Stress Detection

Ayesha Nusrat
Department of Statistics and O.R.
Aligarh Muslim University

March 21, 2025

Abstract

Stress is a major factor affecting mental and physical health, impacting productivity and overall well-being. Traditional stress assessment methods rely on self-reports and clinical evaluations, which can be subjective and time-consuming. With advancements in machine learning, automated stress detection using physiological data has gained importance.

This study aims to classify stress levels using biometric features such as heart rate, respiration rate, snoring rate, body temperature, blood oxygen levels, eye movement, limb movement, and sleep duration. The dataset was sourced from Kaggle and included balanced stress levels ranging from 0 (No Stress) to 4 (High Stress). Data augmentation was applied to increase dataset size but was not needed for class balancing.

We implemented and evaluated six machine learning models: Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Logistic Regression, Naive Bayes, Random Forest, and CatBoost. Each model was trained and tested to determine its effectiveness in stress detection.

Data preprocessing included handling missing values, managing outliers, correlation analysis, and computing descriptive statistics. The models were evaluated using accuracy, precision, recall, and F1-score. Results were summarized and visualized for easy comparison.

Findings showed that KNN achieved the highest accuracy, followed by Random Forest and CatBoost. Traditional models like Naive Bayes, Logistic Regression, and SVC performed slightly lower, indicating that tree-based and distance-based methods were more effective for this task.

The study demonstrates the potential of machine learning for real-time stress monitoring, paving the way for AI-driven mental health solutions and wearable health technologies.

Contents

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Problem Statement . . . 1
- 1.3 Objectives . . . 1
- 1.4 Scope of the Project . . . 2

2 Dataset Description **3**
- 2.1 Features and Labels . . . 3

3 Data Analysis and Preprocessing **5**
- 3.1 Handling Missing Values . . . 5
- 3.2 Managing Outliers . . . 5
- 3.3 Descriptive Statistics . . . 5
- 3.4 Correlation Analysis . . . 6
- 3.5 Dataset Split . . . 6
- 3.6 Data Augmentation . . . 6

4 Model Training and Evaluation **8**
- 4.1 Model Fitting . . . 8
- 4.2 Hyperparameter Tuning . . . 8
- 4.3 Cross-Validation . . . 9
- 4.4 Performance Metrics . . . 9

5 Model Comparison **10**
- 5.1 Observations . . . 10

6 Conclusion **12**
- 6.1 Limitations of the Study . . . 12
- 6.2 Future Scope . . . 12
- 6.3 Reference . . . 13

A Python Code and Outputs **14**
- A.1 Importing libraries . . . 14
- A.2 Loading dataset . . . 14
- A.3 Handling missing values . . . 15
- A.4 Managing outliers . . . 16
- A.5 Descriptive statistics . . . 19
- A.6 Correlation . . . 19
- A.7 Creating test data . . . 19

A.8 Data augmentation 20
A.9 Data subsets for model fitting and evaluation 20
A.10 Hyperparameter tuning 21
A.11 Model fitting and evaluation 23
A.12 Cross validation 24
A.13 Classification report 24
A.14 Confusion matrix 25
A.15 Model performance 26
A.15.1 Model performance at different stress levels 27

Chapter 1

Introduction

1.1 Background

Stress is a growing health concern that affects both mental and physical well-being. Chronic stress has been linked to serious health conditions such as heart disease, anxiety disorders, immune system dysfunction, and sleep disturbances. Traditional stress assessment methods, such as self-reported surveys and psychological questionnaires, rely on subjective responses, which can be inconsistent, biased, and impractical for real-time monitoring.

Advancements in machine learning and physiological monitoring now enable automated stress detection using biometric data collected from wearable sensors and medical devices. Physiological indicators such as heart rate, respiration rate, snoring rate, body temperature, blood oxygen levels, eye movement, limb movement, and sleeping patterns provide valuable insights into a person's stress level. Machine learning models can analyze these signals to detect and classify stress more objectively and accurately than traditional methods.

1.2 Problem Statement

The goal of this project is to develop a machine learning-based stress detection system that predicts stress levels ranging from 0 (No Stress) to 4 (High Stress) based on biometric data. The dataset, obtained from Kaggle, consists of 630 observations and 9 features, including physiological indicators such as heart rate, respiration rate, snoring rate, body temperature, blood oxygen levels, eye movement, limb movement, and sleeping patterns. Various machine learning models will be implemented and evaluated to determine their effectiveness in stress detection.

1.3 Objectives

The key objectives of this project are:

- Train SVC, KNN, Logistic Regression, Random Forest, Naive Bayes, and CatBoost model for fitting approach.
- Perform hyperparameter tuning to optimize selected models.

- Apply cross-validation for a generalized model evaluation.
- Conduct data analysis including missing value handling, outlier management, correlation analysis, and descriptive statistics.
- Summarize and visualize final results for easy comparison.

1.4 Scope of the Project

This study is designed with specific boundaries to ensure a **focused and well-defined approach** to stress detection using machine learning. The scope includes the following key aspects:

- **Biometric-Based Stress Detection**: The project solely relies on biometric data such as heart rate, respiration rate, body temperature, and blood oxygen levels. Psychological surveys or self-reported assessments are not considered.
- **Supervised Machine Learning Techniques**: The study implements supervised learning algorithms to classify stress levels. Unsupervised and reinforcement learning techniques are not explored.
- **Pre-Collected Dataset**: The dataset used in this project is sourced from Kaggle and contains pre-recorded biometric readings. The study does not involve real-time data collection from wearable devices or live monitoring systems.
- **Exclusion of Deep Learning Models**: While deep learning methods such as neural networks could be applied to stress detection, this study focuses on traditional machine learning algorithms for better interpretability and computational efficiency.

Chapter 2

Dataset Description

2.1 Features and Labels

The dataset utilized in this study was obtained from **Kaggle** and comprises **630 observations** with multiple biometric features. It is specifically designed for **stress level classification**, where the target variable represents **five distinct stress levels**:

- **0** – No Stress
- **1** – Low Stress
- **2** – Moderate Stress
- **3** – High Stress
- **4** – Severe Stress

The dataset is **balanced**, ensuring that each stress level has a proportionate number of samples, reducing the risk of model bias toward any particular class.

The features used for prediction include various **physiological and behavioral metrics**, such as:

- **Heart Rate (bpm)** – Measures cardiovascular activity under stress.
- **Limb Movement** – Indicates restlessness, which can be associated with stress.
- **Respiration Rate (breaths/min)** – Affected by stress-induced changes in breathing patterns.
- **Body Temperature (°F)** – Can fluctuate due to stress responses.
- **Eye Movement** – Tracks focus and stress-related eye behavior.
- **Blood Oxygen (SpO%)** – Changes under stress due to altered breathing patterns.
- **Snoring Rate** – Can indicate irregular sleep patterns often linked to stress.
- **Sleeping Hours** – Sleep deprivation is a key indicator of stress.

To improve model performance, **data augmentation** was applied **only to the training set** by introducing **small noise and scaling variations to numerical features**. However, augmentation was performed **only to increase the size of the data set** and was **not required for class balancing**, since the data set already contained an equal distribution of stress levels.

This well-balanced dataset enables **fair evaluation** of machine learning models, ensuring that no single stress level is underrepresented.

Chapter 3

Data Analysis and Preprocessing

3.1 Handling Missing Values

Handling missing values is crucial to ensure data consistency and prevent bias in machine learning models. In this dataset, missing values were checked using exploratory data analysis (EDA). Since the data set contains biometric characteristics, **the median was chosen as the imputation method for the numerical attributes**. This is because the median is less sensitive to extreme values compared to the mean, making it a robust choice for handling missing data.

3.2 Managing Outliers

Outliers can significantly affect model performance by skewing predictions. The outliers were first checked and analyzed through a boxplot and conclusion was made that since physiological features usually vary, these outliers play a vital role for the model; therefore, they were kept except for some features like body temperature, eye movement, blood oxygen, and heart rate because they have some unrealistic values. To manage these outliers, the **Interquartile Range (IQR)** method was used:

- The first quartile ($Q1$) and the third quartile ($Q3$) were calculated for each numeric feature.
- The interquartile range was calculated as $IQR = Q3 - Q1$.
- A threshold was set: Any data point outside the range $[Q1 - 1.5 \times IQR, Q3 + 1.5 \times IQR]$ was considered an outlier.
- Outliers were replaced with the median value of the respective feature.

This approach ensures that extreme values do not disproportionately influence model predictions while maintaining data consistency.

3.3 Descriptive Statistics

Descriptive statistics help to understand the distribution and central tendencies of each feature in the data set. The following statistical measures were computed:

- **Mean:** Provides the average value of each feature.
- **Median:** Represents the middle value, useful for skewed distributions.
- **Standard Deviation:** Indicates the spread of values around the mean.
- **Quartiles:** Divide a dataset into four equal parts, with Q1 (25th percentile), Q2 (median, 50th percentile), and Q3 (75th percentile) marking the values below which 25%, 50%, and 75% of the data fall, respectively.
- **Minimum and Maximum Values:** Show the range of each feature.

This analysis provided insights into potential data imbalances, deviations, and general trends in the dataset.

3.4 Correlation Analysis

Correlation analysis was performed to examine relationships between different features. A **correlation matrix** was generated using Pearson's correlation coefficient (ρ), which measures the strength and direction of relationships between numerical variables:

- $\rho > 0.7$ indicates a strong positive correlation.
- $\rho < -0.7$ indicates a strong negative correlation.
- $\rho \approx 0$ suggests no correlation.

3.5 Dataset Split

To train and evaluate the machine learning models, the dataset was split into training and testing sets:

- **75% of the data** was used for training.
- **25% of the data** was reserved for testing.

This split ensures that models generalize well on unseen data and do not overfit.

3.6 Data Augmentation

To improve model robustness and generalization, **data augmentation** was applied only to the training set. The augmentation techniques used were:

- **Jittering:** Small random **Gaussian noise** was added to numerical features to introduce slight variations in data values.
- **Scaling:** Each feature was multiplied by a random factor within a specified range to create slight distortions.

Important Considerations:

- Augmentation was applied **only to numerical features**, ensuring that categorical variables and labels remained unaffected.
- The process increased the dataset size, enhancing the model's ability to generalize across unseen variations.
- The augmented data was combined with the original training set to create a richer dataset for model training.

This method helped improve the model's robustness by introducing slight variations that mimic real-world biometric fluctuations.

Chapter 4

Model Training and Evaluation

4.1 Model Fitting

To train and compare different machine learning models, a unified approach was adopted where all models were trained together in a single execution loop. The models used in this study include:

- **Support Vector Machine (SVM)**: A robust classifier that finds the optimal hyperplane to separate different stress levels.
- **k-Nearest Neighbors (KNN)**: A distance-based algorithm that classifies an instance based on its closest neighbors.
- **Logistic Regression**: A linear model commonly used for classification tasks.
- **Naive Bayes**: A probabilistic model based on Bayes' theorem, often effective for small datasets.
- **Random Forest**: An ensemble learning method using multiple decision trees to improve accuracy.
- **CatBoost**: A gradient boosting algorithm optimized for categorical data.

Each model was trained using the preprocessed dataset, and their performances were compared based on standard evaluation metrics.

4.2 Hyperparameter Tuning

To enhance model performance, hyperparameter tuning was conducted for selected models using:

- **Grid Search**: Exhaustively searches through a specified set of hyperparameters to find the best combination.
- **Random Search**: Randomly samples hyperparameters within a given range to efficiently find optimal values.

Hyperparameter tuning was applied particularly to models that benefit significantly from parameter optimization, such as SVM, Random Forest, and CatBoost. The best hyperparameter configurations were selected based on cross-validation performance.

4.3 Cross-Validation

To ensure that the models generalize well to unseen data, **5-fold cross-validation** was implemented. This method works as follows:

- The dataset is randomly divided into five equal-sized folds.
- Each model is trained on four folds and tested on the remaining fold.
- The process repeats five times, ensuring that each data point is used for both training and testing.
- The average performance across all folds is reported as the final evaluation metric.

This approach reduces overfitting and ensures that the results are not biased by a particular train-test split.

4.4 Performance Metrics

To assess model performance, several evaluation metrics were used:

- **Accuracy**: Measures the overall proportion of correctly classified stress levels.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

- **Precision**: Indicates the proportion of correct positive predictions out of all predicted positives.

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

- **Recall (Sensitivity)**: Measures the model's ability to identify stressed individuals.

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

- **F1-Score**: A harmonic mean of precision and recall, balancing false positives and false negatives.

$$F1\text{-}Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4.4}$$

These metrics were chosen to provide a comprehensive evaluation of model performance, particularly in distinguishing between different stress levels.

Chapter 5

Model Comparison

After training and evaluation, the models were systematically compared based on their overall performance across key metrics, including accuracy, precision, recall, F1-score. The results were visualized using bar charts and confusion matrices to highlight differences in classification performance. Additionally, a summary table was created for easy interpretation, presenting a side-by-side comparison of each model's strengths and weaknesses. This approach helped identify the most effective classifier for stress detection while considering trade-offs between precision and recall, particularly in distinguishing higher stress levels.

	Model	Accuracy%	Precision%	Recall%	F1-Score%
0	KNN	100.00	100.0	100.0	100.0
1	RandomForest	99.37	99.4	99.4	99.4
2	Logistic	95.57	96.0	95.6	95.6
3	SVC	97.47	97.8	97.6	97.8
4	NaiveBayes	96.20	96.4	96.2	96.4
5	CatBoost	98.73	98.8	98.8	98.8

Figure 5.1: Summary

5.1 Observations

The following key insights were derived from the model comparison:

- **KNN achieved the highest accuracy (100%)**, outperforming all other models. This suggests that the stress detection dataset may contain well-separated classes, making KNN's distance-based classification highly effective.
- **Random Forest and CatBoost closely followed, with accuracies of 99.37% and 98.73% respectively.** These models leverage ensemble learning, which com-

bines multiple decision trees to reduce overfitting and improve robustness. The strong performance of these models confirms that stress levels can be effectively identified using tree-based methods.

- **Naive Bayes performed slightly lower than ensemble-based models (96.2%).** This could be due to the assumption of feature independence, which may not hold true for biometric data. Since stress-related features like heart rate, respiration rate, and limb movement are often correlated, Naive Bayes may struggle with capturing complex feature interactions.

- **Support Vector Machine (SVM) showed strong performance (97.47%) but was slightly behind tree-based models.** SVM works well in high-dimensional spaces, but its performance can be affected by feature scaling and parameter selection.

- **Logistic Regression achieved the lowest accuracy (95.57%) among the models.** Since logistic regression is a linear model, it might struggle with capturing the non-linear relationships between stress and biometric features. This suggests that more complex models are better suited for stress classification.

- **Overall, ensemble learning and distance-based methods performed best.** KNN, Random Forest, and CatBoost outperformed traditional models like Logistic Regression, Naive Bayes, and SVM. The results indicate that for biometric-based stress classification, models that can handle complex, non-linear relationships and interactions among features tend to yield superior performance.

- **Trade-offs between models should be considered based on application needs.** While KNN achieved perfect accuracy, it can be computationally expensive for large datasets. Random Forest and CatBoost offer a balance between accuracy and efficiency, making them practical choices for real-time applications.

The results highlight the importance of selecting appropriate models based on dataset characteristics and the intended application of stress detection systems.

Chapter 6

Conclusion

This study explored biometric-based stress detection using machine learning models. Through comprehensive data analysis, preprocessing, and model evaluation, valuable insights were obtained regarding the effectiveness of various classification techniques. The key findings of the study are summarized as follows:

- Data preprocessing steps such as **handling missing values, outlier management, and correlation analysis** improved data quality, ensuring reliable model performance.
- **KNN achieved the highest accuracy (100%)**, followed by **Random Forest (99.37%)** and **CatBoost (98.73%)**, indicating that non-linear models and ensemble methods are highly effective for stress detection.
- The use of **cross-validation** ensured that model performance was consistent across different data splits, preventing overfitting and improving generalization.

6.1 Limitations of the Study

Although the study produced promising results, several limitations must be acknowledged:

- **Dataset Dependency:** The dataset used in this research was sourced from Kaggle, and its applicability to broader, real-world populations remains uncertain. Differences in data collection conditions could impact model generalization.
- **Synthetic Data Augmentation:** Data augmentation was performed using synthetic noise addition and scaling, which may not fully capture real-world variations in biometric stress responses.
- **Feature Constraints:** The dataset primarily consists of biometric signals. Including additional behavioral and psychological factors might enhance stress prediction accuracy.

6.2 Future Scope

Future work can build upon this study by incorporating the following advancements:

- **Expanding Dataset:** Collecting and integrating more diverse real-world biometric data will enhance model robustness and improve generalization to varied demographics.
- **Real-Time Stress Monitoring:** Developing a real-time stress detection system that integrates with wearable devices and mobile applications can provide continuous stress assessment for practical applications.
- **Exploring Deep Learning Models:** Future studies could investigate deep learning architectures such as LSTMs and CNNs to better capture temporal and spatial dependencies in biometric data.
- **Multimodal Stress Detection:** Combining biometric data with textual, audio, or behavioral cues (e.g., speech patterns and facial expressions) may improve stress detection accuracy.

6.3 Reference

Kaggle, "Stress Detection Dataset," Available at: `https://www.kaggle.com/datasets/dheerov/stress-detection-dataset`.

Appendix A

Python Code and Outputs

A.1 Importing libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score,
    classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from catboost import CatBoostClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from prettytable import PrettyTable
```

A.2 Loading dataset

```
data = pd.read_csv("/content/data_stress.csv")
df=pd.DataFrame(data)
# Renaming the colums
df.columns=['snoring_rate', 'respiration_rate', 'body_temperature', '
    limb_movement', 'blood_oxygen','eye_movement', 'sleeping_hours', '
    heart_rate', 'stress_level']
```

A.3 Handling missing values

```
df.isnull().sum()
```

Feature	Count of missing values
snoring _rate	0
respiration _rate	0
body _temperature	16
limb _movement	12
blood _oxygen	4
eye _movement	18
sleeping _hours	11
heart _rate	24
stress _level	0

```
# Checking distribution of data
df.hist(bins=30,figsize=(12,10),color="thistle")
```

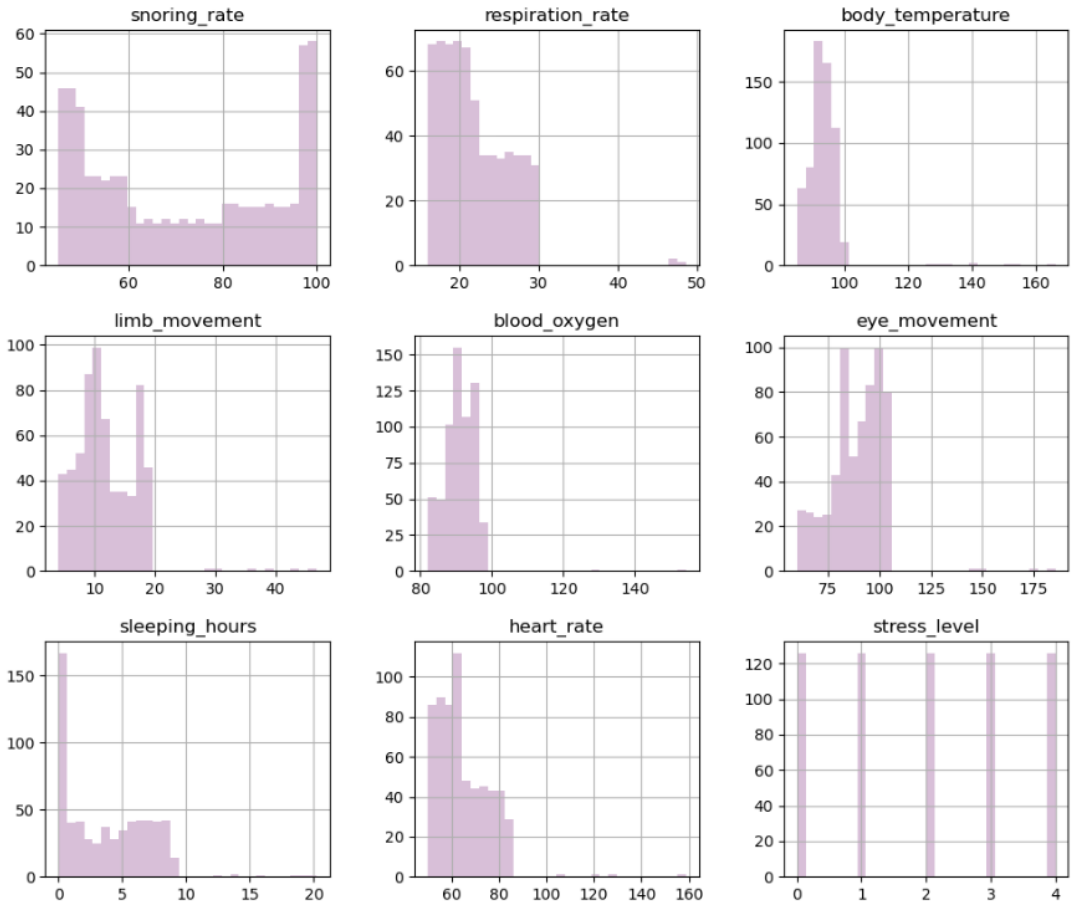


Figure A.1: Histogram

```
# Since attributes having missing values are highly skewed, therefore
    we use median to fill them.
df.fillna(df.median(numeric_only=True), inplace=True)
```

```
df.isnull().sum()
```

Feature	Count of missing values
snoring _rate	0
respiration _rate	0
body _temperature	0
limb _movement	0
blood _oxygen	0
eye _movement	0
sleeping _hours	0
heart _rate	0
stress _level	0

A.4 Managing outliers

```
sns.set_style("whitegrid")
plt.figure(figsize=(12, 6))
sns.boxplot(data=df)
plt.xticks(rotation=45)
plt.title("Boxplot of Stress Detection Features")
plt.show()
```

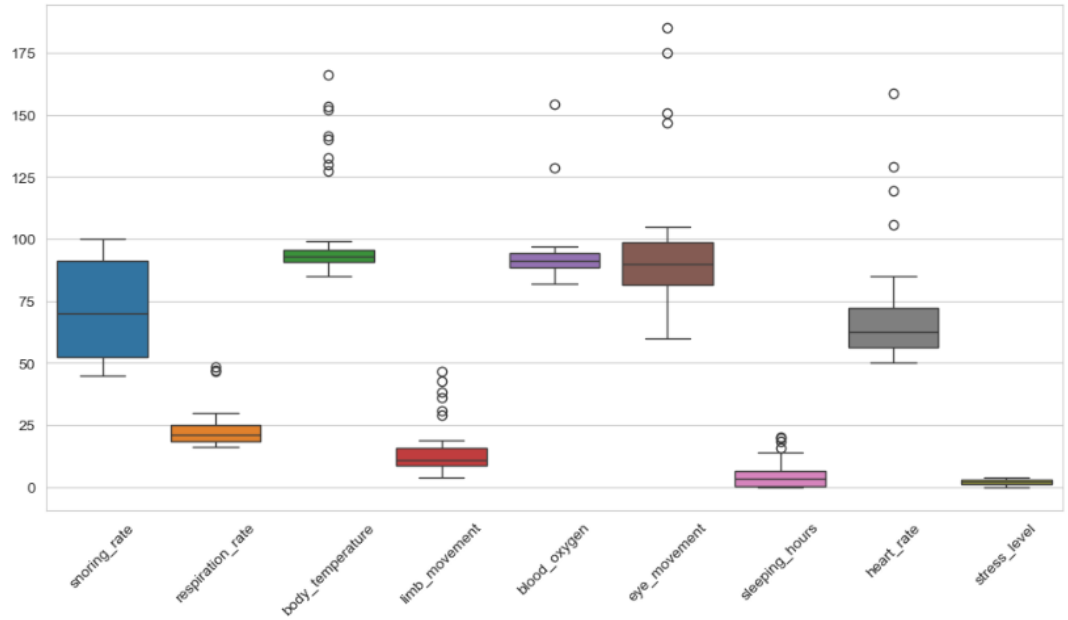


Figure A.2: Boxplot of Stress Detection Features

There are many outliers! But for stress detection, physiological features are very important and usually vary. So, these outliers play a vital role for our model; therefore, we should keep them except for "body temperature", "eye movement", "blood oxygen", and "heart rate" because they have some unrealistic values.

```
# Outliers in body temperatue
Q1 = df["body_temperature"].quantile(0.25)
Q3 = df["body_temperature"].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df[(df["body_temperature"] < lower_bound) |(df["
    body_temperature"] > upper_bound)]
print("Number␣of␣outliers:", len(outliers))

# Replacing those outliers
df["body_temperature"] = np.clip(df["body_temperature"], lower_bound,
    upper_bound)
median_temp = df["body_temperature"].median()
df.loc[df["body_temperature"] > upper_bound, "body_temperature"] =
    median_temp
df.loc[df["body_temperature"] < lower_bound, "body_temperature"] =
    median_temp
```

Number of outliers: 8

```
# Outliers in eye movement
Q1 = df["eye_movement"].quantile(0.25)
Q3 = df["eye_movement"].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df[(df["eye_movement"] < lower_bound) | (df["eye_movement"]
    > upper_bound)]
print("Number␣of␣outliers:", len(outliers))

# Replacing those outliers
df["eye_movement"] = np.clip(df["eye_movement"], lower_bound,
    upper_bound)
median_eye_movement = df["eye_movement"].median()
df.loc[df["eye_movement"] > upper_bound, "eye_movement"] =
    median_eye_movement
df.loc[df["eye_movement"] < lower_bound, "eye_movement"] =
    median_eye_movement
```

Number of outliers: 4

```
# Outliers in blood oxygen
Q1 = df["blood_oxygen"].quantile(0.25)
Q3 = df["blood_oxygen"].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df[(df["blood_oxygen"] < lower_bound) | (df["blood_oxygen"]
    > upper_bound)]
print("Number␣of␣outliers:", len(outliers))

# Replacing those outliers
```

```
df["blood_oxygen"] = np.clip(df["blood_oxygen"], lower_bound,
    upper_bound)
median_blood_oxygen = df["blood_oxygen"].median()
df.loc[df["blood_oxygen"] > upper_bound, "blood_oxygen"] =
    median_blood_oxygen
df.loc[df["blood_oxygen"] < lower_bound, "blood_oxygen"] =
    median_blood_oxygen
```

Number of outliers: 2

```
# Outliers in heart rate
Q1 = df["heart_rate"].quantile(0.25)
Q3 = df["heart_rate"].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df[(df["heart_rate"] < lower_bound) | (df["heart_rate"] >
    upper_bound)]
print("Number of outliers:", len(outliers))

# Replacing those outliers
df["heart_rate"] = np.clip(df["heart_rate"], lower_bound, upper_bound)
median_heart_rate = df["heart_rate"].median()
df.loc[df["heart_rate"] > upper_bound, "heart_rate"] =
    median_heart_rate
df.loc[df["heart_rate"] < lower_bound, "heart_rate"] =
    median_heart_rate
```

Number of outliers: 4

```
sns.set_style("whitegrid")
plt.figure(figsize=(12, 6))
sns.boxplot(data=df)
plt.xticks(rotation=45)
plt.title("Boxplot of Stress Detection Features")
plt.show()
```

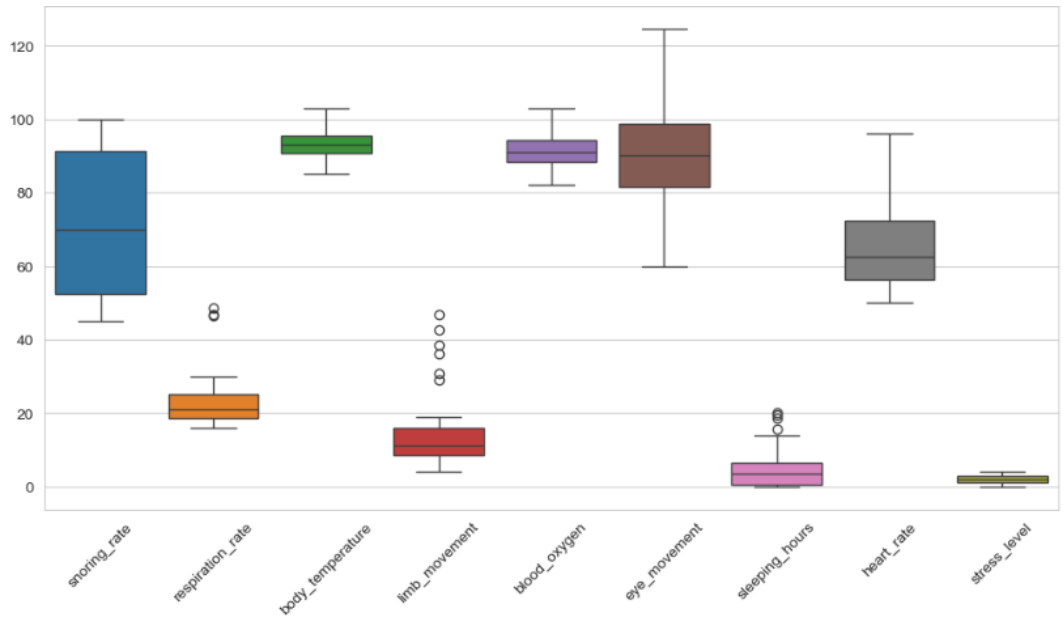


Figure A.3: Boxplot

A.5 Descriptive statistics

```
df.describe()
```

	snoring_rate	respiration_rate	body_temperature	limb_movement	blood_oxygen	eye_movement	sleeping_hours	heart_rate	stress_level
count	630.000000	630.000000	630.000000	630.000000	630.000000	630.000000	630.000000	630.000000	630.000000
mean	71.600000	21.916314	92.951733	11.928098	90.925076	88.743302	3.831765	64.606508	2.000000
std	19.372833	4.336242	3.619973	4.954834	3.949561	12.108409	3.312105	10.013359	1.415337
min	45.000000	16.000000	85.000000	4.000000	82.000000	60.000000	0.000000	50.000000	0.000000
25%	52.500000	18.500000	90.660000	8.564000	88.500000	81.410000	0.516000	56.450000	1.000000
50%	70.000000	21.016000	93.080000	11.048000	91.000000	90.080000	3.608000	62.540000	2.000000
75%	91.250000	25.064000	95.532000	15.830000	94.250000	98.710000	6.548000	72.260000	3.000000
max	100.000000	48.560000	102.840000	46.800000	102.875000	124.660000	20.220000	95.975000	4.000000

Figure A.4: Descriptive Statistics of the data

```
df["stress_level"].value_counts()
```

Stress Level	Count
0	126
1	126
2	126
3	126
4	126

A.6 Correlation

```
df.corr()
```

	snoring_rate	respiration_rate	body_temperature	limb_movement	blood_oxygen	eye_movement	sleeping_hours	heart_rate	stress_level
snoring_rate	1.000000	0.905672	-0.808273	0.827464	-0.882360	0.895499	-0.844455	0.927907	0.975322
respiration_rate	0.905672	1.000000	-0.735787	0.768604	-0.806599	0.823448	-0.770070	0.875359	0.893639
body_temperature	-0.808273	-0.735787	1.000000	-0.685052	0.882107	-0.720650	0.788783	-0.762959	-0.864411
limb_movement	0.827464	0.768604	-0.685052	1.000000	-0.751516	0.760464	-0.709198	0.790927	0.821295
blood_oxygen	-0.882360	-0.806599	0.882107	-0.751516	1.000000	-0.791391	0.852912	-0.833774	-0.941144
eye_movement	0.895499	0.823448	-0.720650	0.760464	-0.791391	1.000000	-0.771200	0.841092	0.899183
sleeping_hours	-0.844455	-0.770070	0.788783	-0.709198	0.852912	-0.771200	1.000000	-0.791567	-0.891761
heart_rate	0.927907	0.875359	-0.762959	0.790927	-0.833774	0.841092	-0.791567	1.000000	0.919268
stress_level	0.975322	0.893639	-0.864411	0.821295	-0.941144	0.899183	-0.891761	0.919268	1.000000

Figure A.5: Correlation matrix

A.7 Creating test data

```
train, test = train_test_split(df,test_size=0.25,random_state=101)
```

A.8 Data augmentation

```python
# Augmentation Function
def augment_data(train, noise_level=0.01, scale_range=(0.9, 1.1)):
    train_aug = train.copy()

    # Apply only on numeric columns
    numeric_cols = train.select_dtypes(include=[np.number]).columns

    np.random.seed(1)
    for col in numeric_cols:
        # Add noise (jittering)
        train_aug[col] += np.random.normal(0, noise_level, size=len(
            train))

        # Scale up/down slightly
        scale_factor = np.random.uniform(scale_range[0], scale_range
            [1])
        train_aug[col] *= scale_factor

    return train_aug

# Apply augmentation
df_augmented = augment_data(train)

# Combine original + augmented data
df_combined = pd.concat([train, df_augmented], ignore_index=True)
```

```python
# Checking distribution of stress levels
a=df_combined["stress_level"]
a=a.astype(int)
a.value_counts()
```

Stress Level	Count
0	198
1	194
2	193
3	180
4	179

The distribution is pretty much balanced.

A.9 Data subsets for model fitting and evaluation

```python
Y_train = df_combined["stress_level"]
X_train = df_combined.drop(["stress_level"], axis= 1)
Y_test = test["stress_level"]
X_test = test.drop(["stress_level"], axis= 1)
```

```python
Y_train=Y_train.astype(int)
```

A.10 Hyperparameter tuning

```
# For K-Nearest Neighbours

# Define parameter grid
param_grid = {
    "n_neighbors": range(1, 31),
    "weights": ["uniform", "distance"],
    "metric": ["euclidean", "manhattan", "minkowski"]
}

# Initialize KNN model
knn = KNeighborsClassifier()

# Perform GridSearchCV
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring="accuracy",
    n_jobs=-1, verbose=2)
grid_search.fit(X_train, Y_train)

# Best parameters
print("Best Parameters:", grid_search.best_params_)
```

Fitting 5 folds for each of 180 candidates, totalling 900 fits
Best Parameters: 'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'uniform'

```
# For Random Forest

# Define the parameter grid
param_dist = {
    "n_estimators": np.arange(100, 1000, 50),
    "max_depth": np.arange(5, 50, 2),
    "min_samples_split": np.arange(1, 20),
    "min_samples_leaf": np.arange(1, 10),
    "max_features": ["sqrt", "log2", None],
    "bootstrap": [True, False]
}

# Create Random Forest model
rf = RandomForestClassifier(random_state=42)

# RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=rf, param_distributions=
    param_dist, n_iter=20, cv=5, scoring='accuracy', n_jobs=-1, verbose
    =2, random_state=42)

# Fit the model
random_search.fit(X_train, Y_train)

# Best parameters
print("Best Parameters:", random_search.best_params_)
```

Fitting 5 folds for each of 20 candidates, totalling 100 fits
Best Parameters: 'n _estimators': 750, 'min _samples _split': 7, 'min _samples _leaf': 4, 'max _features': 'sqrt', 'max _depth': 19, 'bootstrap': False

```
# For Support Vector Classifier

# Define the parameter grid
param_grid = {
    "C": [0.1, 1, 10],
    "kernel": ["linear", "rbf"],
    "gamma": ["scale", "auto", 0.01, 0.1]
}

# Initialize SVC
svc = SVC()

# Perform Grid Search
grid_search = GridSearchCV(svc, param_grid, cv=5, scoring="accuracy",
    n_jobs=-1, verbose=1)

# Fit model
grid_search.fit(X_train, Y_train)

# Best Parameters
print("Best Parameters:", grid_search.best_params_)
```

Fitting 5 folds for each of 24 candidates, totalling 120 fits
Best Parameters: 'C': 1, 'gamma': 0.01, 'kernel': 'rbf'

```
# For CatBoost Classifier

# Define parameter grid
param_grid = {
    "iterations": [200, 500],
    "learning_rate": [0.03, 0.1],
    "depth": [4, 6, 8],
    "l2_leaf_reg": [1, 3, 5],
}

# Initialize CatBoostClassifier
catboost = CatBoostClassifier(verbose=0)

# Perform Grid Search
grid_search = GridSearchCV(catboost, param_grid, cv=5, scoring="
    accuracy", n_jobs=-1, verbose=2)

# Fit model
grid_search.fit(X_train, Y_train)

# Best parameters
print("Best Parameters:", grid_search.best_params_)
```

Fitting 5 folds for each of 36 candidates, totalling 180 fits
Best Parameters: 'depth': 8, 'iterations': 200, 'l2 _leaf _reg': 1, 'learning _rate': 0.03

A.11 Model fitting and evaluation

```
models = {
    "SVC": SVC(kernel='rbf', C=1, gamma=0.01),
    "KNN": KNeighborsClassifier(metric="manhattan", n_neighbors=5,
       weights="uniform"),
    "Logistic Regression": LogisticRegression(),
    "Random Forest": RandomForestClassifier(n_estimators=750, criterion
       ='gini', max_features="sqrt", min_samples_split=7,
       min_samples_leaf=4, max_depth=19, random_state=42),
    "Naive Bayes": GaussianNB(),
    "CatBoost": CatBoostClassifier(random_state=42, iterations=200,
       depth=8, learning_rate=0.03, l2_leaf_reg=1)
}

# Train and evaluate models
accuracy_results = []
for name, model in models.items():
    model.fit(X_train, Y_train)
    Y_pred_train = model.predict(X_train)
    Y_pred_test = model.predict(X_test)
    train_accuracy = accuracy_score(Y_train, Y_pred_train)
    test_accuracy = accuracy_score(Y_test, Y_pred_test)
    accuracy_results.append([name, train_accuracy, test_accuracy])
```

```
pd.DataFrame(accuracy_results,columns=["model","Train Accuracy","Test
    Accuracy"])
```

	model	Train Accuracy	Test Accuracy
0	SVC	0.993644	0.974684
1	KNN	0.993644	1.000000
2	Logistic Regression	0.978814	0.955696
3	Random Forest	0.994703	0.993671
4	Naive Bayes	0.978814	0.962025
5	CatBoost	0.994703	0.987342

A.12 Cross validation

```
cv_results = {}

for name, model in models.items():
    scores = cross_val_score(model, X_train, Y_train, cv=5, scoring="
        accuracy", n_jobs=-1)
    cv_results[name] = (np.mean(scores), np.std(scores))
    print(f"{name}: Mean Accuracy = {np.mean(scores):.4f}   {np.std(
        scores):.4f}")
```

KNN: Mean Accuracy = 0.9936 ± 0.0085
RandomForest: Mean Accuracy = 0.9936 ± 0.0085
Logistic: Mean Accuracy = 0.9650 ± 0.0110
SVC: Mean Accuracy = 0.9883 ± 0.0091
NaiveBayes: Mean Accuracy = 0.9608 ± 0.0179
CatBoost: Mean Accuracy = 0.9947 ± 0.0067

A.13 Classification report

```
# Initialize a table
table = PrettyTable()
table.field_names = ["Model", "Class", "Precision", "Recall", "F1-score
    "]

for name, model in models.items():
    Y_pred = model.predict(X_test)
    report = classification_report(Y_test, Y_pred, output_dict=True)

    # Iterate over class-wise metrics
    for cls in range(5):
        precision = round(report[str(cls)]["precision"], 2)
        recall = round(report[str(cls)]["recall"], 2)
        f1 = round(report[str(cls)]["f1-score"], 2)
        table.add_row([name, cls, precision, recall, f1])
    table.add_row(["-" * 10, "-" * 5, "-" * 9, "-" * 6, "-" * 8])

# Print formatted table
print("\nClassification Report Summary:\n")
print(table)
```

Model	Class	Precision	Recall	F1-score
SVC	0	1.0	0.94	0.97
SVC	1	1.0	1.0	1.0
SVC	2	0.97	0.97	0.97
SVC	3	0.92	0.97	0.95
SVC	4	1.0	1.0	1.0
KNN	0	1.0	1.0	1.0
KNN	1	1.0	1.0	1.0
KNN	2	1.0	1.0	1.0
KNN	3	1.0	1.0	1.0
KNN	4	1.0	1.0	1.0
Logistic Regression	0	1.0	0.88	0.93
Logistic Regression	1	0.89	1.0	0.94
Logistic Regression	2	0.94	1.0	0.97
Logistic Regression	3	0.97	0.94	0.96
Logistic Regression	4	1.0	0.96	0.98
Random Forest	0	0.97	1.0	0.98
Random Forest	1	1.0	0.97	0.99
Random Forest	2	1.0	1.0	1.0
Random Forest	3	1.0	1.0	1.0
Random Forest	4	1.0	1.0	1.0
Naive Bayes	0	0.97	0.94	0.95
Naive Bayes	1	0.94	0.97	0.96
Naive Bayes	2	0.94	1.0	0.97
Naive Bayes	3	0.97	0.94	0.96
Naive Bayes	4	1.0	0.96	0.98
CatBoost	0	1.0	0.94	0.97
CatBoost	1	0.94	1.0	0.97
CatBoost	2	1.0	1.0	1.0
CatBoost	3	1.0	1.0	1.0
CatBoost	4	1.0	1.0	1.0

Figure A.6: Classification Report Summary

A.14 Confusion matrix

```
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
axes = axes.flatten()

for idx, (name, model) in enumerate(models.items()):
    Y_pred = model.predict(X_test)
    cm = confusion_matrix(Y_test, Y_pred)

    sns.heatmap(cm, annot=True, fmt="d", cmap="flare", ax=axes[idx])
    axes[idx].set_title(f"Confusion Matrix - {name}")
    axes[idx].set_xlabel("Predicted Label")
    axes[idx].set_ylabel("True Label")

plt.subplots_adjust(wspace=0.3, hspace=0.4)
plt.show()
```

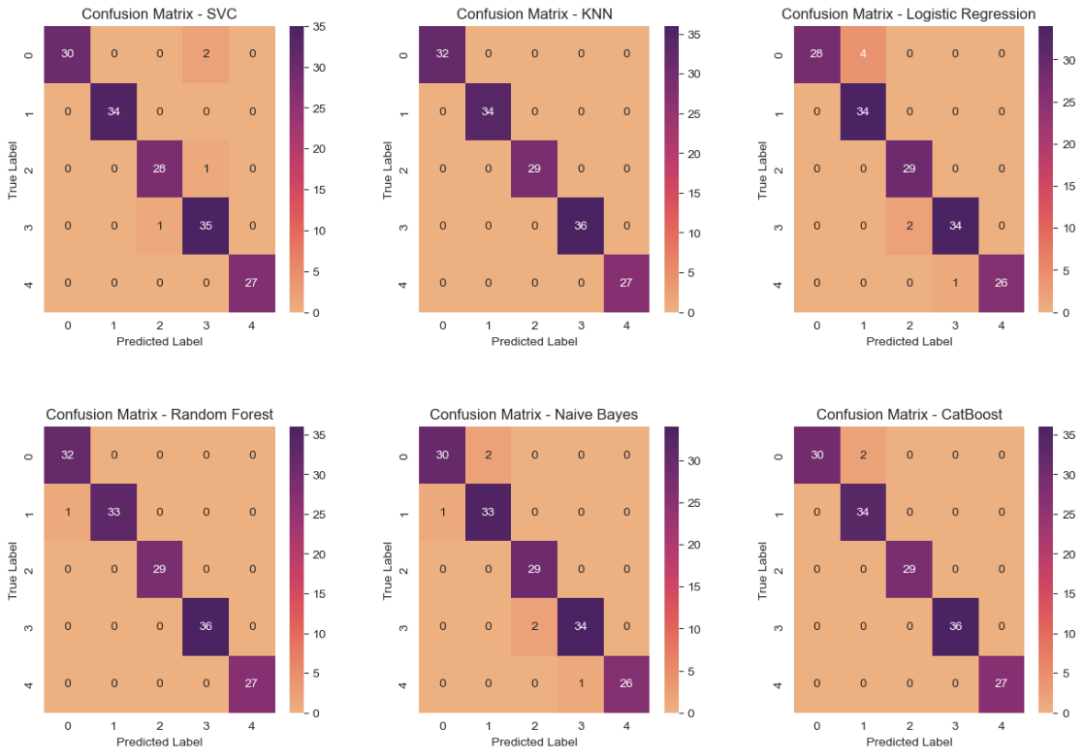
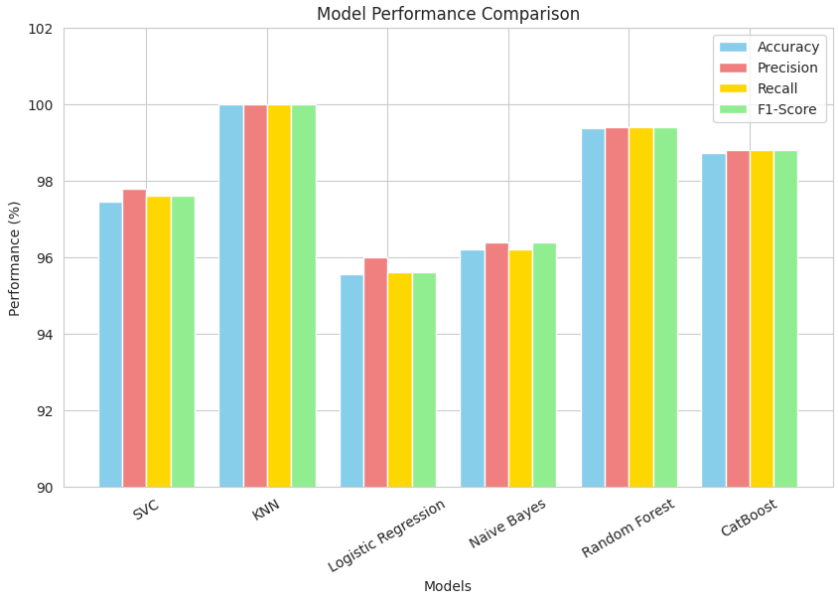


Figure A.7: Confusion Matrix

A.15 Model performance

```
# Define models and their respective performance metrics
models = ["SVC", "KNN", "Logistic Regression", "Naive Bayes", "Random Forest", "CatBoost"]
accuracy = [97.47, 100, 95.57, 96.2, 99.37, 98.73]
precision = [97.8, 100, 96, 96.4, 99.4, 98.8]
recall = [97.6, 100, 95.6, 96.2, 99.4, 98.8]
f1_score = [97.6, 100, 95.6, 96.4, 99.4, 98.8]

# Convert to NumPy array for easier manipulation
metrics = np.array([accuracy, precision, recall, f1_score])

# Define bar width and x-axis positions
bar_width = 0.2
x = np.arange(len(models))

# Plot grouped bars
plt.figure(figsize=(10, 6))
plt.bar(x - 1.5*bar_width, accuracy, width=bar_width, label="Accuracy", color="skyblue")
plt.bar(x - 0.5*bar_width, precision, width=bar_width, label="Precision", color="lightcoral")
plt.bar(x + 0.5*bar_width, recall, width=bar_width, label="Recall", color="gold")
plt.bar(x + 1.5*bar_width, f1_score, width=bar_width, label="F1-Score", color="lightgreen")

# Add labels and title
```

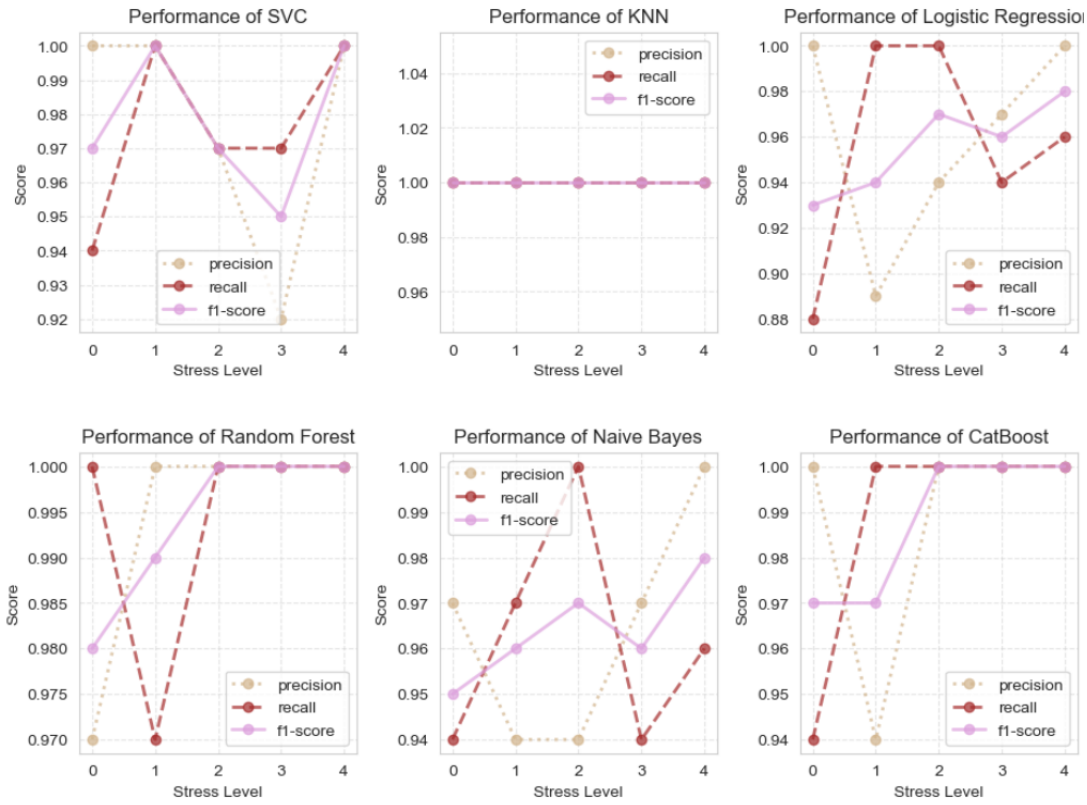
```
plt.xlabel("Models")
plt.ylabel("Performance (%)")
plt.title("Model Performance Comparison")
plt.xticks(ticks=x, labels=models, rotation=30)
plt.ylim(90, 102)
plt.legend()
plt.show()
```



A.15.1 Model performance at different stress levels

```
reports = {}

for name, model in models.items():
    Y_pred = model.predict(X_test)
    report = classification_report(Y_test, Y_pred, output_dict=True)

    # Extract only precision, recall, and f1-score
    class_metrics = {cls: report[str(cls)] for cls in range(5)}
    reports[name] = {f"{cls}_{metric}": round(class_metrics[cls][metric], 2)
                     for cls in class_metrics for metric in ["precision", "recall", "f1-score"]}

# Convert to DataFrame
df_report = pd.DataFrame(reports).T

# Define metrics, stress levels, and colors
metrics = ["precision", "recall", "f1-score"]
colors = {"precision": "tan", "recall": "brown", "f1-score": "plum"}
linestyles = {"precision": ":", "recall": "--", "f1-score": "-"}
stress_levels = [0, 1, 2, 3, 4]

# Determine subplot grid size
num_models = len(df_report.index)
cols = 3
rows = int(np.ceil(num_models / cols))
```

```python

# Create subplots
fig, axes = plt.subplots(rows, cols, figsize=(11, 8))

# Flatten axes if there's more than one row
axes = axes.flatten() if num_models > 1 else [axes]

# Iterate over models and plot on subplots
for idx, model in enumerate(df_report.index):
    ax = axes[idx]  # Select the subplot

    # Extract the model's data
    model_data = df_report.loc[model]

    for metric in metrics:
        values = [model_data[f"{cls}_{metric}"] for cls in
           stress_levels]
        ax.plot(
            stress_levels, values, marker="o", linestyle=linestyles[
               metric],
            color=colors[metric], alpha=0.7, linewidth=2, label=metric
        )

    ax.set_xticks(stress_levels)
    ax.set_xlabel("Stress␣Level")
    ax.set_ylabel("Score")
    ax.set_title(f"Performance␣of␣{model}")
    ax.legend()
    ax.grid(True, linestyle="--", alpha=0.5)

# Remove empty subplots if models < rows*cols
for i in range(num_models, rows * cols):
    fig.delaxes(axes[i])

# Adjust layout for better spacing
plt.subplots_adjust(wspace=0.3, hspace=0.4)
plt.show()
```


Performance of SVC
Performance of KNN
Performance of Logistic Regressio
Performance of Random Forest
Performance of Naive Bayes
Performance of CatBoost
precision
recall
f1-score
Score
Stress Level