TIPI9', 'TIPI10',
8             'education', 'urban', 'gender', 'religion', 'orientation', 'race',
9             'married', 'familysize', 'age_group'
10        ]
11    ]
12    random_lists.append(random_list)

```

```

In [526]: 1 for i, random_list in enumerate(random_lists, start=1):
2     print(f"random_values{i} = {random_list}")
3
4 random_values1 = [0, 2, 0, 2, 1, 2, 1, 0, 3, 2, 2, 2, 3, 0, 1, 6, 2, 6, 3, 1, 5, 2, 5, 6, 4, 3, 1, 2, 5, 10, 3, 4, 6]
5 random_values2 = [2, 1, 2, 2, 1, 1, 3, 2, 2, 2, 1, 1, 1, 3, 5, 4, 3, 2, 1, 2, 6, 3, 2, 2, 10, 1, 50, 3, 29, 4]
6 random_values3 = [3, 2, 0, 2, 0, 2, 3, 0, 1, 2, 1, 2, 7, 1, 5, 5, 5, 4, 1, 7, 2, 2, 1, 3, 2, 7, 2, 50, 2, 30, 5]
7 random_values4 = [1, 0, 1, 0, 1, 3, 2, 2, 3, 3, 1, 1, 3, 2, 5, 4, 1, 3, 5, 1, 7, 1, 3, 1, 4, 2, 2, 12, 3, 30, 1, 66, 1]
8 random_values5 = [3, 3, 2, 1, 2, 2, 1, 1, 1, 0, 2, 2, 0, 0, 2, 4, 5, 3, 6, 6, 7, 5, 5, 3, 3, 2, 2, 11, 3, 30, 1, 38, 3]
9 random_values6 = [2, 1, 3, 3, 0, 1, 0, 1, 2, 2, 0, 3, 1, 1, 7, 5, 1, 4, 2, 3, 6, 4, 4, 7, 1, 3, 3, 6, 3, 20, 1, 10, 5]
10 random_values7 = [0, 2, 0, 2, 1, 3, 3, 0, 1, 2, 0, 0, 2, 2, 2, 4, 3, 4, 6, 3, 1, 7, 4, 7, 2, 3, 3, 4, 2, 40, 1, 73, 6]
11 random_values8 = [2, 0, 0, 1, 0, 2, 2, 2, 1, 0, 3, 1, 3, 6, 2, 2, 6, 4, 1, 7, 7, 2, 4, 3, 2, 2, 6, 4, 50, 3, 34, 6]
12 random_values9 = [3, 0, 3, 1, 3, 3, 0, 0, 0, 2, 1, 2, 5, 2, 4, 3, 4, 7, 7, 5, 1, 3, 1, 2, 8, 5, 70, 1, 73, 2]
13 random_values10 = [3, 1, 1, 1, 3, 0, 1, 0, 3, 0, 2, 2, 3, 2, 7, 1, 4, 7, 5, 1, 1, 1, 1, 3, 2, 3, 1, 5, 2, 30, 2, 81, 3]
14 random_values11 = [1, 2, 2, 1, 0, 0, 3, 2, 1, 3, 3, 2, 0, 3, 5, 2, 1, 6, 4, 6, 3, 2, 4, 3, 2, 2, 3, 2, 4, 60, 2, 10, 3]
15 random_values12 = [2, 2, 0, 0, 0, 3, 2, 2, 3, 2, 0, 2, 4, 1, 3, 2, 3, 3, 4, 1, 1, 4, 3, 3, 1, 5, 70, 1, 20, 3]
16 random_values13 = [3, 2, 0, 0, 3, 2, 0, 1, 3, 3, 1, 2, 0, 1, 3, 7, 4, 6, 7, 7, 6, 3, 7, 4, 1, 2, 3, 10, 1, 70, 2, 73, 5]
17 random_values14 = [2, 3, 0, 2, 0, 0, 3, 3, 1, 0, 2, 3, 3, 6, 7, 7, 5, 5, 6, 3, 6, 4, 2, 3, 2, 2, 9, 5, 30, 1, 25, 2]
18 random_values15 = [3, 2, 3, 3, 2, 2, 0, 0, 2, 0, 0, 2, 3, 2, 2, 7, 6, 1, 1, 1, 7, 5, 6, 2, 2, 3, 4, 2, 60, 3, 86, 6]
19 random_values16 = [0, 0, 3, 2, 1, 0, 3, 3, 0, 1, 1, 1, 2, 3, 4, 5, 6, 7, 2, 3, 3, 7, 2, 2, 2, 3, 10, 1, 50, 1, 43, 3]
20 random_values17 = [3, 1, 3, 1, 0, 3, 2, 0, 0, 0, 1, 3, 3, 4, 7, 2, 4, 2, 7, 3, 5, 7, 7, 2, 1, 2, 3, 3, 40, 3, 3, 2]
21 random_values18 = [2, 2, 2, 1, 2, 1, 0, 0, 3, 2, 0, 0, 2, 2, 2, 6, 7, 2, 4, 3, 2, 4, 7, 5, 1, 1, 2, 5, 4, 20, 3, 67, 1]
22 random_values19 = [2, 0, 1, 1, 3, 2, 1, 0, 1, 0, 0, 3, 2, 1, 1, 3, 7, 3, 5, 1, 4, 1, 1, 6, 1, 2, 3, 6, 1, 70, 2, 36, 1]
23 random_values20 = [1, 3, 0, 1, 3, 2, 1, 3, 3, 0, 2, 3, 1, 2, 1, 1, 5, 1, 4, 6, 3, 3, 2, 2, 3, 3, 2, 4, 4, 50, 1, 88, 4]
24 random_values21 = [2, 0, 2, 0, 2, 0, 3, 3, 2, 0, 0, 2, 3, 2, 2, 7, 6, 1, 1, 1, 7, 5, 6, 2, 2, 3, 4, 2, 60, 3, 47, 5]
25 random_values22 = [0, 0, 3, 0, 0, 3, 2, 3, 2, 0, 2, 2, 3, 3, 2, 1, 3, 3, 4, 5, 6, 7, 2, 3, 2, 2, 3, 10, 1, 50, 1, 43, 3]
26 random_values23 = [1, 3, 3, 3, 0, 0, 1, 0, 0, 2, 0, 1, 2, 2, 4, 1, 1, 2, 6, 2, 4, 5, 4, 5, 1, 2, 2, 12, 1, 70, 2, 3, 2]
27 random_values24 = [0, 1, 3, 3, 1, 0, 2, 0, 0, 1, 1, 2, 3, 2, 1, 4, 6, 4, 5, 3, 7, 7, 4, 3, 2, 2, 5, 1, 50, 2, 24, 1]
28 random_values25 = [2, 1, 0, 2, 0, 3, 2, 2, 1, 0, 1, 0, 1, 4, 4, 1, 5, 6, 3, 2, 5, 4, 7, 3, 1, 2, 6, 2, 20, 2, 4, 5]
29 random_values26 = [2, 2, 1, 3, 2, 1, 0, 2, 3, 0, 2, 1, 3, 2, 1, 6, 5, 6, 4, 6, 5, 7, 1, 6, 3, 3, 1, 9, 1, 20, 1, 63, 6]
30 random_values27 = [1, 2, 2, 3, 3, 3, 2, 3, 1, 2, 0, 1, 0, 2, 5, 2, 6, 3, 5, 4, 4, 3, 2, 5, 1, 3, 1, 2, 4, 50, 3, 67, 5]
31 random_values28 = [3, 3, 2, 0, 0, 3, 2, 3, 0, 3, 2, 2, 1, 2, 4, 6, 6, 5, 5, 2, 1, 4, 5, 7, 6, 3, 1, 1, 4, 1, 40, 1, 54, 1]
32 random_values29 = [1, 0, 2, 0, 1, 3, 3, 0, 1, 2, 0, 0, 2, 1, 7, 6, 2, 4, 1, 2, 1, 5, 5, 6, 4, 2, 1, 2, 4, 20, 2, 53, 3]
33 random_values30 = [1, 2, 2, 0, 3, 2, 0, 3, 1, 0, 3, 0, 7, 4, 6, 3, 5, 7, 6, 6, 7, 2, 2, 4, 4, 60, 2, 69, 5]

```

```

In [527]: 1 random_values1 = [0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 7, 6, 4, 5, 3, 2, 6, 3, 5, 2, 4, 2, 2, 2, 2, 60, 1, 2, 4]
2 random_values2 = [3, 1, 3, 0, 3, 2, 2, 2, 2, 2, 3, 2, 0, 4, 1, 3, 5, 5, 1, 4, 1, 1, 7, 2, 1, 1, 10, 1, 40, 3, 58, 1]
3 random_values3 = [1, 0, 2, 2, 0, 2, 3, 2, 2, 0, 1, 3, 2, 1, 2, 2, 1, 3, 5, 5, 1, 5, 3, 6, 3, 2, 2, 3, 4, 40, 1, 22, 6]
4 random_values4 = [1, 2, 0, 2, 1, 3, 2, 1, 0, 3, 3, 1, 1, 4, 3, 4, 5, 6, 3, 2, 4, 2, 2, 40, 2, 62, 4]
5 random_values5 = [2, 0, 3, 1, 0, 0, 3, 3, 2, 2, 2, 1, 0, 1, 3, 4, 3, 6, 1, 1, 2, 4, 7, 4, 3, 1, 2, 2, 2, 30, 1, 11, 3]
6 random_values6 = [0, 3, 0, 3, 3, 1, 1, 3, 0, 1, 0, 1, 1, 2, 6, 2, 2, 4, 2, 3, 1, 6, 4, 2, 1, 1, 2, 60, 3, 98, 3]
7 random_values7 = [1, 2, 1, 0, 0, 1, 1, 2, 2, 1, 2, 2, 3, 2, 6, 2, 1, 6, 5, 7, 4, 4, 7, 4, 1, 2, 3, 4, 40, 2, 75, 1]
8 random_values8 = [1, 0, 2, 2, 3, 3, 0, 1, 3, 0, 2, 1, 1, 1, 7, 3, 4, 7, 4, 3, 6, 1, 1, 1, 3, 10, 3, 60, 3, 47, 6]
9 random_values9 = [2, 1, 2, 1, 0, 1, 2, 1, 2, 1, 0, 2, 1, 3, 2, 4, 6, 5, 7, 1, 3, 3, 1, 4, 3, 3, 6, 2, 50, 2, 51, 5]
10 random_values10 = [1, 0, 0, 0, 2, 2, 1, 0, 2, 2, 0, 3, 3, 2, 6, 5, 5, 4, 3, 5, 1, 2, 7, 7, 4, 2, 3, 7, 2, 50, 1, 8, 4]
11 random_values11 = [1, 0, 0, 0, 1, 0, 2, 0, 1, 0, 1, 0, 6, 2, 7, 6, 1, 7, 5, 1, 2, 2, 2, 1, 2, 2, 5, 70, 2, 5, 2]
12 random_values12 = [1, 2, 0, 2, 0, 0, 1, 0, 3, 2, 2, 0, 1, 0, 2, 5, 4, 3, 4, 7, 4, 6, 4, 2, 1, 3, 2, 1, 6, 4, 50, 2, 4, 2]
13 random_values13 = [2, 0, 3, 1, 0, 2, 3, 1, 0, 3, 1, 0, 3, 3, 1, 2, 7, 5, 6, 7, 6, 3, 5, 3, 3, 3, 1, 1, 60, 1, 56, 6]
14 random_values14 = [1, 1, 2, 1, 3, 2, 3, 1, 2, 1, 0, 3, 2, 1, 5, 3, 4, 7, 1, 6, 2, 1, 6, 5, 2, 1, 2, 8, 1, 20, 1, 90, 4]
15 random_values15 = [2, 1, 0, 2, 3, 2, 0, 2, 3, 1, 0, 2, 3, 2, 3, 3, 2, 1, 2, 2, 4, 7, 6, 3, 2, 1, 9, 3, 10, 2, 37, 6]
16 random_values16 = [2, 0, 3, 2, 0, 0, 3, 0, 3, 2, 3, 0, 2, 3, 7, 1, 6, 2, 5, 5, 1, 2, 1, 1, 3, 18, 4, 50, 2, 8, 1]
17 random_values17 = [2, 0, 2, 3, 3, 2, 2, 2, 2, 1, 2, 0, 2, 6, 6, 7, 2, 1, 1, 7, 7, 5, 4, 2, 1, 2, 8, 3, 70, 3, 61, 5]
18 random_values18 = [1, 2, 2, 2, 1, 3, 2, 2, 0, 1, 0, 2, 1, 0, 1, 3, 3, 1, 5, 5, 2, 5, 7, 1, 2, 2, 1, 3, 70, 3, 27, 1]
19 random_values19 = [3, 1, 1, 3, 3, 2, 3, 1, 1, 0, 1, 0, 1, 5, 1, 3, 3, 4, 7, 7, 4, 6, 4, 1, 3, 7, 2, 20, 3, 95, 6]
20 random_values20 = [2, 1, 1, 0, 0, 3, 1, 1, 0, 2, 0, 3, 1, 0, 3, 6, 3, 3, 2, 2, 3, 2, 1, 2, 2, 7, 1, 70, 1, 66, 3]
21 random_values21 = [3, 1, 0, 0, 2, 0, 2, 0, 2, 0, 2, 3, 3, 6, 6, 7, 4, 3, 1, 1, 5, 2, 3, 3, 3, 5, 3, 40, 2, 17, 2]
22 random_values22 = [2, 1, 0, 0, 3, 1, 3, 3, 0, 3, 1, 0, 2, 0, 2, 4, 4, 4, 5, 6, 4, 2, 1, 3, 3, 5, 20, 3, 37, 1]
23 random_values23 = [1, 3, 1, 1, 0, 1, 3, 0, 3, 0, 0, 2, 2, 4, 6, 2, 5, 2, 3, 7, 6, 1, 2, 3, 1, 1, 2, 60, 1, 26, 5]
24 random_values24 = [0, 1, 1, 2, 3, 2, 2, 0, 2, 1, 0, 0, 2, 3, 4, 5, 6, 4, 3, 2, 3, 3, 7, 1, 1, 1, 3, 4, 30, 3, 2, 2]
25 random_values25 = [2, 0, 2, 3, 2, 0, 0, 2, 3, 4, 2, 3, 4, 6, 5, 7, 1, 2, 1, 3, 2, 5, 1, 3, 2, 5, 10, 1, 60, 4]
26 random_values26 = [0, 1, 0, 3, 0, 1, 1, 2, 0, 2, 3, 1, 0, 6, 5, 3, 6, 2, 2, 6, 2, 3, 1, 1, 3, 5, 1, 20, 1, 73, 5]
27 random_values27 = [0, 3, 2, 3, 2, 3, 1, 3, 2, 0, 3, 1, 0, 2, 4, 6, 6, 5, 2, 1, 4, 5, 7, 6, 3, 1, 1, 4, 1, 40, 1, 69, 3]
28 random_values28 = [2, 2, 3, 3, 1, 0, 2, 2, 0, 3, 0, 2, 3, 1, 4, 4, 4, 7, 4, 1, 1, 3, 4, 5, 1, 3, 3, 2, 1, 60, 1, 52, 5]
29 random_values29 = [0, 1, 0, 1, 3, 0, 2, 3, 0, 2, 1, 0, 3, 0, 7, 4, 6, 3, 5, 7, 6, 6, 7, 2, 2, 4, 4, 60, 2, 69, 5]

```

```
30 random_values30 = [0, 0, 0, 0, 2, 2, 1, 0, 0, 0, 0, 0, 0, 0, 5, 1, 1, 1, 7, 1, 6, 1, 1, 1, 1, 4, 2, 3, 3, 1, 60, 1, 4, 1]

In [528]: 1 input_data_scaled = scaler.transform([random_values1])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Normal

In [529]: 1 input_data_scaled = scaler.transform([random_values2])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Extremely Severe

In [530]: 1 input_data_scaled = scaler.transform([random_values3])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Extremely Severe

In [531]: 1 input_data_scaled = scaler.transform([random_values4])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Extremely Severe

In [532]: 1 input_data_scaled = scaler.transform([random_values5])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Extremely Severe

In [533]: 1 input_data_scaled = scaler.transform([random_values6])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Severe

In [534]: 1 input_data_scaled = scaler.transform([random_values7])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Extremely Severe

In [535]: 1 input_data_scaled = scaler.transform([random_values8])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Extremely Severe

In [536]: 1 input_data_scaled = scaler.transform([random_values9])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Severe

In [537]: 1 input_data_scaled = scaler.transform([random_values10])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Severe

In [538]: 1 input_data_scaled = scaler.transform([random_values11])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Normal

In [539]: 1 input_data_scaled = scaler.transform([random_values12])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Severe

In [540]: 1 input_data_scaled = scaler.transform([random_values13])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Severe

In [541]: 1 input_data_scaled = scaler.transform([random_values14])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Extremely Severe

In [542]: 1 input_data_scaled = scaler.transform([random_values15])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Extremely Severe

In [543]: 1 input_data_scaled = scaler.transform([random_values15])
2 prediction = svm_depression.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Severe

In [544]: 1 input_data_scaled = scaler.transform([random_values16])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])
```

```

Prediction: Extremely Severe

In [545]: 1 input_data_scaled = scaler.transform([random_values17])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Extremely Severe

In [546]: 1 input_data_scaled = scaler.transform([random_values18])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Severe

In [547]: 1 input_data_scaled = scaler.transform([random_values19])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Extremely Severe

In [548]: 1 input_data_scaled = scaler.transform([random_values20])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Severe

In [549]: 1 input_data_scaled = scaler.transform([random_values21])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Extremely Severe

In [550]: 1 input_data_scaled = scaler.transform([random_values22])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Severe

In [551]: 1 input_data_scaled = scaler.transform([random_values23])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Severe

In [552]: 1 input_data_scaled = scaler.transform([random_values24])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Severe

In [553]: 1 input_data_scaled = scaler.transform([random_values25])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Extremely Severe

In [554]: 1 input_data_scaled = scaler.transform([random_values26])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Severe

In [555]: 1 input_data_scaled = scaler.transform([random_values27])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Extremely Severe

In [556]: 1 input_data_scaled = scaler.transform([random_values28])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Extremely Severe

In [557]: 1 input_data_scaled = scaler.transform([random_values29])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Severe

In [558]: 1 input_data_scaled = scaler.transform([random_values30])
2 prediction = svm_anxiety.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Normal

```

### 3.1 Model for Stress

```

In [559]: 1 Stress.columns

Out[559]: Index(['Q1A', 'Q6A', 'Q8A', 'Q11A', 'Q12A', 'Q14A', 'Q18A', 'Q22A', 'Q27A',
   'Q29A', 'Q32A', 'Q33A', 'Q35A', 'Q39A', 'TIPI1', 'TIPI2', 'TIPI3',
   'TIPI4', 'TIPI5', 'TIPI6', 'TIPI7', 'TIPI8', 'TIPI9', 'TIPI10']).

```

```

'education', 'urban', 'gender', 'religion', 'orientation', 'race',
'married', 'familysize', 'age_group', 'predict'],
dtype='object')

In [560]: 1 X = Stress[['Q1A', 'Q6A', 'Q11A', 'Q12A', 'Q14A', 'Q18A', 'Q22A',
2 'Q27A', 'Q29A', 'Q32A', 'Q33A', 'Q35A', 'Q39A', 'TIPI1', 'TIPI2', 'TIPI3',
3 'TIPI4', 'TIPI5', 'TIPI6', 'TIPI7', 'TIPI8', 'TIPI9', 'TIPI10',
4 'education', 'urban', 'gender', 'religion', 'orientation', 'race',
5 'married', 'familysize', 'age_group']]

In [561]: 1 y=Stress[['predict']].values
2 y = y.flatten()

In [562]: 1 X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.35,random_state=0)
2 X_train_scaled = scaler.fit_transform(X_train)
3 X_test_scaled = scaler.transform(X_test)

In [563]: 1 print('Training Set:',X_train.shape,y_train.shape)
2 print('Test Set:',X_test.shape,y_test.shape)

Training Set: (18428, 33) (18428),
Test Set: (9923, 33) (9923,)

In [564]: 1 clf = LazyClassifier(verbose=0,ignore_warnings=True, custom_metric=None)
2 models, predictions = clf.fit(X_train, X_test, y_train, y_test)
3 print(models)

97%|██████████| 28/29 [09:53<00:07, 7.31s/it]

[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.004405 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 217
[LightGBM] [Info] Number of data points in the train set: 18428, number of used features: 33
[LightGBM] [Info] Start training from score -2.090134
[LightGBM] [Info] Start training from score -2.051826
[LightGBM] [Info] Start training from score -1.513181
[LightGBM] [Info] Start training from score -1.124281
[LightGBM] [Info] Start training from score -1.595856

100%|██████████| 29/29 [09:56<00:00, 20.58s/it]

          Accuracy  Balanced Accuracy  ROC AUC  F1 Score  \
Model
LogisticRegression      1.00        1.00    None   1.00
SVC                     0.98        0.97    None   0.98
LinearDiscriminantAnalysis 0.94        0.94    None   0.94
NuSVC                   0.95        0.93    None   0.95
QuadraticDiscriminantAnalysis 0.94        0.93    None   0.94
LGBMClassifier          0.93        0.92    None   0.93
ExtraTreesClassifier    0.90        0.87    None   0.90
NearestCentroid          0.84        0.86    None   0.85
RandomForestClassifier   0.89        0.85    None   0.88
GaussianNB               0.85        0.85    None   0.86
BaggingClassifier         0.82        0.79    None   0.82
KNeighborsClassifier     0.79        0.77    None   0.79
CalibratedClassifierCV   0.83        0.76    None   0.79
LinearSVC                 0.80        0.74    None   0.76
BernoulliNB                0.76        0.73    None   0.76
AdaBoostClassifier        0.71        0.71    None   0.72
DecisionTreeClassifier    0.73        0.70    None   0.73
LabelSpreading             0.72        0.69    None   0.72
LabelPropagation           0.72        0.69    None   0.72
SGDClassifier              0.72        0.67    None   0.71
ExtraTreeClassifier        0.69        0.66    None   0.69
Perceptron                  0.70        0.65    None   0.69
PassiveAggressiveClassifier 0.69        0.65    None   0.69
RidgeClassifier             0.61        0.51    None   0.54
RidgeClassifierCV           0.61        0.51    None   0.54
DummyClassifier              0.32        0.20    None   0.16

          Time Taken
Model
LogisticRegression       2.74
SVC                      31.01
LinearDiscriminantAnalysis 1.01
NuSVC                    123.28
QuadraticDiscriminantAnalysis 0.85
LGBMClassifier            3.35
ExtraTreesClassifier      5.71
NearestCentroid            0.33
RandomForestClassifier     6.84
GaussianNB                 0.34
BaggingClassifier           2.86
KNeighborsClassifier        7.29
CalibratedClassifierCV     82.64
LinearSVC                   20.20
BernoulliNB                  0.43
AdaBoostClassifier           5.06
DecisionTreeClassifier       0.64
LabelSpreading                 179.45
LabelPropagation                 116.20
SGDClassifier                   1.38
ExtraTreeClassifier                 0.41
Perceptron                      0.68
PassiveAggressiveClassifier      1.13
RidgeClassifier                  0.85
RidgeClassifierCV                  0.82
DummyClassifier                     0.25

```

## Algorithm (1): K Nearest Neighbors

```
In [565]: 1 knn_stress = KNeighborsClassifier(n_neighbors=15).fit(X_train_scaled, y_train)

In [566]: 1 accuracy_knn = round(accuracy_score(y_test, knn_stress.predict(X_test_scaled)), 3)
2 precision_knn = round(precision_score(y_test, knn_stress.predict(X_test_scaled), average='weighted'), 3)
3 recall_knn = round(recall_score(y_test, knn_stress.predict(X_test_scaled), average='weighted'), 3)
4 f1_score_knn = round(f1_score(y_test, knn_stress.predict(X_test_scaled), average='weighted'), 3)
5 cross_validation_knn = round(np.mean(cross_val_score(knn_stress, X_train_scaled, y_train, cv = 6)), 3)
```

```
In [567]: 1 print('Accuracy:', accuracy_knn)
2 print('Precision:', precision_knn)
3 print('Recall:', recall_knn)
4 print('F1_Score:', f1_score_knn)
5 print('Cross Validation:', cross_validation_knn)
```

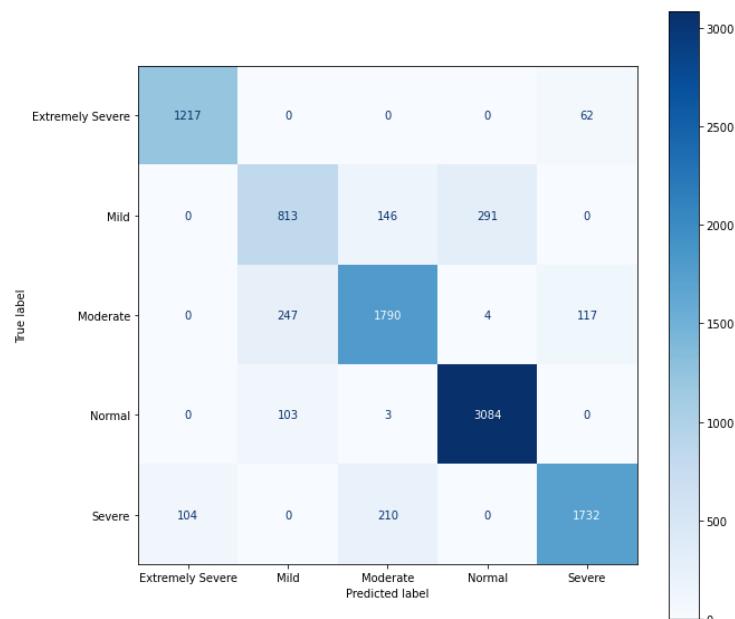
Accuracy: 0.87  
Precision: 0.868  
Recall: 0.87  
F1\_Score: 0.869  
Cross Validation: 0.859

```
In [568]: 1 classification = classification_report(digits=4, y_true=y_test, y_pred=knn_stress.predict(X_test_scaled))
2 print(classification)
```

|                  | precision | recall | f1-score | support |
|------------------|-----------|--------|----------|---------|
| Extremely Severe | 0.9213    | 0.9515 | 0.9362   | 1279    |
| Mild             | 0.6991    | 0.6504 | 0.6738   | 1250    |
| Moderate         | 0.8329    | 0.8295 | 0.8312   | 2158    |
| Normal           | 0.9127    | 0.9668 | 0.9390   | 3190    |
| Severe           | 0.9063    | 0.8465 | 0.8754   | 2046    |
| accuracy         |           |        | 0.8703   | 9923    |
| macro avg        | 0.8545    | 0.8489 | 0.8511   | 9923    |
| weighted avg     | 0.8682    | 0.8703 | 0.8687   | 9923    |

```
In [569]: 1 fig, ax = plt.subplots(figsize=(10, 10))
2 plot_confusion_matrix(knn_stress, X_test_scaled, y_test, ax=ax, cmap='Blues')
```

Out[569]: <sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x14bc18c3340>



## Algorithm (2): Decision Tree

```
In [570]: 1 decision_tree_stress = DecisionTreeClassifier(criterion='entropy', splitter='best', min_samples_split=9, min_samples_leaf=2,
```

```
In [571]: 1 accuracy_dt = round(accuracy_score(y_test, decision_tree_stress.predict(X_test_scaled)), 3)
2 precision_dt = round(precision_score(y_test, decision_tree_stress.predict(X_test_scaled), average='weighted'), 3)
3 recall_dt = round(recall_score(y_test, decision_tree_stress.predict(X_test_scaled), average='weighted'), 3)
4 f1_score_dt = round(f1_score(y_test, decision_tree_stress.predict(X_test_scaled), average='weighted'), 3)
5 cross_validation_dt = round(np.mean(cross_val_score(decision_tree_stress, X_train_scaled, y_train, cv = 6)), 3)
```

```
In [572]: 1 print('Accuracy:', accuracy_dt)
2 print('Precision:', precision_dt)
3 print('Recall:', recall_dt)
4 print('F1_Score:', f1_score_dt)
5 print('Cross Validation:', cross_validation_dt)
```

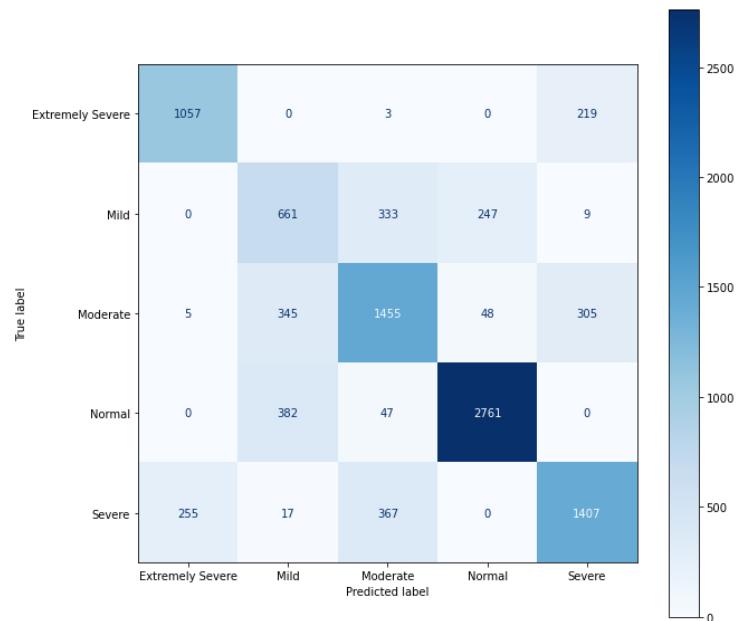
Accuracy: 0.74  
Precision: 0.746  
Recall: 0.74  
F1\_Score: 0.743  
Cross Validation: 0.735

```
In [573]: 1 classification = classification_report(digits=4, y_true=y_test, y_pred=decision_tree_stress.predict(X_test_scaled))
2 print(classification)
```

|  | precision | recall | f1-score | support |
|--|-----------|--------|----------|---------|
|--|-----------|--------|----------|---------|

|                  | precision | recall | f1-score | support |
|------------------|-----------|--------|----------|---------|
| Extremely Severe | 0.8026    | 0.8264 | 0.8143   | 1279    |
| Mild             | 0.4705    | 0.5288 | 0.4979   | 1250    |
| Moderate         | 0.6599    | 0.6742 | 0.6670   | 2158    |
| Normal           | 0.9035    | 0.8655 | 0.8841   | 3190    |
| Severe           | 0.7253    | 0.6877 | 0.7060   | 2046    |
| accuracy         |           |        | 0.7398   | 9923    |
| macro avg        | 0.7123    | 0.7165 | 0.7139   | 9923    |
| weighted avg     | 0.7462    | 0.7398 | 0.7425   | 9923    |

```
In [574]: 1 fig, ax = plt.subplots(figsize=(10, 10))
2 plot_confusion_matrix(decision_tree_stress, X_test_scaled, y_test, ax=ax, cmap='Blues')
Out[574]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x14baf545d00>
```



### Algorithm (3): Random Forest

```
In [575]: 1 random_forest_stress = RandomForestClassifier(n_estimators=200, min_samples_split=3, min_samples_leaf=1, max_depth=160, max_
   ↓
   ↑
```

```
In [576]: 1 accuracy_rf = round(accuracy_score(y_test, random_forest_stress.predict(X_test_scaled)), 3)
2 precision_rf = round(precision_score(y_test, random_forest_stress.predict(X_test_scaled), average='weighted'), 3)
3 recall_rf = round(recall_score(y_test, random_forest_stress.predict(X_test_scaled), average='weighted'), 3)
4 f1_Score_rf = round(f1_score(y_test, random_forest_stress.predict(X_test_scaled), average='weighted'), 3)
5 cross_validation_rf = round(np.mean(cross_val_score(random_forest_stress, X_train_scaled, y_train, cv = 6)),3)
6
```

```
In [577]: 1 print('Accuracy:', accuracy_rf)
2 print('Precision:', precision_rf)
3 print('Recall:', recall_rf)
4 print('F1_Score:', f1_Score_rf)
5 print('Cross Validation:', cross_validation_rf)
```

Accuracy: 0.891  
Precision: 0.892  
Recall: 0.891  
F1\_Score: 0.887  
Cross Validation: 0.885

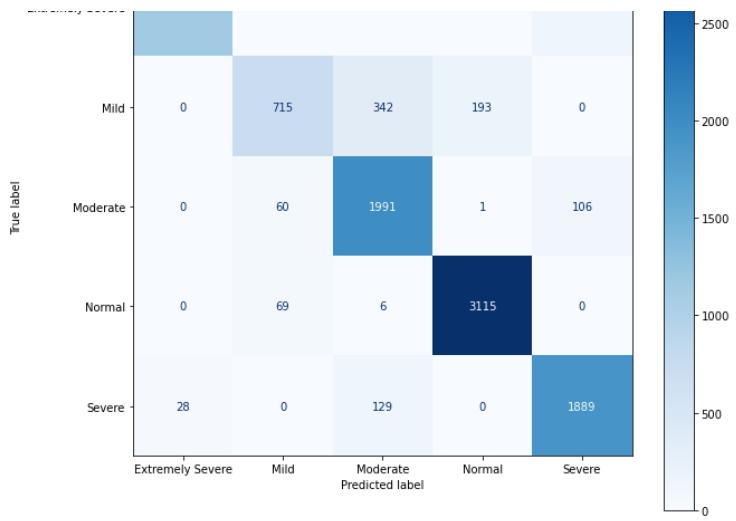
```
In [578]: 1 classification=classification_report(digits=4, y_true=y_test, y_pred=random_forest_stress.predict(X_test_scaled))
2 print(classification)
```

|                  | precision | recall | f1-score | support |
|------------------|-----------|--------|----------|---------|
| Extremely Severe | 0.9759    | 0.8858 | 0.9287   | 1279    |
| Mild             | 0.8472    | 0.5720 | 0.6829   | 1250    |
| Moderate         | 0.8067    | 0.9226 | 0.8608   | 2158    |
| Normal           | 0.9414    | 0.9765 | 0.9586   | 3190    |
| Severe           | 0.8823    | 0.9233 | 0.9023   | 2046    |
| accuracy         |           |        | 0.8912   | 9923    |
| macro avg        | 0.8907    | 0.8560 | 0.8667   | 9923    |
| weighted avg     | 0.8925    | 0.8912 | 0.8871   | 9923    |

```
In [579]: 1 fig, ax = plt.subplots(figsize=(10, 10))
2 plot_confusion_matrix(random_forest_stress, X_test_scaled, y_test, ax=ax, cmap='Blues')
```

```
Out[579]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x14baf345040>
```





### Algorithm (4): Naive Bayes

```
In [580]: 1 naive_bayse_stress = GaussianNB().fit(X_train_scaled, y_train)

In [581]: 1 accuracy_nb = round(accuracy_score(y_test, naive_bayse_stress.predict(X_test_scaled)), 3)
2 precision_nb = round(precision_score(y_test, naive_bayse_stress.predict(X_test_scaled), average='weighted'), 3)
3 recall_nb = round(recall_score(y_test, naive_bayse_stress.predict(X_test_scaled), average='weighted'), 3)
4 f1_score_nb = round(f1_score(y_test, naive_bayse_stress.predict(X_test_scaled), average='weighted'), 3)
5 cross_validation_nb = round(np.mean(cross_val_score(naive_bayse_stress, X_train_scaled, y_train, cv = 6)),3)

In [582]: 1 print('Accuracy:', accuracy_nb)
2 print('Precision:', precision_nb)
3 print('Recall:', recall_nb)
4 print('F1_Score:',f1_score_nb)
5 print('Cross Validation:', cross_validation_nb)

Accuracy: 0.853
Precision: 0.873
Recall 0.853
F1_Score: 0.859
Cross Validation: 0.852

In [583]: 1 classification=classification_report(digits=4, y_true=y_test, y_pred=naive_bayse_stress.predict(X_test_scaled))
2 print(classification)

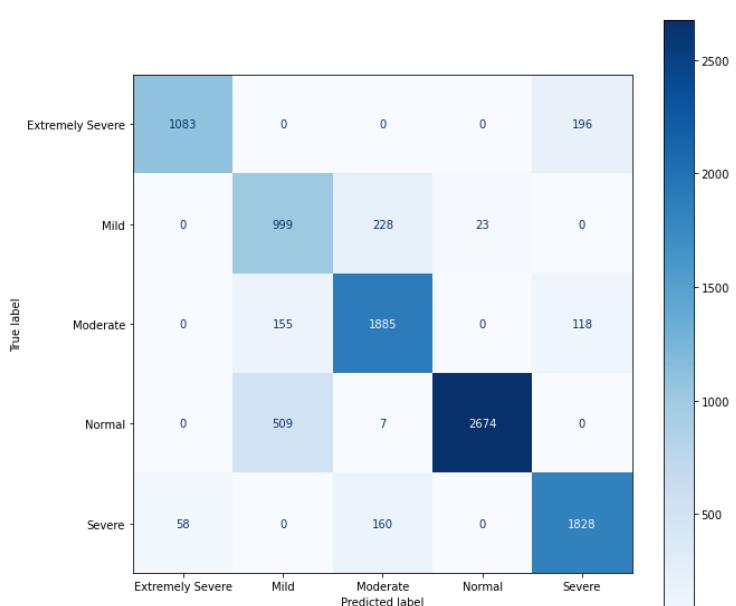
          precision    recall  f1-score   support

Extremely Severe    0.9492   0.8468   0.8950    1279
      Mild           0.6007   0.7992   0.6859    1250
      Moderate        0.8268   0.8735   0.8495    2158
      Normal          0.9915   0.8382   0.9084    3190
      Severe          0.8534   0.8935   0.8730    2046

   accuracy         0.8443   0.8502   0.8424    9923
  macro avg       0.8443   0.8502   0.8424    9923
weighted avg     0.8725   0.8535   0.8585    9923
```

```
In [584]: 1 fig, ax = plt.subplots(figsize=(10, 10))
2 plot_confusion_matrix(naive_bayse_stress, X_test_scaled, y_test,ax=ax, cmap='Blues')

Out[584]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x14baf9f7df0>
```



## Algorithm (5): Logistic Regression

```
In [585]: 1 logistic_regression_stress = LogisticRegression().fit(X_train_scaled, y_train)

In [586]: 1 accuracy_lr = round(accuracy_score(y_test, logistic_regression_stress.predict(X_test_scaled)), 3)
2 precision_lr = round(precision_score(y_test, logistic_regression_stress.predict(X_test_scaled), average='weighted'), 3)
3 recall_lr = round(recall_score(y_test, logistic_regression_stress.predict(X_test_scaled), average='weighted'), 3)
4 f1_score_lr = round(f1_score(y_test, logistic_regression_stress.predict(X_test_scaled), average='weighted'), 3)
5 cross_validation_lr = round(np.mean(cross_val_score(logistic_regression_stress, X_train_scaled, y_train, cv=6)), 3)

In [587]: 1 print('Accuracy:', accuracy_lr)
2 print('Precision:', precision_lr)
3 print('Recall:', recall_lr)
4 print('F1_Score:', f1_score_lr)
5 print('Cross Validation:', cross_validation_lr)

Accuracy: 0.993
Precision: 0.993
Recall: 0.993
F1_Score: 0.993
Cross Validation: 0.989

In [588]: 1 classification = classification_report(digits=4, y_true=y_test, y_pred=logistic_regression_stress.predict(X_test_scaled))
2 print(classification)

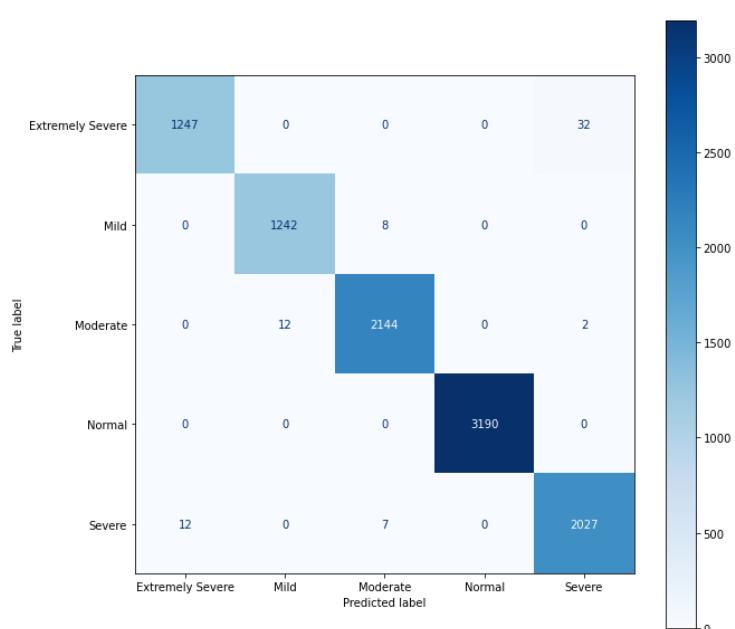
      precision    recall  f1-score   support

Extremely Severe    0.9905    0.9750    0.9827     1279
          Mild    0.9904    0.9936    0.9920     1250
        Moderate    0.9931    0.9935    0.9933     2158
         Normal    1.0000    1.0000    1.0000     3190
        Severe    0.9835    0.9907    0.9871     2046

   accuracy         0.9926
  macro avg    0.9915    0.9906    0.9910     9923
weighted avg    0.9927    0.9926    0.9926     9923
```

```
In [589]: 1 fig, ax = plt.subplots(figsize=(10, 10))
2 plot_confusion_matrix(logistic_regression_stress, X_test_scaled, y_test, ax=ax, cmap='Blues')

Out[589]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x14baef2ff3d0>
```



## Algorithm (6): Support Vector Classifier

```
In [590]: 1 svm_stress = SVC(C=100, gamma=0.1, kernel='rbf').fit(X_train_scaled, y_train)

In [591]: 1 accuracy_svm = round(accuracy_score(y_test, svm_stress.predict(X_test_scaled)), 3)
2 precision_svm = round(precision_score(y_test, svm_stress.predict(X_test_scaled), average='weighted'), 3)
3 recall_svm = round(recall_score(y_test, svm_stress.predict(X_test_scaled), average='weighted'), 3)
4 f1_score_svm = round(f1_score(y_test, svm_stress.predict(X_test_scaled), average='weighted'), 3)
5 cross_validation_svm = round(np.mean(cross_val_score(svm_stress, X_train_scaled, y_train, cv=6)), 3)

In [592]: 1 print('Accuracy:', accuracy_svm)
2 print('Recall:', recall_svm)
3 print('Precision:', precision_svm)
4 print('F1_Score:', f1_score_svm)
5 print('Cross Validation:', cross_validation_svm)

Accuracy: 1.0
Recall: 1.0
Precision: 1.0
F1_Score: 1.0
```

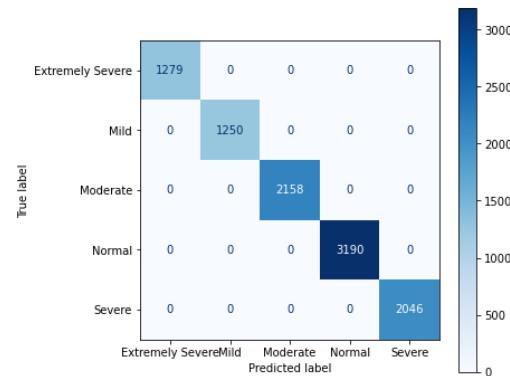
Cross Validation: 1.0

```
In [593]: 1 classification = classification_report(digits=4, y_true=y_test, y_pred=svm_stress.predict(X_test_scaled))
2 print(classification)
```

|                  | precision | recall | f1-score | support |
|------------------|-----------|--------|----------|---------|
| Extremely Severe | 1.0000    | 1.0000 | 1.0000   | 1279    |
| Mild             | 1.0000    | 1.0000 | 1.0000   | 1250    |
| Moderate         | 1.0000    | 1.0000 | 1.0000   | 2158    |
| Normal           | 1.0000    | 1.0000 | 1.0000   | 3190    |
| Severe           | 1.0000    | 1.0000 | 1.0000   | 2046    |
| accuracy         |           |        | 1.0000   | 9923    |
| macro avg        | 1.0000    | 1.0000 | 1.0000   | 9923    |
| weighted avg     | 1.0000    | 1.0000 | 1.0000   | 9923    |

```
In [594]: 1 fig, ax = plt.subplots(figsize=(6, 6))
2 plot_confusion_matrix(svm_stress, X_test_scaled, y_test, ax=ax, cmap='Blues')
```

Out[594]: <sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x14bacaf1c70>



```
In [595]: 1 Result={'Model':['Random_Forest','Decision_Tree','Naive_Baise','K_Nearest_Neighbors','Logistic_Regression','SVM'],
2 'Accuracy(%)':[accuracy_rf*100,accuracy_dt*100,accuracy_nb*100,accuracy_knn*100,accuracy_lr*100, accuracy_svm*100],
3 'Precision(%)': [precision_rf*100,precision_dt*100,precision_nb*100,precision_knn*100,precision_lr*100, precision_svm*100],
4 'Recall(%)': [recall_rf*100,recall_dt*100,recall_nb*100,recall_knn*100,recall_lr*100, recall_svm*100],
5 'F1_Score(%)': [f1_Score_rf*100,f1_score_dt*100,f1_score_nb*100,f1_score_knn*100,f1_score_lr*100,f1_score_svm*100]}
```

```
In [596]: 1 Result_Stress=pd.DataFrame(Result)
2 Result_Stress.to_csv('D:/COURSES/Programming_Language/python/data_scince_track/projects/NTI/Graduation_Project/Datasets/Depression.csv')
```

```
In [597]: 1 Result_Stress
```

```
Out[597]:
```

|   | Model               | Accuracy(%) | Precision(%) | Recall(%) | F1_Score(%) |
|---|---------------------|-------------|--------------|-----------|-------------|
| 0 | Random_Forest       | 89.10       | 89.20        | 89.10     | 88.70       |
| 1 | Decision_Tree       | 74.00       | 74.60        | 74.00     | 74.30       |
| 2 | Naive_Baise         | 85.30       | 87.30        | 85.30     | 85.90       |
| 3 | K_Nearest_Neighbors | 87.00       | 86.80        | 87.00     | 86.90       |
| 4 | Logistic_Regression | 99.30       | 99.30        | 99.30     | 99.30       |
| 5 | SVM                 | 100.00      | 100.00       | 100.00    | 100.00      |

## Test by My Self

```
In [598]: 1 Stress
```

Out[598]:

|       | Q1A | Q6A | Q8A | Q11A | Q12A | Q14A | Q18A | Q22A | Q27A | Q29A | Q32A | Q33A | Q35A | Q39A | TIP11 | TIP12 | TIP13 | TIP14 | TIP15 | TIP16 | TIP17 | TIP18 | TIP19 | TIP20 |   |
|-------|-----|-----|-----|------|------|------|------|------|------|------|------|------|------|------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|---|
| 0     | 1   | 2   | 1   | 1    | 0    | 3    | 0    | 0    | 3    | 2    | 0    | 1    | 0    | 2    | 1     | 1     | 2     | 2     | 5     | 3     | 6     | 5     | 5     | 6     | 3 |
| 1     | 1   | 1   | 3   | 1    | 3    | 3    | 3    | 2    | 1    | 1    | 3    | 3    | 2    | 2    | 2     | 2     | 5     | 3     | 6     | 5     | 5     | 3     | 2     | 6     | 3 |
| 2     | 0   | 0   | 2   | 0    | 1    | 1    | 1    | 0    | 0    | 1    | 1    | 3    | 1    | 3    | 2     | 5     | 6     | 5     | 3     | 2     | 6     | 3     | 5     | 6     | 3 |
| 3     | 0   | 1   | 0   | 1    | 0    | 1    | 1    | 1    | 0    | 1    | 0    | 0    | 0    | 0    | 1     | 7     | 6     | 4     | 5     | 3     | 2     | 6     | 3     | 5     | 6 |
| 4     | 3   | 3   | 3   | 3    | 3    | 2    | 3    | 3    | 1    | 3    | 1    | 3    | 1    | 1    | 1     | 4     | 5     | 7     | 5     | 7     | 6     | 7     | 1     | 1     |   |
| ...   | ... | ... | ... | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   |   |
| 28346 | 1   | 1   | 1   | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1     | 6     | 5     | 6     | 7     | 6     | 4     | 6     | 5     | 5     |   |
| 28347 | 2   | 3   | 3   | 3    | 3    | 3    | 2    | 3    | 2    | 2    | 2    | 2    | 2    | 2    | 2     | 4     | 5     | 5     | 7     | 4     | 6     | 4     | 7     | 4     |   |
| 28348 | 1   | 0   | 0   | 0    | 0    | 0    | 0    | 0    | 1    | 1    | 0    | 1    | 0    | 1    | 6     | 6     | 7     | 5     | 6     | 3     | 6     | 1     | 5     |       |   |
| 28349 | 2   | 2   | 3   | 2    | 2    | 3    | 2    | 3    | 3    | 2    | 2    | 3    | 2    | 2    | 1     | 6     | 5     | 7     | 3     | 5     | 3     | 5     | 5     |       |   |
| 28350 | 1   | 1   | 0   | 2    | 0    | 0    | 2    | 1    | 2    | 2    | 1    | 1    | 2    | 6    | 2     | 3     | 5     | 6     | 3     | 5     | 5     | 5     | 1     |       |   |

28351 rows x 34 columns

```
In [599]: 1 Stress.columns
```

Out[599]: Index(['Q1A', 'Q6A', 'Q8A', 'Q11A', 'Q12A', 'Q14A', 'Q18A', 'Q22A', 'Q27A', 'Q29A', 'Q32A', 'Q33A', 'Q35A', 'Q39A', 'TIP11', 'TIP12', 'TIP13', 'TIP14', 'TIP15', 'TIP16', 'TIP17', 'TIP18', 'TIP19', 'TIP20'])

```
'education', 'urban', 'gender', 'religion', 'orientation', 'race',
'married', 'familysize', 'age_group', 'predict'],
dtype='object')
```

```
In [600]: 1 Depression[Depression['predict'] == 'Moderate']
```

```
Out[600]:
```

|       | Q3A | Q5A | Q10A | Q13A | Q16A | Q17A | Q21A | Q24A | Q26A | Q31A | Q34A | Q37A | Q38A | Q42A | TIP1 | TIP12 | TIP13 | TIP14 | TIP15 | TIP16 | TIP17 | TIP18 | TIP |
|-------|-----|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|-------|-------|-------|-------|-------|-------|-------|-----|
| 0     | 1   | 2   | 2    | 0    | 1    | 2    | 0    | 0    | 1    | 2    | 3    | 1    | 0    | 1    | 1    | 1     | 7     | 4     | 6     | 4     | 6     | 1     |     |
| 6     | 0   | 0   | 2    | 2    | 2    | 2    | 2    | 1    | 2    | 0    | 1    | 3    | 1    | 1    | 1    | 5     | 1     | 4     | 6     | 5     | 5     | 7     | 6   |
| 7     | 0   | 2   | 0    | 3    | 2    | 3    | 0    | 2    | 0    | 1    | 0    | 0    | 0    | 2    | 5    | 2     | 5     | 5     | 2     | 6     | 7     | 6     |     |
| 10    | 1   | 1   | 1    | 1    | 1    | 2    | 1    | 1    | 1    | 1    | 2    | 1    | 1    | 0    | 0    | 1     | 4     | 4     | 7     | 7     | 5     | 4     | 4   |
| 11    | 1   | 2   | 1    | 0    | 2    | 0    | 0    | 2    | 0    | 2    | 0    | 0    | 1    | 3    | 2    | 7     | 7     | 1     | 6     | 5     | 4     | 7     |     |
| ...   | ... | ... | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...   | ...   | ...   | ...   | ...   | ...   | ...   |     |
| 28315 | 0   | 1   | 1    | 2    | 1    | 1    | 2    | 1    | 2    | 1    | 1    | 1    | 1    | 1    | 5    | 6     | 5     | 5     | 6     | 6     | 6     | 6     |     |
| 28316 | 1   | 0   | 2    | 1    | 1    | 1    | 0    | 1    | 1    | 1    | 1    | 3    | 0    | 3    | 4    | 4     | 6     | 5     | 2     | 5     | 4     | 6     |     |
| 28317 | 1   | 1   | 2    | 3    | 1    | 2    | 0    | 1    | 2    | 0    | 3    | 1    | 0    | 1    | 4    | 4     | 5     | 5     | 5     | 6     | 6     | 5     |     |
| 28341 | 1   | 2   | 2    | 1    | 2    | 2    | 1    | 1    | 1    | 1    | 2    | 1    | 1    | 1    | 5    | 4     | 5     | 6     | 5     | 5     | 6     | 5     |     |
| 28349 | 1   | 2   | 0    | 3    | 0    | 1    | 1    | 1    | 2    | 1    | 1    | 0    | 1    | 2    | 1    | 6     | 5     | 7     | 3     | 5     | 3     | 5     |     |

5277 rows × 34 columns

```
In [601]: 1 input_data_Mild = [1, 2, 1, 1, 0, 3, 0, 0, 3, 2, 0, 1, 0, 2, 1, 1, 7, 4, 6, 4, 6, 1, 6, 1, 1, 3, 2, 4, 5, 70, 1, 5, 2]
2 input_data_scaled = scaler.transform([input_data_Mild])
```

```
In [602]: 1 prediction = svm_stress.predict(input_data_scaled)
2 print("Prediction:", prediction[0])
```

Prediction: Mild

```
In [603]: 1 input_data_Severe = [1, 1, 3, 1, 3, 3, 3, 2, 1, 1, 3, 3, 2, 2, 2, 5, 3, 6, 5, 5, 5, 6, 3, 3, 3, 2, 2, 10, 1, 10, 1, 4, 3]
2 input_data_scaled = scaler.transform([input_data_Severe])
```

```
In [604]: 1 prediction = svm_stress.predict(input_data_scaled)
2 print("Prediction:", prediction[0])
```

Prediction: Severe

```
In [605]: 1 input_data_Normal = [0, 0, 2, 0, 1, 1, 0, 0, 1, 1, 3, 1, 3, 2, 5, 6, 5, 3, 2, 6, 3, 5, 5, 2, 3, 2, 7, 2, 60, 1, 4, 3]
2 input_data_scaled = scaler.transform([input_data_Normal])
```

```
In [606]: 1 prediction = svm_stress.predict(input_data_scaled)
2 print("Prediction:", prediction[0])
```

Prediction: Normal

```
In [607]: 1 input_data_Exremely_severe = [2, 3, 3, 3, 3, 2, 3, 2, 2, 2, 2, 2, 4, 5, 5, 7, 4, 6, 4, 7, 4, 4, 3, 2, 1, 10, 1, 10, 1
2 input_data_scaled = scaler.transform([input_data_Exremely_severe])
```

```
In [608]: 1 prediction = svm_stress.predict(input_data_scaled)
2 print("Prediction:", prediction[0])
```

Prediction: Extremely Severe

```
In [609]: 1 input_data_Moderate = [1, 2, 2, 0, 1, 2, 0, 0, 1, 2, 3, 1, 0, 1, 1, 1, 7, 4, 6, 4, 6, 1, 6, 1, 1, 3, 2, 4, 5, 70, 1, 5, 2]
2 input_data_scaled = scaler.transform([input_data_Moderate])
```

```
In [610]: 1 prediction = svm_stress.predict(input_data_scaled)
2 print("Prediction:", prediction[0])
```

Prediction: Mild

```
In [611]: 1 column_scales = {
2     'Q1A': list(range(4)),
3     'Q6A': list(range(4)),
4     'Q8A': list(range(4)),
5     'Q11A': list(range(4)),
6     'Q12A': list(range(4)),
7     'Q14A': list(range(4)),
8     'Q18A': list(range(4)),
9     'Q22A': list(range(4)),
10    'Q27A': list(range(4)),
11    'Q29A': list(range(4)),
12    'Q32A': list(range(4)),
13    'Q33A': list(range(4)),
14    'Q35A': list(range(4)),
15    'Q39A': list(range(4)),
16    'TIP11': list(range(1, 8)),
17    'TIP12': list(range(1, 8)),
18    'TIP13': list(range(1, 8)),
19    'TIP14': list(range(1, 8)),
20    'TIP15': list(range(1, 8)),
21    'TIP16': list(range(1, 8)),
22    'TIP17': list(range(1, 8)),
23    'TIP18': list(range(1, 8)),
24    'TIP19': list(range(1, 8)),
25    'TIP10': list(range(1, 8)),
26    'education': list(range(1, 5)),
27    'urban': list(range(1, 4)),
28    'gender': list(range(1, 4)),
29    'religion': list(range(1, 13)),
30    'orientation': list(range(1, 6)),
31    'race': [10, 20, 30, 40, 50, 60, 70],
```

```

32     'married': list(range(1, 4)),
33     'familysize': list(range(1, 101)),
34     'age_group': list(range(1, 7))
35 }

```

```

In [612]: 1 random_lists = []
2 for i in range(30):
3     random_list = [
4         random.choice(column_scales[column]) for column in [
5             'Q1A', 'Q6A', 'Q8A', 'Q11A', 'Q12A', 'Q14A', 'Q18A', 'Q22A', 'Q27A',
6             'Q29A', 'Q32A', 'Q33A', 'Q35A', 'Q39A', 'TIP11', 'TIP12', 'TIP13',
7             'TIP14', 'TIP15', 'TIP16', 'TIP17', 'TIP18', 'TIP19', 'TIP10',
8             'education', 'urban', 'gender', 'religion', 'orientation', 'race',
9             'married', 'familysize', 'age_group'
10        ]
11    ]
12 random_lists.append(random_list)

```

```

In [613]: 1 for i, random_list in enumerate(random_lists, start=1):
2     print(f"random_values{i} = {random_list}")
3
random_values1 = [2, 2, 0, 0, 1, 3, 2, 2, 3, 1, 2, 3, 0, 5, 4, 4, 1, 1, 5, 1, 6, 6, 7, 5, 4, 1, 1, 11, 1, 70, 1, 82, 4]
random_values2 = [2, 0, 3, 2, 2, 3, 1, 3, 3, 1, 2, 1, 2, 5, 3, 7, 2, 2, 5, 7, 5, 4, 1, 2, 1, 2, 4, 4, 50, 3, 17, 3]
random_values3 = [3, 0, 0, 3, 3, 2, 0, 1, 2, 2, 0, 2, 3, 0, 4, 1, 4, 5, 6, 3, 2, 1, 6, 1, 1, 2, 3, 3, 3, 10, 1, 6, 1]
random_values4 = [0, 1, 0, 2, 1, 2, 0, 2, 1, 1, 2, 2, 1, 3, 1, 5, 1, 4, 7, 1, 6, 5, 7, 2, 7, 2, 2, 3, 12, 1, 60, 2, 55, 5]
random_values5 = [1, 1, 3, 1, 1, 3, 3, 3, 0, 2, 3, 3, 3, 2, 1, 4, 5, 2, 5, 3, 1, 1, 5, 1, 4, 2, 2, 11, 5, 20, 2, 59, 6]
random_values6 = [2, 2, 0, 2, 3, 3, 0, 2, 1, 0, 0, 3, 3, 3, 7, 5, 5, 4, 1, 5, 3, 2, 7, 1, 1, 3, 10, 5, 30, 1, 62, 5]
random_values7 = [0, 1, 0, 3, 0, 1, 3, 2, 0, 3, 0, 2, 0, 2, 1, 7, 5, 1, 4, 5, 6, 4, 2, 2, 2, 3, 1, 12, 2, 40, 3, 16, 5]
random_values8 = [2, 3, 2, 2, 0, 1, 3, 3, 2, 1, 0, 1, 0, 3, 2, 1, 6, 1, 5, 5, 6, 7, 5, 4, 2, 3, 10, 4, 70, 2, 72, 5]
random_values9 = [1, 0, 3, 1, 2, 1, 3, 1, 3, 1, 0, 0, 6, 2, 4, 1, 2, 2, 3, 3, 3, 8, 1, 20, 2, 100, 2]
random_values10 = [3, 0, 2, 1, 0, 2, 1, 3, 3, 3, 0, 0, 3, 3, 2, 6, 5, 4, 6, 5, 5, 1, 1, 3, 3, 1, 3, 6, 2, 70, 3, 58, 3]
random_values11 = [0, 2, 0, 3, 3, 2, 0, 3, 1, 3, 2, 1, 3, 2, 2, 3, 7, 4, 6, 6, 2, 3, 2, 7, 2, 3, 2, 2, 4, 20, 2, 32, 1]
random_values12 = [2, 1, 3, 0, 1, 2, 3, 0, 1, 0, 3, 2, 3, 1, 5, 3, 7, 5, 2, 7, 2, 3, 2, 6, 4, 2, 3, 8, 4, 20, 2, 7, 2]
random_values13 = [1, 0, 1, 2, 3, 2, 0, 2, 0, 1, 0, 0, 1, 4, 5, 6, 5, 2, 6, 4, 7, 6, 4, 3, 3, 3, 6, 3, 40, 3, 21, 1]
random_values14 = [0, 3, 1, 2, 3, 1, 0, 2, 3, 1, 2, 3, 0, 3, 3, 2, 5, 1, 3, 3, 5, 5, 4, 2, 2, 1, 2, 5, 3, 20, 2, 29, 6]
random_values15 = [0, 3, 0, 0, 1, 2, 1, 0, 1, 2, 3, 2, 5, 4, 3, 5, 7, 4, 1, 1, 2, 6, 3, 3, 1, 2, 5, 60, 2, 68, 3]
random_values16 = [3, 3, 0, 1, 0, 1, 2, 3, 1, 3, 0, 1, 0, 1, 7, 5, 5, 1, 6, 3, 6, 3, 5, 1, 3, 4, 2, 2, 10, 2, 10, 3, 19, 1]
random_values17 = [0, 2, 3, 1, 3, 0, 1, 0, 3, 1, 2, 0, 0, 1, 7, 1, 1, 4, 4, 5, 3, 5, 5, 3, 3, 1, 3, 12, 5, 30, 3, 44, 4]
random_values18 = [0, 1, 1, 2, 0, 1, 1, 3, 0, 0, 0, 0, 3, 3, 1, 1, 6, 2, 4, 6, 4, 6, 3, 2, 1, 3, 5, 3, 40, 3, 36, 3]
random_values19 = [1, 0, 0, 0, 2, 2, 1, 2, 1, 3, 2, 3, 6, 1, 1, 4, 5, 6, 1, 5, 6, 3, 4, 2, 1, 5, 2, 30, 3, 36, 3]
random_values20 = [2, 3, 2, 0, 3, 2, 3, 1, 3, 1, 2, 2, 1, 5, 1, 2, 6, 1, 4, 2, 6, 6, 4, 4, 1, 2, 2, 4, 30, 3, 1, 3]
random_values21 = [3, 0, 0, 2, 1, 2, 2, 0, 2, 0, 1, 1, 4, 7, 5, 3, 4, 5, 6, 4, 5, 1, 2, 2, 9, 5, 70, 1, 3, 2]
random_values22 = [3, 3, 1, 2, 3, 3, 0, 2, 1, 2, 2, 0, 3, 1, 7, 2, 5, 4, 1, 2, 5, 2, 4, 5, 2, 2, 1, 5, 3, 10, 1, 85, 1]
random_values23 = [0, 0, 2, 3, 0, 2, 0, 3, 3, 0, 1, 2, 2, 0, 3, 7, 3, 4, 7, 1, 4, 7, 2, 7, 1, 2, 2, 8, 3, 20, 3, 65, 6]
random_values24 = [3, 1, 0, 3, 2, 2, 1, 2, 0, 1, 2, 1, 2, 5, 3, 7, 7, 5, 5, 1, 6, 4, 7, 1, 1, 1, 10, 1, 40, 3, 51, 3]
random_values25 = [2, 3, 1, 1, 3, 3, 0, 1, 1, 3, 0, 1, 3, 0, 3, 3, 6, 4, 4, 2, 1, 2, 7, 3, 4, 1, 3, 7, 1, 10, 2, 73, 6]
random_values26 = [0, 1, 3, 3, 3, 0, 0, 2, 0, 2, 1, 2, 3, 0, 7, 1, 5, 6, 2, 1, 2, 4, 5, 7, 1, 2, 1, 3, 2, 60, 2, 90, 1]
random_values27 = [1, 1, 0, 0, 1, 0, 1, 3, 1, 2, 1, 0, 2, 3, 7, 6, 5, 7, 4, 3, 6, 4, 3, 2, 4, 1, 3, 7, 4, 40, 1, 41, 1]
random_values28 = [1, 1, 2, 1, 0, 2, 2, 3, 0, 0, 3, 2, 3, 1, 4, 3, 6, 5, 7, 2, 7, 4, 4, 1, 3, 3, 2, 5, 5, 40, 1, 79, 4]
random_values29 = [0, 3, 3, 3, 1, 3, 1, 0, 2, 2, 0, 2, 1, 0, 4, 4, 6, 2, 1, 6, 3, 6, 1, 3, 3, 3, 1, 5, 20, 1, 65, 3]
random_values30 = [2, 2, 2, 2, 3, 0, 2, 3, 2, 1, 1, 1, 3, 2, 2, 7, 3, 1, 3, 5, 2, 1, 2, 3, 2, 3, 9, 3, 50, 1, 94, 5]

```

```

In [614]: 1 random_values1 = [2, 0, 1, 2, 3, 3, 0, 0, 0, 0, 3, 0, 0, 3, 4, 2, 1, 3, 2, 3, 1, 2, 1, 4, 4, 2, 2, 5, 4, 50, 2, 62, 6]
2 random_values2 = [1, 0, 1, 3, 1, 1, 3, 2, 2, 3, 2, 1, 3, 0, 3, 5, 5, 2, 6, 1, 4, 1, 2, 1, 3, 2, 1, 1, 2, 20, 2, 35, 1]
3 random_values3 = [3, 1, 1, 1, 1, 0, 3, 2, 1, 2, 1, 3, 2, 1, 6, 5, 2, 4, 6, 3, 7, 2, 5, 5, 3, 3, 2, 3, 2, 10, 2, 85, 1]
4 random_values4 = [1, 0, 3, 2, 0, 2, 1, 2, 2, 2, 0, 2, 3, 6, 1, 2, 3, 6, 2, 7, 4, 4, 4, 3, 1, 7, 5, 20, 3, 27, 5]
5 random_values5 = [3, 1, 2, 0, 1, 1, 3, 1, 3, 2, 2, 2, 3, 4, 5, 7, 1, 6, 1, 5, 5, 5, 3, 4, 1, 3, 11, 5, 10, 3, 78, 2]
6 random_values6 = [3, 3, 3, 2, 2, 3, 3, 3, 2, 2, 0, 4, 6, 6, 4, 3, 1, 1, 3, 2, 5, 3, 1, 3, 4, 1, 1, 70, 3, 40, 6]
7 random_values7 = [1, 3, 1, 1, 0, 1, 3, 3, 2, 1, 0, 1, 3, 7, 1, 7, 4, 6, 1, 4, 7, 2, 3, 1, 11, 3, 70, 2, 59, 3]
8 random_values8 = [1, 2, 1, 0, 3, 3, 0, 0, 2, 3, 1, 0, 2, 3, 6, 6, 6, 7, 2, 5, 7, 2, 4, 1, 2, 7, 2, 70, 2, 88, 6]
9 random_values9 = [2, 3, 3, 2, 2, 3, 0, 0, 2, 0, 1, 1, 0, 1, 0, 2, 2, 4, 7, 3, 6, 6, 2, 7, 2, 1, 2, 9, 5, 60, 1, 24, 6]
10 random_values10 = [2, 2, 3, 0, 0, 3, 1, 3, 3, 3, 1, 7, 4, 5, 5, 6, 3, 7, 3, 5, 2, 3, 3, 1, 2, 70, 3, 86, 4]
11 random_values11 = [1, 2, 2, 1, 2, 3, 0, 0, 1, 0, 1, 3, 0, 2, 1, 1, 3, 3, 7, 1, 3, 5, 7, 2, 3, 3, 3, 10, 4, 50, 1, 74, 2]
12 random_values12 = [2, 2, 2, 1, 0, 3, 0, 1, 1, 1, 0, 3, 0, 7, 4, 3, 5, 6, 7, 1, 3, 4, 5, 4, 2, 1, 7, 5, 20, 1, 27, 1]
13 random_values13 = [1, 2, 3, 0, 0, 2, 3, 1, 3, 1, 3, 0, 0, 3, 2, 4, 4, 3, 1, 7, 6, 5, 6, 2, 1, 3, 2, 7, 3, 30, 3, 94, 6]
14 random_values14 = [1, 0, 2, 2, 1, 0, 3, 0, 1, 3, 0, 0, 3, 1, 1, 7, 2, 7, 3, 1, 4, 6, 7, 4, 3, 2, 3, 5, 20, 1, 28, 4]
15 random_values15 = [0, 0, 3, 2, 1, 3, 2, 0, 0, 3, 2, 1, 2, 2, 6, 7, 3, 7, 3, 6, 1, 6, 2, 2, 1, 2, 11, 4, 10, 1, 39, 4]
16 random_values16 = [1, 2, 0, 0, 1, 0, 0, 0, 3, 2, 2, 3, 1, 2, 3, 6, 5, 6, 1, 7, 7, 2, 1, 1, 3, 50, 3, 3, 5]
17 random_values17 = [2, 3, 3, 1, 2, 0, 0, 1, 2, 1, 0, 2, 3, 2, 7, 3, 1, 2, 1, 6, 7, 2, 5, 3, 3, 2, 12, 4, 50, 1, 36, 5]
18 random_values18 = [3, 2, 2, 3, 3, 2, 1, 2, 2, 3, 1, 1, 7, 7, 5, 1, 2, 5, 1, 5, 5, 3, 2, 3, 2, 2, 60, 3, 12, 4]
19 random_values19 = [2, 3, 0, 3, 2, 2, 1, 1, 0, 3, 2, 3, 1, 2, 3, 1, 7, 6, 4, 2, 6, 1, 6, 5, 4, 2, 1, 3, 3, 40, 1, 88, 5]
20 random_values20 = [0, 0, 3, 0, 2, 1, 0, 1, 2, 2, 3, 1, 2, 0, 1, 6, 6, 4, 5, 7, 3, 2, 5, 1, 6, 4, 1, 3, 7, 4, 40, 1, 13, 2]
21 random_values21 = [3, 3, 0, 3, 1, 0, 1, 1, 2, 0, 0, 2, 3, 1, 4, 1, 4, 3, 3, 4, 4, 4, 1, 4, 2, 2, 3, 8, 5, 70, 1, 97, 6]
22 random_values22 = [0, 2, 1, 0, 2, 2, 0, 0, 3, 1, 1, 2, 3, 7, 4, 2, 5, 6, 3, 6, 5, 7, 2, 2, 2, 3, 1, 3, 10, 3, 74, 5]
23 random_values23 = [3, 0, 1, 2, 3, 2, 3, 3, 1, 3, 0, 0, 5, 7, 6, 3, 7, 3, 2, 3, 7, 4, 4, 2, 2, 4, 3, 40, 3, 82, 3]
24 random_values24 = [2, 3, 1, 0, 2, 3, 3, 2, 0, 1, 3, 4, 5, 2, 1, 3, 7, 3, 3, 4, 2, 3, 2, 3, 3, 30, 1, 49, 2]
25 random_values25 = [0, 2, 2, 1, 0, 3, 2, 1, 0, 2, 3, 1, 0, 0, 3, 1, 7, 1, 2, 2, 2, 7, 2, 2, 1, 3, 2, 3, 30, 3, 73, 1]
26 random_values26 = [0, 0, 3, 0, 2, 1, 0, 2, 1, 0, 2, 2, 1, 0, 1, 6, 6, 4, 5, 6, 5, 2, 2, 3, 2, 3, 3, 2, 6, 4, 60, 1, 46, 1]
27 random_values27 = [3, 3, 1, 1, 0, 3, 1, 0, 3, 1, 2, 2, 0, 3, 1, 4, 4, 3, 4, 4, 6, 3, 3, 6, 3, 1, 3, 4, 3, 30, 2, 39, 3]
28 random_values28 = [3, 0, 1, 0, 2, 3, 2, 2, 1, 0, 0, 0, 3, 7, 5, 1, 5, 4, 1, 1, 5, 2, 2, 1, 1, 4, 5, 40, 3, 31, 1]
29 random_values29 = [2, 0, 0, 1, 0, 2, 2, 2, 2, 3, 1, 2, 1, 6, 3, 2, 2, 7, 6, 1, 3, 2, 3, 1, 4, 1, 50, 1, 45, 6]
30 random_values30 = [0, 1, 3, 2, 3, 3, 1, 2, 2, 0, 3, 2, 1, 3, 4, 6, 3, 5, 7, 2, 1, 4, 5, 4, 3, 2, 2, 5, 5, 40, 3, 35, 2]

```

```

In [615]: 1 input_data_scaled = scaler.transform([random_values1])
2 prediction = svm_stress.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

```

Prediction: Mild

```

In [616]: 1 input_data_scaled = scaler.transform([random_values2])
2 prediction = svm_stress.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

```

Prediction: Moderate

```

In [617]: 1 input_data_scaled = scaler.transform([random_values3])
2 prediction = svm_stress.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

```

Prediction: Moderate

```

In [618]: 1 input_data_scaled = scaler.transform([random_values4])
2 prediction = svm_stress.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

```

```
5 |print("Prediction:", prediction[0])  
Prediction: Moderate  
  
In [619]: 1 input_data_scaled = scaler.transform([random_values5])  
2 prediction = svm_stress.predict(input_data_scaled)  
3 print("Prediction:", prediction[0])  
Prediction: Mild  
  
In [620]: 1 input_data_scaled = scaler.transform([random_values6])  
2 prediction = svm_stress.predict(input_data_scaled)  
3 print("Prediction:", prediction[0])  
Prediction: Extremely Severe  
  
In [621]: 1 input_data_scaled = scaler.transform([random_values7])  
2 prediction = svm_stress.predict(input_data_scaled)  
3 print("Prediction:", prediction[0])  
Prediction: Moderate  
  
In [622]: 1 input_data_scaled = scaler.transform([random_values8])  
2 prediction = svm_stress.predict(input_data_scaled)  
3 print("Prediction:", prediction[0])  
Prediction: Moderate  
  
In [623]: 1 input_data_scaled = scaler.transform([random_values9])  
2 prediction = svm_stress.predict(input_data_scaled)  
3 print("Prediction:", prediction[0])  
Prediction: Moderate  
  
In [624]: 1 input_data_scaled = scaler.transform([random_values10])  
2 prediction = svm_stress.predict(input_data_scaled)  
3 print("Prediction:", prediction[0])  
Prediction: Severe  
  
In [625]: 1 input_data_scaled = scaler.transform([random_values11])  
2 prediction = svm_stress.predict(input_data_scaled)  
3 print("Prediction:", prediction[0])  
Prediction: Moderate  
  
In [626]: 1 input_data_scaled = scaler.transform([random_values12])  
2 prediction = svm_stress.predict(input_data_scaled)  
3 print("Prediction:", prediction[0])  
Prediction: Mild  
  
In [627]: 1 input_data_scaled = scaler.transform([random_values13])  
2 prediction = svm_stress.predict(input_data_scaled)  
3 print("Prediction:", prediction[0])  
Prediction: Mild  
  
In [628]: 1 input_data_scaled = scaler.transform([random_values14])  
2 prediction = svm_stress.predict(input_data_scaled)  
3 print("Prediction:", prediction[0])  
Prediction: Mild  
  
In [629]: 1 input_data_scaled = scaler.transform([random_values15])  
2 prediction = svm_stress.predict(input_data_scaled)  
3 print("Prediction:", prediction[0])  
Prediction: Moderate  
  
In [630]: 1 input_data_scaled = scaler.transform([random_values16])  
2 prediction = svm_stress.predict(input_data_scaled)  
3 print("Prediction:", prediction[0])  
Prediction: Mild  
  
In [631]: 1 input_data_scaled = scaler.transform([random_values17])  
2 prediction = svm_stress.predict(input_data_scaled)  
3 print("Prediction:", prediction[0])  
Prediction: Moderate  
  
In [632]: 1 input_data_scaled = scaler.transform([random_values18])  
2 prediction = svm_stress.predict(input_data_scaled)  
3 print("Prediction:", prediction[0])  
Prediction: Severe  
  
In [633]: 1 input_data_scaled = scaler.transform([random_values19])  
2 prediction = svm_stress.predict(input_data_scaled)  
3 print("Prediction:", prediction[0])  
Prediction: Mild  
  
In [634]: 1 input_data_scaled = scaler.transform([random_values20])  
2 prediction = svm_stress.predict(input_data_scaled)  
3 print("Prediction:", prediction[0])  
Prediction: Moderate  
  
In [635]: 1 input_data_scaled = scaler.transform([random_values21])  
2 prediction = svm_stress.predict(input_data_scaled)
```

```

2 prediction = svm_stress.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Moderate

In [636]: 1 input_data_scaled = scaler.transform([random_values22])
2 prediction = svm_stress.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Moderate

In [637]: 1 input_data_scaled = scaler.transform([random_values23])
2 prediction = svm_stress.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Severe

In [638]: 1 input_data_scaled = scaler.transform([random_values24])
2 prediction = svm_stress.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Moderate

In [639]: 1 input_data_scaled = scaler.transform([random_values25])
2 prediction = svm_stress.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Mild

In [640]: 1 input_data_scaled = scaler.transform([random_values26])
2 prediction = svm_stress.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Mild

In [641]: 1 input_data_scaled = scaler.transform([random_values27])
2 prediction = svm_stress.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Moderate

In [642]: 1 input_data_scaled = scaler.transform([random_values28])
2 prediction = svm_stress.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Normal

In [643]: 1 input_data_scaled = scaler.transform([random_values29])
2 prediction = svm_stress.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Mild

In [644]: 1 input_data_scaled = scaler.transform([random_values30])
2 prediction = svm_stress.predict(input_data_scaled)
3 print("Prediction:", prediction[0])

Prediction: Severe

```

## (5) Saving Models

### Saving Model of Depression

```

In [645]: 1 svm_model_path = "D:/COURSES/Programming_Language/python/data_science_track/projects/NTI/Graduation_Project/Datasets/Depression"
2 with open(svm_model_path, 'wb') as file:
3     pickle.dump(svm_depression, file)

In [646]: 1 svm_model_path_depression = "D:/COURSES/Programming_Language/python/data_science_track/projects/NTI/Graduation_Project/Datasets/Depression"
2 loaded_model = pickle.load(open(svm_model_path_depression, 'rb'))

In [647]: 1 input_data_Normal = [0, 2, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 7, 6, 4, 5, 3, 2, 6, 3, 5, 2, 4, 2, 2, 2, 2, 60, 1, 2, 4]
2 input_data_scaled = scaler.transform([input_data_Normal])

In [648]: 1 predict = loaded_model.predict(input_data_scaled)
2 print(predict)

['Normal']

```

### Saving a Model of Anxiety

```

In [649]: 1 svm_model_path = "D:/COURSES/Programming_Language/python/data_science_track/projects/NTI/Graduation_Project/Datasets/Depression"
2 with open(svm_model_path, 'wb') as file:
3     pickle.dump(svm_anxiety, file)

In [650]: 1 svm_model_path_depression = "D:/COURSES/Programming_Language/python/data_science_track/projects/NTI/Graduation_Project/Datasets/Depression"
2 loaded_model = pickle.load(open(svm_model_path_depression, 'rb'))

In [651]: 1 input_data_Extremely_Severe = [1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 5, 3, 6, 5, 5, 5, 6, 3, 3, 3, 2, 2, 10, 1, 10, 1]

```

```
In [652]: 1 predict = loaded_model.predict(input_data_scaled)
2 print(predict)
['Extremely Severe']
```

## Saving a Model of Stress

```
In [653]: 1 svm_model_path = "D:/COURSES/Programming Language/python/data_scince_track/projects/NTI/Graduation_Project/Datasets/Depressi
2 with open(svm_model_path, 'wb') as file:
3     pickle.dump(svm_stress, file)
```

```
In [654]: 1 svm_model_path_depression = "D:/COURSES/Programming Language/python/data_scince_track/projects/NTI/Graduation_Project/Database
2 loaded_model = pickle.load(open(svm_model_path_depression, 'rb'))
```

```
In [655]: 1 input_data_Mild = [1, 2, 1, 1, 0, 3, 0, 0, 3, 2, 0, 1, 0, 2, 1, 1, 7, 4, 6, 4, 6, 1, 6, 1, 1, 3, 2, 4, 5, 70, 1, 5, 2]
2 input_data_scaled = scaler.transform([input_data_Mild])
```

```
In [656]: 1 predict = loaded_model.predict(input_data_scaled)
2 print(predict)
['Mild']
```

# Saved Models Working Well 🌟

## (6) References

- 1- [Mental health of youth ML model](#)
- 2- [Indian Youth Mental Health](#)
- 3- [The DASS-42 is designed to measure the emotional states of depression, anxiety and stress.](#)
- 4- [Depression Anxiety Stress Scales Responses](#)
- 5- [Depression Classification \[EDA + 97% SVC acc\]](#)
- 6- [DAS PREDICTION](#)
- 7- [TIPIs Questions](#)

متناسش تصلي على النبي ❤️