# INTERNATIONAL SCHOOL OF MANAGEMENT & TECHNOLOGY

## ASSIGNMENT COVER SHEET

### STUDENT DETAILS

| | | | |
|---|---|---|---|
| Student ID | **219467076** | Reg No. | **219467076** |
| Family Name | Barsha Basnet | Given Name | Barsha Basnet |
| Enrolment Year | 2019 | Section | C |
| Semester | 1st sem (3rd Year) | Email | BarshaBasnet4422@gmail.com |

### UNIT DETAILS

| | | | |
|---|---|---|---|
| Unit Title | **Artificial Intelligence** | Unit Code | CET313 |
| Assessor Name | Dr Kate MacFarlane | Submission Due Date | 8th April 2022 |
| Assignment Title | Intelligent Prototype Development | | |
| Assignment No | | Submitted On | 8th April 2022 |
| Qualification | Bsc.IT | Campus | ISMT |

## 3. Prototype Development:

### 3.2. Data Visualization:

Here, I am going to discuss in short about some of the libraries, functions, metrics that I have implemented for the development of my prototype and its data visualization. They are:

**1. Pandas:**

Pandas is such a library kit for the Python programming language that is useful in manipulating data tables or other key tasks.

**2. Numpy:**

NumPy is such a library that allows the users to have more data storage with less memory.

**3. Matplotlib:**

Matplotlib is a plotting library for the Python programming language as a component of NumPy that uses an object oriented API to embed plots in Python applications.

**4. Lazy Predict:**

LazyPredict is an open-source python library useful for semi-automating our Machine Learning Task by building multiple models without writing much code.

**5. Label Encoder:**

Label Encoder is a library tool useful for encoding the levels of categorical features into numeric values.

**6. Seaborn:**

Seaborn is a library for python that is useful for making statistical graphics by exploring and understanding our data.

**7. Train_test_split function:**

Train_test_split function in Sklearn model selection allows splitting of data sets into two subsets i.e. for training and for testing data.

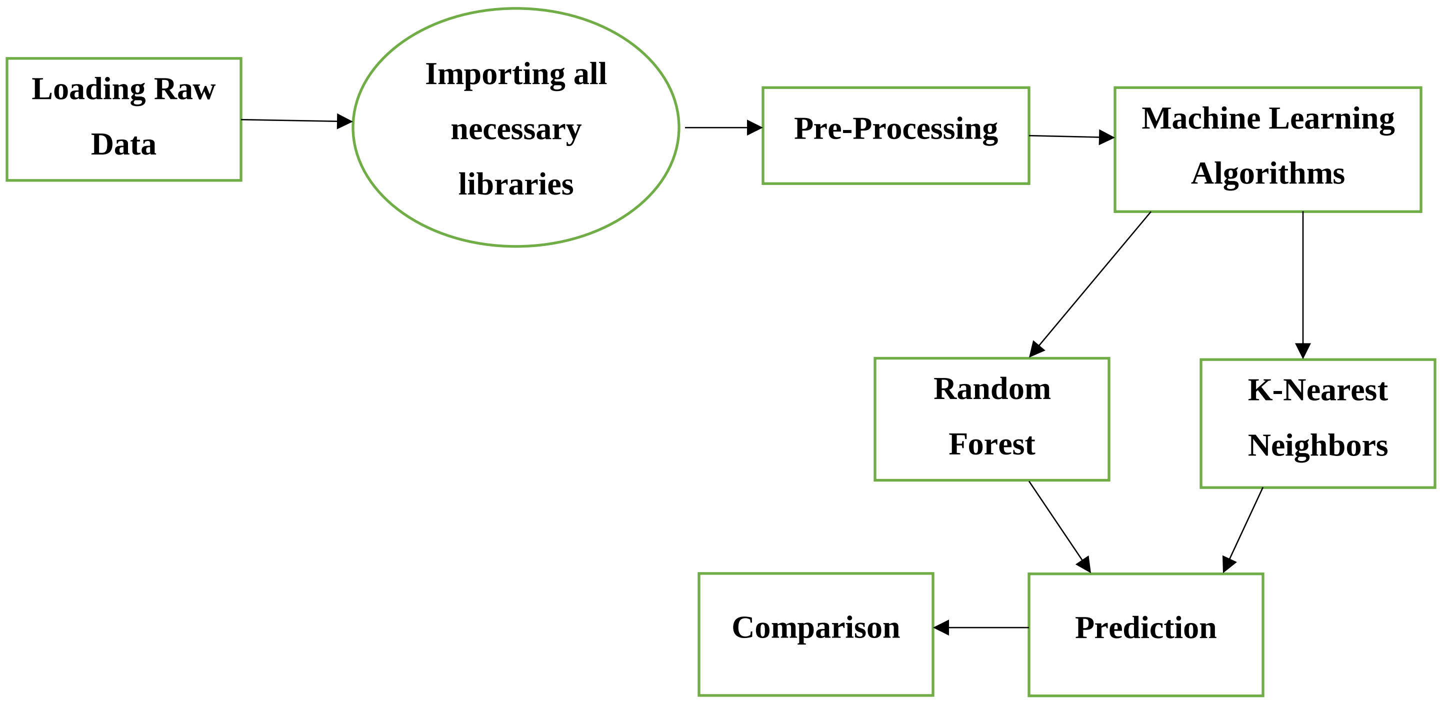
**8. Classification Report:**

It is a performance evaluation metric for showing the precision, recall, F1 score and support of trained classification model

.

**3.2. Planning Diagram:**

The diagram that demonstrate how I have planned to develop my prototype as per the planning made for its development as described above, by running code in jupyter is as shown below:




The diagram shown above is the diagram of the planning that I have done to develop or execute my prototype. To describe, in short about the planning process, at first I will load the data in dataset and after that I will import all the necessary libraries required to execute my prototype. After loading the dataset, the data set will be preprocessed to ultimately get the train and test set that we will require while building my prototype in further stages. After that, I will be comparing the two model or algorithms i.e. Random Forest and K-Nearest Neighbors and determine whether the algorithm that I have specifically used form my prototype gives the accurate result than the other algorithm or not. However, many other models can be implemented for stress detection but will not be compared specifically.

### 3.3. Testing Documentation:

Testing documentation is basically developed before or during the testing of any prototype or software so that it will be helpful to estimate testing effort needed, test coverage, resource tracking, execution progress, etc. My main aim is to develop a prototype that can detect the stress level of people caused due to lack of quality sleep. Therefore, I will be performing testing by making the use of the stress related dataset to predict and analyze whether an individual is stressed or not. When developing the prototype, I will be implementing or showing all the supervised algorithms to train the data. However, I will be focusing most specifically towards the outcome that I will get from supervised Lazy or KNN machine learning algorithm. The program has been developed by using jupyter notebook. My prototype will basically classify the stress level of an individual based on five different categories that is high, medium high, medium low, low/normal and medium level stressed after the prediction. While performing testing, the output of prototype should be able to meet the development objectives.

#### 3.3.1. Features of my prototype:

All the features present in the dataset that has been used for my prototype development is as explained below:

Stress level is determined by the eight different features of an individual seen during their sleep. They are ‘sr’= “snoring rate”, ‘rr’= “respiration rate”, ‘t’ = “body temperature”, ‘lm’ = “limb movement”, ‘bo’ = “blood oxygen”, ‘rem’= “eye movement”, ‘sr.1’ = “sleeping hours” and ‘hr’ = “heart rate”.

### 3.4. Screenshots of code output (DUTTA, n.d.):

The lists of screenshots of code along with its output is shown below with comments:

**1. Imported panda library and train dataset was read.**

```
In [17]: #importing the necessary panda Python library:
         import pandas as pd

         ## reading the train dataset:
         train=pd.read_csv('SaYoPillow.csv')
```

**2. Seaborn library imported by giving name as “sns”:**

```
In [18]: import seaborn as sns
```

**3. Implemented “info ( )” function to show all information about columns:**

```
In [19]: #info() function to print a concise summary of a DataFrame.
         #i.e. information about a DataFrame including the index dtype
         #and column dtypes,
         #non-null values and memory usage.

         train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 630 entries, 0 to 629
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   sr      630 non-null    float64
 1   rr      630 non-null    float64
 2   t       630 non-null    float64
 3   lm      630 non-null    float64
 4   bo      630 non-null    float64
 5   rem     630 non-null    float64
 6   sr.1    630 non-null    float64
 7   hr      630 non-null    float64
 8   sl      630 non-null    int64
dtypes: float64(8), int64(1)
memory usage: 44.4 KB
```

**4. Preprocessed the data and "value counts ( )" function was defined to return object containing counts of unique values.**

```
In [20]: #Preprocessing:
         # returning a copy of a string. Thus, the old substring remains the same,
         #but a new copy gets created.
         #The label column in this dataset contains labels as 0,1,2,3 an 4.
         #0 means no stress-low stress,
         #1 means medium low stress, 2 means medium stress,
         #3 means medium high stress and 4 means high stress.
         #I have used above mentioned names instead of 0, 1,2,3 and 4
         #and selected the text
         #and label columns for the process of training a machine learning model:

         train.sl=train.sl.replace({0:'low/normal',1:'medium low' , 2: 'medium',
                                   3:'medium high', 4:'high'})

         # defining value_counts() function to return object containing counts
         #of unique values.
         train.sl.value_counts()

Out[20]: high           126
         medium high    126
         low/normal     126
         medium         126
         medium low     126
         Name: sl, dtype: int64
```

**5. "info ( )" function to view all information about columns:**

```
In [21]: train.info()

         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 630 entries, 0 to 629
         Data columns (total 9 columns):
          #   Column  Non-Null Count  Dtype
         ---  ------  --------------  -----
          0   sr      630 non-null    float64
          1   rr      630 non-null    float64
          2   t       630 non-null    float64
          3   lm      630 non-null    float64
          4   bo      630 non-null    float64
          5   rem     630 non-null    float64
          6   sr.1    630 non-null    float64
          7   hr      630 non-null    float64
          8   sl      630 non-null    object
         dtypes: float64(8), object(1)
         memory usage: 44.4+ KB
```

**6. Label encoder was implemented to normalize labels, transform non-numerical labels to numerical labels, fit label encoder and to return encoded labels.**

```
In [47]: #labelencoder convert:
         from sklearn.preprocessing import LabelEncoder

         encoder = LabelEncoder()
         train['sl'] = encoder.fit_transform(train['sl'])
         stress = {index : label for index, label in enumerate(encoder.classes_)}
         stress

Out[47]: {0: 'high', 1: 'low/normal', 2: 'medium', 3: 'medium high', 4: 'medium l
         ow'}
```

**7. Number of unique values were returned for each column:**

```
In [53]: #returning the number of unique values for each column:
         train.nunique()

Out[53]: snoring rate         627
         respiration rate     626
         body temperature     626
         limb movement        626
         blood oxygen         626
         eye movement         626
         sleeping hours       501
         heart rate           626
         stress level           5
         dtype: int64
```

**8. Set shape of dataset:**

```
In [54]: #shape of data
         train.shape

Out[54]: (630, 9)
```

**9. Duplicate datasets were dropped:**

```
In [25]: #dropping duplicate datas:
         train.drop_duplicates(inplace=True)
         train.shape

Out[25]: (630, 9)
```

**10. Summary of all the data of all fields were shown:**

```
In [26]: #Showing summary of all the data of all fields.
         train.head()
```

Out[26]:

| | sr | rr | t | lm | bo | rem | sr.1 | hr | sl |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 93.80 | 25.680 | 91.840 | 16.600 | 89.840 | 99.60 | 1.840 | 74.20 | medium high |
| **1** | 91.64 | 25.104 | 91.552 | 15.880 | 89.552 | 98.88 | 1.552 | 72.76 | medium high |
| **2** | 60.00 | 20.000 | 96.000 | 10.000 | 95.000 | 85.00 | 7.000 | 60.00 | medium low |
| **3** | 85.76 | 23.536 | 90.768 | 13.920 | 88.768 | 96.92 | 0.768 | 68.84 | medium high |
| **4** | 48.12 | 17.248 | 97.872 | 6.496 | 96.248 | 72.48 | 8.248 | 53.12 | low/normal |

**11. The parameters were renamed for better understanding and again the summary of all the data were shown after making changes of names:**

In [27]:
```
#renaming the parameters name for better understanding:
train.rename(columns = {'sr':'snoring rate', 'rr':'respiration rate',
                        't':'body temperature', 'lm':'limb movement',
                        'bo':'blood oxygen', 'rem':'eye movement',
                        'sr.1':'sleeping hours','hr':'heart rate',
                        'sl':'stress level'}, inplace = True)

#Showing summary of all the data after renaming:
train.head()
```

Out[27]:

| | snoring rate | respiration rate | body temperature | limb movement | blood oxygen | eye movement | sleeping hours | heart rate | stress level |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 93.80 | 25.680 | 91.840 | 16.600 | 89.840 | 99.60 | 1.840 | 74.20 | medium high |
| 1 | 91.64 | 25.104 | 91.552 | 15.880 | 89.552 | 98.88 | 1.552 | 72.76 | medium high |
| 2 | 60.00 | 20.000 | 96.000 | 10.000 | 95.000 | 85.00 | 7.000 | 60.00 | medium low |
| 3 | 85.76 | 23.536 | 90.768 | 13.920 | 88.768 | 96.92 | 0.768 | 68.84 | medium high |
| 4 | 48.12 | 17.248 | 97.872 | 6.496 | 96.248 | 72.48 | 8.248 | 53.12 | low/normal |

### 12. Correlation between columns or within data were shown:

```
In [28]: #showing correlation between columns or within data:
         train.corr()
```

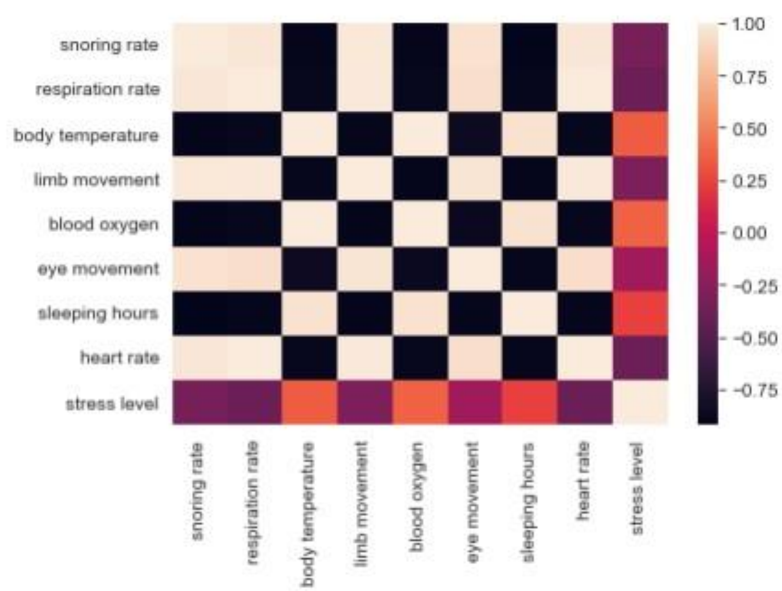
Out[28]:

| | snoring rate | respiration rate | body temperature | limb movement | blood oxygen | eye movement | sleeping hours | hear |
|---|---|---|---|---|---|---|---|---|
| **snoring rate** | 1.000000 | 0.976268 | -0.902475 | 0.981078 | -0.903140 | 0.950600 | -0.920554 | 0.97 |
| **respiration rate** | 0.976268 | 1.000000 | -0.889237 | 0.991738 | -0.889210 | 0.935572 | -0.891855 | 1.0 |
| **body temperature** | -0.902475 | -0.889237 | 1.000000 | -0.896412 | 0.998108 | -0.857299 | 0.954860 | -0.8 |
| **limb movement** | 0.981078 | 0.991738 | -0.896412 | 1.000000 | -0.898527 | 0.964703 | -0.901102 | 0.9 |
| **blood oxygen** | -0.903140 | -0.889210 | 0.998108 | -0.898527 | 1.000000 | -0.862136 | 0.950189 | -0.8 |
| **eye movement** | 0.950600 | 0.935572 | -0.857299 | 0.964703 | -0.862136 | 1.000000 | -0.893952 | 0.9 |
| **sleeping hours** | -0.920554 | -0.891855 | 0.954860 | -0.901102 | 0.950189 | -0.893952 | 1.000000 | -0.8 |
| **heart rate** | 0.976268 | 1.000000 | -0.889237 | 0.991738 | -0.889210 | 0.935572 | -0.891855 | 1.0 |

### 13. After that, train dataframe was plotted into the heatmap:

```
In [43]: #plotting dataframe into heatmap:
         sns.heatmap(data=train.corr())
```

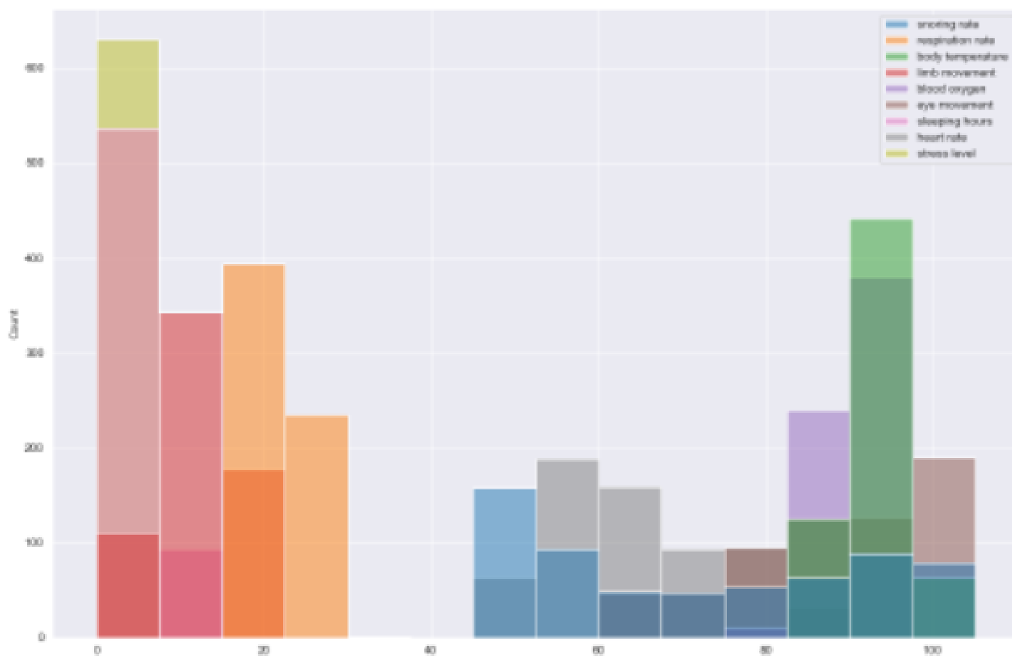
Out[43]: <AxesSubplot:>



**14. Data was visualized by importing import matplotlib.pyplot as plt where size of figure was defined along with the passing of price values and plotting in histogram.**

In [59]:
```
#datavisualizing:
import matplotlib.pyplot as plt

sns.set_style("darkgrid")
plt.figure(figsize=(15, 10)) # defining the size of figure using matplotlib.
sns.histplot(train)  # Passing of price values and plotting in histogram.
plt.show()
```




**15. Summary of all the data wer shown once again after visualizing:**

In [60]:
```
#Showing summary of all the data after visualizing:
train.head()
```

Out[60]:

| | snoring rate | respiration rate | body temperature | limb movement | blood oxygen | eye movement | sleeping hours | heart rate | stress level |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 93.80 | 25.68 | 91.84 | 16.60 | 89.84 | 99.60 | 1.84 | 74.20 | 3 |
| **1** | 91.64 | 25.10 | 91.55 | 15.88 | 89.55 | 98.88 | 1.55 | 72.76 | 3 |
| **2** | 60.00 | 20.00 | 96.00 | 10.00 | 95.00 | 85.00 | 7.00 | 60.00 | 4 |
| **3** | 85.76 | 23.54 | 90.77 | 13.92 | 88.77 | 96.92 | 0.77 | 68.84 | 3 |
| **4** | 48.12 | 17.25 | 97.87 | 6.50 | 96.25 | 72.48 | 8.25 | 53.12 | 1 |

**16. Specifically using KNN algorithm to solve stress detection by importing lazypredict library:**

```
In [32]: #Solving Stress detection using Lazy Predict.
         # importing lazypredict library:
         import lazypredict
```

**17. After that, LazyClassifier was imported:**

```
In [33]: # importing LazyClassifier for classification problem
         #because here I am solving Classification use case:
         from lazypredict.Supervised import LazyClassifier

F:\Anaconda\lib\site-packages\sklearn\utils\deprecation.py:14
3: FutureWarning: The sklearn.utils.testing module is  deprec
ated in version 0.22 and will be removed in version 0.24. The
corresponding classes / functions should instead be imported
from sklearn.utils. Anything that cannot be imported from skl
earn.utils is now part of the private API.
  warnings.warn(message, FutureWarning)
```

**18. "Numpy" library was imported:**

```
In [34]: # importing numpy Python library:
         import numpy as np
```

**19. Imported train_test_split:**

```
In [35]: #spliting dataset into training and testing part
         from sklearn.model_selection import train_test_split
```

**20. "Pop ( )" function was used:**

```
In [15]: # Using pop() and storing the return value:
         y = train.pop('stress level')
         X = train
```

## 21. Prepared train and test splits and splitted dataset into it:

```
In [16]: #Preparing the train and test splits
         from sklearn.model_selection import train_test_split

         #splitting dataset into train and test set
         X_train, X_test, y_train, y_test = train_test_split( X, y, tes
```

## 22. Created object of LazyClassifier Method and performed model fitting in it:

```
In [17]: #creating an object of LazyClassifier class:
         clf = LazyClassifier(verbose=0,predictions=True)

         # model fitting data in LazyClassifier:
         # Here "clf" is returning two values i.e Model and Prediction:
         # model means all the models and with some metrics and
         # prediction means all the predicted value that is ŷ.
         models,predictions = clf.fit(X_train, X_test, y_train, y_test)

         100%|██████████| 29/29 [00:03<00:00,  8.50it/s]
```

## 23. Predicted which model showed more high accuracy value by printing:

```
In [18]: #printing or showing which model did better on Stress detection:
         models
Out[18]:
```

| Model | Accuracy | Balanced Accuracy | ROC AUC | F1 Score | Time Taken |
|---|---|---|---|---|---|
| LinearSVC | 1.00 | 1.00 | None | 1.00 | 0.07 |
| QuadraticDiscriminantAnalysis | 1.00 | 1.00 | None | 1.00 | 0.02 |
| NearestCentroid | 1.00 | 1.00 | None | 1.00 | 0.04 |
| LogisticRegression | 1.00 | 1.00 | None | 1.00 | 0.10 |
| LinearDiscriminantAnalysis | 1.00 | 1.00 | None | 1.00 | 0.03 |
| LabelSpreading | 1.00 | 1.00 | None | 1.00 | 0.08 |
| LabelPropagation | 1.00 | 1.00 | None | 1.00 | 0.05 |
| KNeighborsClassifier | 1.00 | 1.00 | None | 1.00 | 0.04 |
| GaussianNB | 1.00 | 1.00 | None | 1.00 | 0.03 |
| ExtraTreesClassifier | 1.00 | 1.00 | None | 1.00 | 0.32 |
| ExtraTreeClassifier | 1.00 | 1.00 | None | 1.00 | 0.02 |
| SVC | 1.00 | 1.00 | None | 1.00 | 0.04 |
| CalibratedClassifierCV | 1.00 | 1.00 | None | 1.00 | 0.31 |
| NuSVC | 1.00 | 1.00 | None | 1.00 | 0.06 |
| BaggingClassifier | 0.98 | 0.98 | None | 0.98 | 0.08 |
| RandomForestClassifier | 0.98 | 0.98 | None | 0.98 | 0.41 |

| | | | | | |
|---|---|---|---|---|---|
| XGBClassifier | 0.98 | 0.98 | None | 0.98 | 0.25 |
| LGBMClassifier | 0.98 | 0.98 | None | 0.98 | 0.80 |
| DecisionTreeClassifier | 0.98 | 0.98 | None | 0.98 | 0.04 |
| RidgeClassifierCV | 0.96 | 0.96 | None | 0.96 | 0.03 |
| RidgeClassifier | 0.96 | 0.96 | None | 0.96 | 0.05 |
| SGDClassifier | 0.96 | 0.96 | None | 0.96 | 0.03 |
| Perceptron | 0.94 | 0.94 | None | 0.94 | 0.05 |
| PassiveAggressiveClassifier | 0.89 | 0.89 | None | 0.89 | 0.04 |
| BernoulliNB | 0.60 | 0.60 | None | 0.46 | 0.03 |
| AdaBoostClassifier | 0.59 | 0.59 | None | 0.49 | 0.35 |
| DummyClassifier | 0.16 | 0.16 | None | 0.16 | 0.03 |

## 24. Predicted summary data was shown:

In [69]:
```
#Showing predicted summary data result:
predictions.head()
```

Out[69]:

| | AdaBoostClassifier | BaggingClassifier | BernoulliNB | CalibratedClassifierCV | DecisionTreeClassi |
|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | |
| **1** | 1 | 1 | 1 | 1 | |
| **2** | 1 | 1 | 1 | 1 | |
| **3** | 2 | 4 | 1 | 4 | |
| **4** | 2 | 3 | 0 | 3 | |

5 rows × 27 columns

**This is predicted summary data result of Neighbors and Random Forest Classifier between which I will be doing comparison from all other classifiers.**

In [69]:
```
#Showing predicted summary data result:
predictions.head()
```

Out[69]:

| r | GaussianNB | KNeighborsClassifier | ... | PassiveAggressiveClassifier |
|---|---|---|---|---|
| 0 | 0 | 0 | ... | 0 |
| 1 | 1 | 1 | ... | 1 |
| 1 | 1 | 1 | ... | 2 |
| 4 | 4 | 4 | ... | 2 |
| 3 | 3 | 3 | ... | 3 |

```
In [70]: #Showing predicted summary data result:
         predictions.head()
```

Out[70]:

| Analysis | RandomForestClassifier | RidgeClassifier | RidgeClassifierCV | SG |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | |
| 1 | 1 | 1 | 1 | |
| 1 | 1 | 1 | 1 | |
| 4 | 4 | 4 | 4 | |
| 3 | 3 | 3 | 3 | |

**25. Precision, Recall, F1 Score and support score of my trained classification model were shown:**

```
In [20]: # showing the precision, recall, F1 Score and support score of my trained classification model.
         from sklearn.metrics import classification_report
         for i in predictions.columns.tolist():
             print('\t\t',i,'\n')
             print(classification_report(y_test, predictions[i]),'\n')
```

```
                          AdaBoostClassifier

                      precision    recall  f1-score   support

                high       0.96      1.00      0.98        25
          low/normal       1.00      0.96      0.98        25
              medium       0.33      1.00      0.50        25
         medium high       0.00      0.00      0.00        26
          medium low       0.00      0.00      0.00        25

            accuracy                           0.59       126
           macro avg       0.46      0.59      0.49       126
        weighted avg       0.45      0.59      0.49       126


                          BaggingClassifier

                      precision    recall  f1-score   support
```

**Classification Report of KNeighbors:**

KNeighborsClassifier

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| high | 1.00 | 1.00 | 1.00 | 25 |
| low/normal | 1.00 | 1.00 | 1.00 | 25 |
| medium | 1.00 | 1.00 | 1.00 | 25 |
| medium high | 1.00 | 1.00 | 1.00 | 26 |
| medium low | 1.00 | 1.00 | 1.00 | 25 |
| accuracy | | | 1.00 | 126 |
| macro avg | 1.00 | 1.00 | 1.00 | 126 |
| weighted avg | 1.00 | 1.00 | 1.00 | 126 |

**Classification Report of RandomForest:**

RandomForestClassifier

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| high | 1.00 | 1.00 | 1.00 | 25 |
| low/normal | 1.00 | 0.96 | 0.98 | 25 |
| medium | 1.00 | 0.96 | 0.98 | 25 |
| medium high | 0.96 | 1.00 | 0.98 | 26 |
| medium low | 0.96 | 1.00 | 0.98 | 25 |
| accuracy | | | 0.98 | 126 |
| macro avg | 0.98 | 0.98 | 0.98 | 126 |
| weighted avg | 0.98 | 0.98 | 0.98 | 126 |

## References

DUTTA, G., n.d. *human-stress-detection-lazy-predict.* [Online] Available at: https://www.kaggle.com/code/gauravduttakiit/human-stress-detection-lazy-predict
[Accessed 07 03 2022].