DRIVER DETECTION SYSTEM

MINI PROJECT REPORT

Submitted by

ADWAITH NISHIN- ASI20CA007

AMRUTHA DAS- ASI20CA013

ER ARAVIND NAMPOOTHIRI – ASI20CA029

Under the Guidance of

Prof. Prabhu

to

the APJ Abdul Kalam Technological University

in partial fulfillment of the requirements for the award of the Degree

of

Bachelor of Technology

In

Bachelor of Technology in
Computer Science and Engineering
(Artificial Intelligence)

Department of Computer Science & Artificial Intelligence

ADI SHANKARA INSTITUTE OF ENGINEERING AND TECHNOLOGY

JUNE 2023

DEPARTMENT OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE

CERTIFICATE

Certified that this is a bonafide record of the mini project entitled

DRIVER DETECTION SYSTEM

Submitted by

ADWAITH NISHIN

ASI20CA007

AMRUTHA DAS
ASI20CA013

ER ARAVIND

ASI20CA030
Under the Guidance of

Prof. Prabhu

during the year 2022-23 in partial fulfilment of the requirement for the

award of the degree of

Bachelor of Technology in Computer Science and Engineering
(Artificial Intelligence)

Internal Guide External Supervisor

Project Coordinator Head of the Department

CONTENTS

ACKNOWLEDGEMENT

I express my heartfelt gratitude to the almighty, the supreme guide for bestowing his blessings in my entire endeavor.

I wish to give a deep sense of acknowledgement to our Principal in charge **Dr. Sreepriya S.**, for her constant encouragement and valuable advice throughout the course.

I am deeply indebted to **Prof. P V Rajaraman**, Head of the department of Computer Science and Artificial Intelligence, for his sincere and dedicated cooperation and encouragement throughout the duration of my project.

I would also like to thank my project coordinators **Ms. Vydehi**, Assistant Professor, and **Ms. Shobha T**, Assistant Professor, Department of Computer Science and Engineering and my project guide **Mr. Prabhu** , Assistant Professor, department of Computer Science and Engineering for their valuable advice and whole hearted cooperation without which this project would not have seen the light of the day. I express my sincere gratitude to all the faculty members of the department of Computer Science and Engineering for their cooperation and support to our project.

I am highly obliged to my parents for their encouragement and I express my sincere thanks to each and every one who has directly or indirectly helped me during the period of this project.

ABSTRACT

The project focuses on developing a computer vision-based system to detect and analyse driver behaviour for safer transportation systems. By combining computer vision techniques and machine learning algorithms, the system identifies potentially dangerous driving actions in real-time. It utilizes video data from in-vehicle cameras to analyse behaviours like lane changing, overtaking, distraction, drowsiness, and aggressive driving. The system extracts visual features and feeds them into a deep learning model that combines convolutional and recurrent neural networks for accurate detection and tracking over time. A diverse dataset of real-world driving scenarios is collected and annotated to evaluate the system's performance, including accuracy, precision, recall, and F1-score. The results show high accuracy rates, enabling early detection of risky actions and timely warnings to the driver or autonomous control system. The system's real-time capabilities and computational efficiency make it suitable for integration into autonomous vehicles and driver-assistance systems, enhancing overall safety and driving experience. This project presents a robust computer vision-based system to detect driver behaviour, contributing to safer transportation and intelligent transportation system development.

In conclusion, the computer vision-based system for detecting driver behaviour improves transportation safety. It accurately identifies behaviours in real-time and integrates well with autonomous vehicles. This system enhances overall safety and contributes to the advancement of intelligent transportation systems.

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In recent years, the advancement of computer vision and artificial intelligence (AI) has opened up new possibilities for improving road safety. One crucial aspect of ensuring safe driving is monitoring driver behaviour and detecting potential risks in real-time. With this objective in mind, the project "Driver Detection Using Computer Vision" aims to leverage the power of AI to enhance road safety and prevent accidents caused by distracted, drowsy, or reckless driving.

By harnessing the capabilities of computer vision techniques, this project focuses on developing an intelligent system that can analyse and interpret various driver actions and behaviours. The system utilizes cameras mounted within vehicles to capture real-time visual data of the driver's face, eyes, and body movements. The collected data is then processed and analysed using sophisticated algorithms to identify critical behavioural patterns, such as fatigue, distraction, aggressive driving, and other hazardous actions.

Through the integration of AI-powered computer vision techniques, this project strives to provide an effective solution for driver behaviour monitoring that can be deployed in vehicles, fleet management systems, and traffic surveillance systems. Ultimately, the aim is to enable timely interventions, such as issuing warnings or alerts, to both drivers and relevant authorities, thereby minimizing the risks associated with unsafe driving behaviours and fostering a safer road environment for all road users.

CHAPTER 2

LITERATURE REVIEW

Introduction

Driver detection systems are critical for ensuring road safety by monitoring and assessing driver behavior. This survey focuses on existing driver detection systems and highlights their shortcomings.

Types of Driver Detection Systems

- Passive systems: These monitor driver behavior without active intervention.
- Active systems: These provide feedback or alerts to the driver based on their behavior.

Types of Driver Detection Systems

The landscape of driver detection systems is diverse and evolving, offering a range of technologies to enhance road safety and driver awareness. One notable system is the Driver Attention Monitoring System (DAMS), which employs cameras to monitor eye movement and head position to detect driver drowsiness and distractions. However, DAMS does have limitations, such as reduced effectiveness in low-light conditions and challenges in detecting distractions that do not involve noticeable head movements.

Another significant player in this field is the Lane Departure Warning System (LDWS), which utilizes cameras to monitor a vehicle's lane position and alerts the driver if they veer out of their lane. While LDWS can contribute to safer driving, it is susceptible to errors in adverse weather or when road markings are faded. Additionally, there is a potential for false positives or negatives, particularly in situations involving abrupt lane changes.

The concept of Advanced Driver Assistance Systems (ADAS) encompasses various features designed to augment driving safety, including adaptive cruise control and collision avoidance. While ADAS presents promising benefits, there's a concern of drivers becoming overly reliant on the technology, which can lead to reduced attention and vigilance behind the wheel. Furthermore, ADAS systems may have limitations in complex traffic scenarios, necessitating continuous advancements to improve their performance and adaptability.

Machine learning-based approaches have gained prominence in driver detection systems. Behavioral Analysis with Machine Learning, for instance, employs historical data to predict risky driving behaviors. However, this approach demands substantial training data for accurate predictions and might struggle with previously unseen behaviors. Another approach involves physiological monitoring with AI, which examines signals like EEG and ECG to identify driver fatigue. Despite its potential, this method requires expensive hardware for precise data collection and faces challenges in real-time analysis and calibration.

As with any technology, driver detection systems encounter challenges and limitations that impact their reliability and effectiveness. Sensor reliability is a crucial factor, as inaccurate or malfunctioning sensors can result in false alarms or missed detections. Factors like poor weather, low visibility, and extreme lighting conditions can further affect the performance of these systems, emphasizing the need for robust sensor technology. Additionally, the deployment of these systems raises privacy and ethical concerns, such as continuous monitoring infringing on driver privacy and apprehensions about extensive data collection without consent.

In summary, the realm of driver detection systems offers a spectrum of solutions aimed at enhancing road safety and driver awareness. While these systems present promising advantages, they also face drawbacks and challenges that necessitate ongoing research and innovation to refine their capabilities and address ethical considerations. The intricate balance between technological advancement and privacy preservation underscores the importance of responsible development and deployment of driver detection technology.

2.1 Existing System

One example of an existing system that combines drowsiness detection and driver behavior pattern analysis is the "Driver Monitoring System" developed by major automotive manufacturers, such as Mercedes-Benz and BMW. These systems are often included as part of their Advanced Driver Assistance Systems (ADAS) to enhance road safety. Let's take a closer look at this type of system, along with its disadvantages:

Driver Monitoring System with Drowsiness Detection and Behavior Pattern Analysis

The Driver Monitoring System utilizes cameras and sensors located inside the vehicle to monitor the driver's behavior and condition. It employs two main features:

1. Drowsiness Detection: The system continuously analyzes the driver's facial features, eye movement, and head posture to detect signs of drowsiness or distraction. If the system detects that the driver's eyes are closing, their head is nodding, or they are not paying attention to the road, it issues an alert to warn the driver. This alert could be visual, auditory, or tactile, such as a vibration in the driver's seat or steering wheel.

2. Behavior Pattern Analysis: The system also observes the driver's behavior patterns, such as abrupt lane changes, rapid acceleration, or frequent lane departures. It compares these patterns to established norms for safe driving. If the system detects erratic or unsafe behavior, it can provide warnings or even intervene by adjusting vehicle settings, such as reducing speed or tightening seatbelt tension.

Disadvantages of the Driver Monitoring System

1. False Positives and Negatives: Like many advanced driver assistance systems, the driver monitoring system might generate false alarms (false positives) or fail to detect a drowsy or distracted driver in certain situations (false negatives). For instance, if the lighting conditions are

poor, or the driver's face is obscured, the system might not accurately detect drowsiness.

2. Sensitivity to Individual Differences: People have different behavior patterns and responses to drowsiness. The system's algorithms might not work equally well for all individuals. Some drivers may naturally exhibit behaviors that the system interprets as signs of drowsiness or distraction, leading to unnecessary alerts.

3. Driver Adaptation: Over time, drivers could become accustomed to the alerts and learn to ignore them, reducing the system's effectiveness. This adaptation could occur if the alerts are frequent or if drivers perceive them as overly sensitive.

4. Privacy Concerns: Cameras and sensors constantly monitoring the driver's face might raise privacy concerns for some users. Drivers may be uncomfortable with the idea of their facial data being captured and analyzed by the vehicle's systems.

5. Limited Contextual Understanding: The system primarily relies on visual cues and behavioral patterns. It might not fully understand the driver's context, such as a momentarily distracted gaze towards the passenger seat while conversing.

6. Dependence on Adequate Hardware: The accuracy of the system heavily depends on the quality and reliability of the cameras and sensors. Any malfunctions or limitations in these components could affect the system's performance.

7. Complexity and Cost: Implementing a comprehensive driver monitoring system with drowsiness detection and behavior pattern analysis requires advanced technology and software integration. This complexity can lead to increased manufacturing costs and potential maintenance challenges.

In conclusion, while driver monitoring systems with drowsiness detection and behavior pattern analysis offer promising solutions to enhance road safety, they come with their own set of disadvantages and challenges. It's crucial to continuously refine and improve these systems to address the limitations and ensure effective and accurate performance.

2.2 Proposed System

The proposed system of driver detection system combines several unique features and advantages to enhance road safety by monitoring driver alertness and responding to signs of drowsiness and driver pattern.

Unique Features

1. Integration of Drowsiness Detection and Eye Blink Analysis: The system combines drowsiness detection with eye blink analysis, allowing it to comprehensively assess the driver's level of alertness. This approach considers both facial landmarks and blink patterns to make accurate judgments about the driver's condition.

2. Real-time Monitoring: The system operates in real-time, continuously analyzing the driver's facial features and eye movements. This ensures prompt alerts and intervention when signs of drowsiness are detected.

3. Multimodal Input: By using the camera to capture facial landmarks and eye movements, your system leverages multiple data sources for more accurate and reliable drowsiness detection. This multimodal approach enhances the system's ability to differentiate between various states of alertness.

4. Customizable Alerts: Depending on the level of drowsiness detected, the system provides customizable alerts. These alerts could range from notifications indicating mild drowsiness to more urgent warnings for severe fatigue.

5. Adaptive Thresholds: Your system uses adaptive thresholds for blink analysis, taking into account individual variations in blink patterns. This adaptive approach increases the system's adaptability to different users' characteristics.

A driver pattern detection system encompasses features like speed monitoring, lane departure detection, braking and acceleration behavior analysis, tailgating detection, phone usage detection, drowsiness monitoring, traffic light compliance assessment, time-of-day analysis, geolocation and route tracking, driver behavior scoring, reporting and analytics, emergency response integration, driver coaching, and integration with telematics and fleet management systems. This system aims to enhance road safety by monitoring and alerting drivers about their driving habits, encouraging safer behavior, preventing accidents, and providing valuable insights for individuals and fleet managers to improve overall driving performance and habits.

Advantages:

1. Early Warning: Identifies drowsiness signs in real-time, allowing timely intervention.
2. Comprehensive Monitoring: Combines drowsiness and blink analysis for accurate detection.
3. Customizable Alerts: Offers tailored alerts based on severity of drowsiness.
4. Adaptive Thresholds: Personalized detection for individual drivers' blink patterns.
5. Non-Intrusive: Relies on camera input without requiring additional equipment.
6. User-Friendly: Provides intuitive visual alerts for easy driver understanding.

Driver Pattern

1. Behavioral Insights: Monitors various driving patterns for safer driving habits.
2. Real-Time Feedback: Offers immediate alerts to encourage safer driving behavior.
3. Personalized Coaching: Provides customized feedback to individual drivers.
4. Accident Prevention: Detects aggressive driving actions, reducing collision risks.
5. Data-Driven Improvement: Offers analytics for informed driving performance enhancement.
6. Fleet Management: Supports fleet operators in monitoring and optimizing operations.

CHAPTER 3

DESIGN AND ANALYSIS

It is most important phase of the driver detection system project. It takes analysis of maximum time and budget in this project.

Analysis

The analysis model is a concise, precise abstraction of what the desired system mustdo, or how it will be done.

Functional and Non-Functional Requirements:

Functional requirements for a driver detection system

1. **Real-Time Drowsiness Detection:**

- The system should monitor the driver's facial features, eye movement, and head posture in real time to identify signs of drowsiness.

- It should analyze blink patterns and eye closure duration to determine the level of alertness.

- When drowsiness is detected, the system must issue timely alerts to the driver, encouraging immediate action to prevent potential accidents.

2. **Behavioral Pattern Analysis**

- The system should track various driving behaviors such as speeding, abrupt lane changes, aggressive acceleration, and tailgating.

- It should continuously analyze driving patterns to provide real-time feedback to the driver about their behavior and suggest safer alternatives.

- Alerts should be triggered for unsafe driving actions, promoting conscious driving habits.

3. **Customizable Alert System**

- The system should provide configurable alerts based on the severity of detected drowsiness or unsafe driving patterns.

- Alerts can range from subtle visual cues to more prominent auditory or haptic notifications, depending on the user's preferences.

- Customizable thresholds for both drowsiness and behavior patterns should be available to suit individual driver profiles

4. **Data Recording and Analysis**

- The system should maintain a log of driver behavior patterns, drowsiness instances, and alert history.

- Recorded data should be accessible for analysis and reporting, allowing drivers and fleet managers to review performance and trends over time.

- The system should provide insights into improvements made based on past behavior and alert responses.

These functional requirements ensure that the driver detection system effectively addresses both drowsiness detection and driver pattern analysis, contributing to enhanced road safety and driving performance.

And nonfunctional requirement includes the following

1. **Accuracy and Reliability**

- The system must demonstrate high accuracy in detecting drowsiness and analyzing driver behavior patterns to minimize false positives and negatives.

- It should reliably differentiate between genuine drowsiness and temporary distractions or eye closures.

2**. Real-Time Responsiveness**

- The system should provide real-time responses to detected drowsiness and unsafe driving patterns, ensuring timely alerts to the driver.

- It should process data quickly and deliver alerts without significant delays to support immediate corrective actions.

3. **User Interface Intuitiveness**

- The user interface should be intuitive, presenting alerts and feedback in a clear and easily understandable manner for drivers.

- Visual, auditory, or haptic alerts should be designed to prevent driver confusion or distraction while on the road.

4. **Adaptability and Customization**

- The system should adapt to various driving environments, lighting conditions, and individual driver behaviors, enhancing its effectiveness across diverse scenarios.

- Users should be able to customize alert thresholds, preferences, and feedback settings to align with their driving habits and preferences.

These non-functional requirements ensure that the driver detection system not only meets its functional goals but also delivers a reliable, user-friendly, and adaptable experience for drivers while enhancing road safety.

3.1 System Architecture Diagram

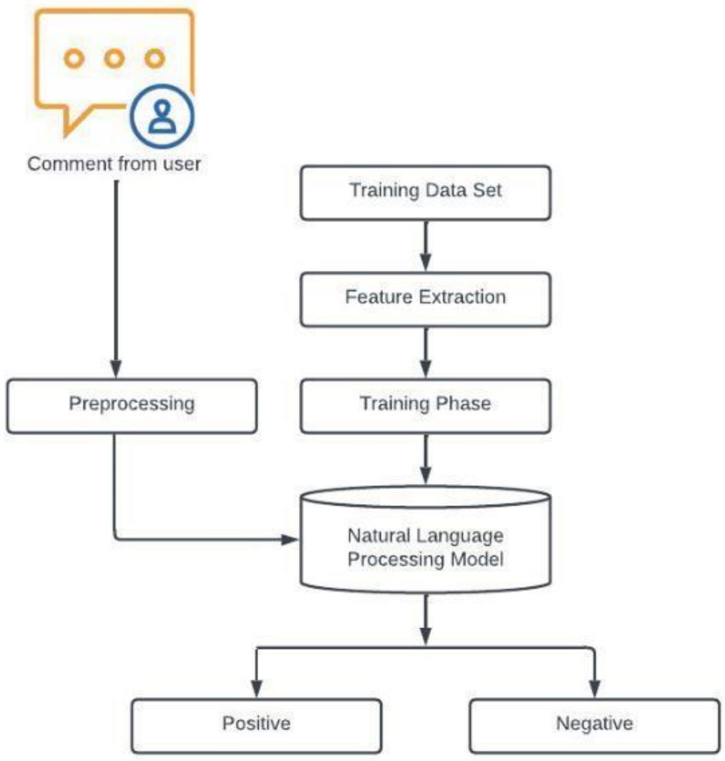


Fig 3.1: System Architecture Diagram

3.2 Use Case Diagram

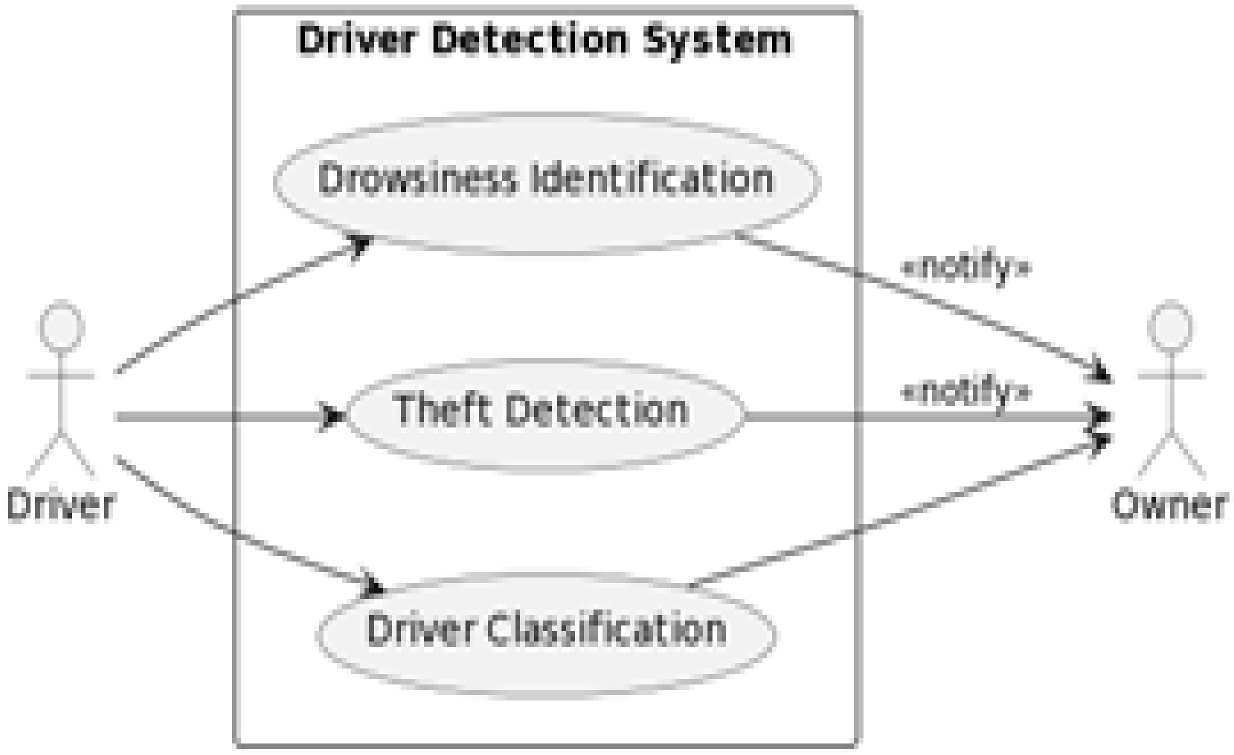


Fig 3.2 Use case diagram

3.3 Class Diagram

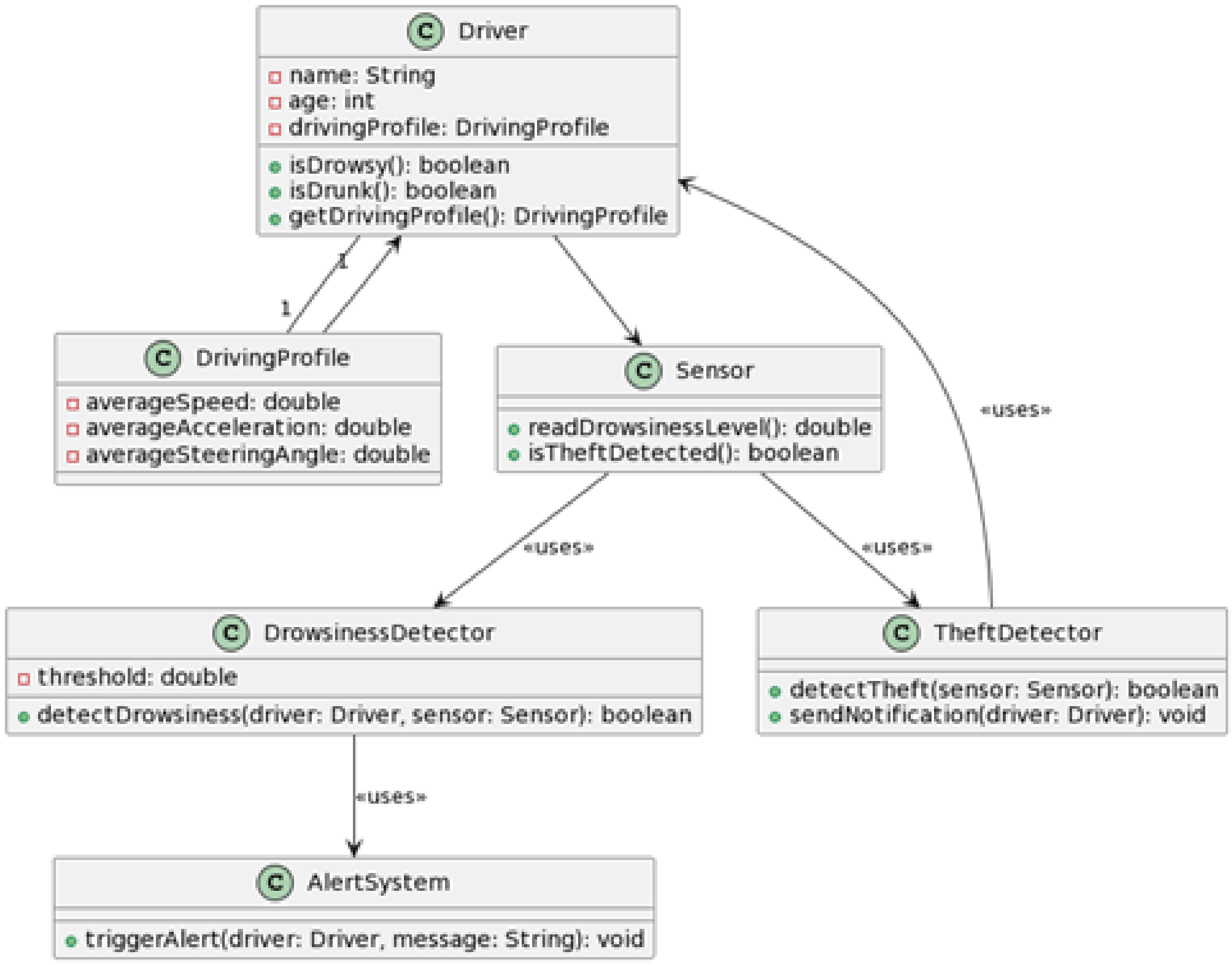

C Driver
name: String
age: int
drivingProfile: DrivingProfile
isDrowsy(): boolean
isDrunk(): boolean
getDrivingProfile(): DrivingProfile
1
1
C DrivingProfile
averageSpeed: double
averageAcceleration: double
averageSteeringAngle: double
C Sensor
readDrowsinessLevel(): double
isTheftDetected(): boolean
«uses»
«uses»
«uses»
C DrowsinessDetector
threshold: double
detectDrowsiness(driver: Driver, sensor: Sensor): boolean
C TheftDetector
detectTheft(sensor: Sensor): boolean
sendNotification(driver: Driver): void
«uses»
C AlertSystem
triggerAlert(driver: Driver, message: String): void

Fig 3.3 Class diagram

3.4 Sequence Diagram

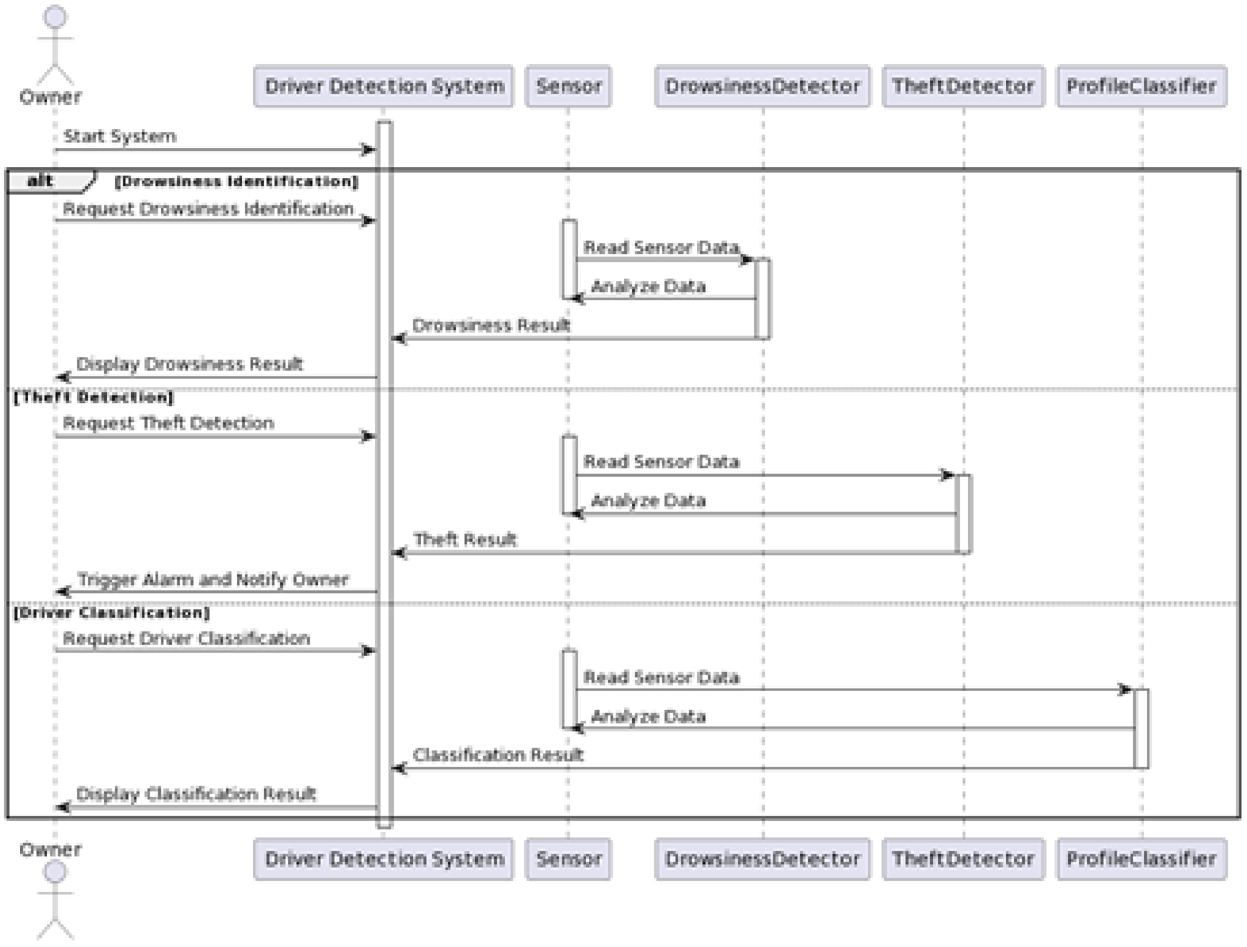


Fig 3.4 Sequence diagram

CHAPTER 4

REQUIREMENT PHASE

4.1 SYSTEM REQUIREMENTS

SOFTWARE AND HARDWARE REQUIREMENT

The software requirement specification (SRS) and hardware specification forms the basis of software development. A main purpose of software requirement specification is the clear definition and specification of functionality and of the software product. It allows the developer to be carried out, performance level to be obtained and corresponding interface to be established.

HARDWARE REQUIREMENTS

1. High-resolution cameras for facial and eye tracking.
2. In-vehicle sensors for speed, acceleration, and vehicle data.
3. Powerful multi-core processor for real-time processing.
4. Sufficient RAM and storage for data processing and logs.
5. Quality display for real-time alerts and interaction.
6. Connectivity (Wi-Fi, cellular) for data transfer and updates.
7. GPS module for accurate geolocation tracking.
8. Secure mounting hardware for cameras and sensors.
9. Integration interfaces with telematics and emergency systems.
10. Adequate power supply with voltage regulation.
11. Optional GPU for accelerated image processing.
12. Protective casings for hardware components.
13. Backup power source for continuous operation (optional).
14. Cooling solutions to prevent overheating (optional).

These requirements ensure a robust and capable hardware setup for effective drowsiness detection and driver pattern analysis.

SOFTWARE REQUIREMENTS

The software requirements for the proposed driver detection system are:

1. Operating System
   - Compatible OS (e.g., Windows, Linux) to run the application.

2. Programming Languages
   - Python, C++ for implementing algorithms.

3. Image Processing
   - OpenCV, Dlib for real-time image analysis.

4. Data Management
   - Database system for storing driver behavior data.

5. User Interface
   - GUI framework (e.g., Tkinter, Qt) for user-friendly alerts and customization.

6. Connectivity Tools
   - Libraries for Wi-Fi, cellular, GPS integration.

7. Cloud Integration (Optional)
   - Cloud platforms (e.g., AWS, Azure) for data backup and storage.

8. Development Tools
   - IDE (e.g., Visual Studio Code) for coding.

9. Testing Frameworks
   - Testing tools (e.g., pytest) for system verification.

10. Analytics and Reporting

- Data analytics tools (e.g., Pandas, Matplotlib) for generating reports.

11. Security Measures
    - Encryption tools for data protection.

12. Documentation Tools
    - Documentation software for user manuals and guides.

13. Deployment and Packaging
    - Tools for application deployment.

14. Backup and Recovery
    - Backup tools for data integrity.

These requirements form the software foundation for an effective driver detection system encompassing drowsiness detection and driver pattern analysis.

CHAPTER 5

TESTING

Testing is the process of detecting errors. Testing perform a critical role in quality assurance and for ensuring there liability of software .The results of testing are used later on during maintenance also.

Testing Objectives

The main objective of testing is to uncover a host of errors, systematically and with minimum effort and time. Stating formally, we can say,

- Testing is a process of executing a program with the intent of finding an error.
- A successful test is one that uncovers an as yet undiscovered error.
- A good test case is one that has a high probability of finding error, if it exists.
- The tests are in adequate to detect possibly present errors.
- The software more or less confirms to the quality and reliable standards.

There are different types of testing. Unit testing focuses verification effort on the smallest unit of software i.e. the module. Link testing does not test software but rather the integration of each module in a system. In integration testing, the goal is to see if modules can be integrated properly. This testing activity can be considered as testing the design and hence the emphasis on testing module interactions. The code testing strategy examines the logic of the program. Validation testing can be defined in many ways, but a simple definition is that validation succeeds when software functions in a manner that can be reasonable expected by the customer. Black Box Testing attempts to find errors in following areas or categories, incorrect or missing functions, interface error, errors in data structures, performance error and initialization and termination error. White box testing is a testing case design method that uses the control structure of the procedure design to derive test cases.

5.1 Module Testing

The starting point of testing is Unit testing. In this, a module is tested separately at each step. This helps to detect syntax and logical errors in the program and is performed by the coder himself /herself during coding.

5.2 Module Testing

The starting point of testing is Unit testing. In this, a module is tested separately at each step. This helps to detect syntax and logical errors in the program and is performed by the coder himself /herself during coding.

Module: Login ***Table 5.1 Login Case Test***

Test	**Input**	**Actual Output**	**Obtained Output**	**Description**
Valid	Email, Password			
Invalid Login	Email Id, Password	Login Failed	Login Failed	Test Passed.
Invalid	Null,	Login Failed	Login Failed	Test Passed.

Module: User Registration

Table 5.2 User Registration Case Test

Test Case	**Input**	**Actual Output**	**Obtained Output**	**Description**
Register User	Email Id Password	Success	Success	Test Passed. .
Invalid Email	Email id Password	Failed	Enter Valid Email	Test Passed.

Module: Driver Detection

Table 5.3: Driver Detection

Test Case	**Input**	**Actual Output**	**Obtained Output**	**Description**
Driver Pattern Feature 1	Different driving measures, accelerator value, engine value, torque etc..	Success	Success	Test Passed. .
Drowsiness Detection Feature 2	Camera capturing, positions etc..	Detect position Sleeping Active Drowsy	Success	Test Passed.

5.2 Integration Testing

The modules, which are tested in the Unit Testing, are integrated to build the overall system. It is observed that many errors crop up when the modules are joined together. Integration testing uncovers these errors while integrating the modules. It helps in establishing confidence (correctness) in the complete, assembled system. It tests the System Design. It focus on control, communication, interfaces, performance (other system qualities). It make use of stubs, test-beds, data generators. It is the phase of software testing in which indivisoftware modules are combined and tested as a group. It follows unit testing and precedes system testing.

Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

Integration testing concentrates entirely on module interactions, assuming that the details within each module are accurate. Module and Integration testing can be combined, verifying the details of each module's implementation in an integration context. Many projects compromise, combining module testing with the lowest level of subsystem integration testing, and then performing pure integration testing at higher levels. Each of these views of integration testing may be appropriate for any given project, so an integration testing method should be flexible enough to accommodate them all.

CHAPTER 6

ALGORITHM AND DATA

Algorithm for Driver Drowsiness

1. Initialization:
   - Initialize cameras, sensors, and required modules (OpenCV, Dlib).
   - Load pre-trained facial landmark detection model.
   - Initialize sleep, drowsy, active counters, and status.

2. Main Loop
   - Enter a continuous loop for real-time monitoring.
   - Capture a frame from the camera feed.
   - Convert the frame to grayscale.

3. Face and Landmark Detection
   - Detect faces using the frontal face detector.
   - For each detected face:
   - Extract facial landmarks using the shape predictor model.
   - Convert landmarks to NumPy array for processing.

4. Blink Detection and Drowsiness Assessment
   - Implement the "blinked" function to determine blink status.
   - Calculate left and right blink levels using landmark points.
   - Based on blink levels:
   - Update sleep, drowsy, and active counters.
   - Determine if the driver is sleeping, drowsy, or active.
   - Update status and color for visual feedback.

5. User Interface and Display
   - Display the frame with detected faces and landmarks.
   - Overlay status text on the frame indicating driver condition.
   - Display facial landmarks using circles on the frame.

6. Alert Generation and User Interaction
   - If drowsiness or sleep is detected for a threshold:
   - Generate an alert and notify the driver of drowsiness.
   - Allow the driver to acknowledge the alert and take corrective actions.

7. Data Logging and Reporting (Optional)
   - Store data related to driver condition, blink patterns, and timestamps.
   - Generate reports or logs for analysis and historical tracking.

8. Exit Condition
   - Listen for a key press event to exit the loop.
   - Release camera resources and close the application when desired.

9. Future Enhancements (Future Scope)
   - Discuss potential enhancements such as advanced machine learning, multi-sensor integration, and cloud-based analytics.

Certainly, here's an algorithm outline for the driver behavior pattern analysis process, incorporating initialization, data analysis (Exploratory Data Analysis or EDA), machine learning modeling, evaluation using a confusion matrix, and displaying relevant graphs:

<u>Algorithm for Driver Behavior Pattern Analysis</u>

1. Initialization
   - Import necessary libraries (e.g., pandas, numpy, scikit-learn, matplotlib).

- Load the dataset containing driver behavior data.
- Split the dataset into features (X) and target labels (y).

2. Exploratory Data Analysis (EDA)
   - Perform basic data exploration to understand the structure of the dataset.
   - Analyze data distribution, summary statistics, and missing values.
   - Visualize features using histograms, box plots, or scatter plots.

3. Feature Selection (Optional)
   - If needed, apply feature selection techniques (e.g., correlation analysis) to choose relevant features.

4. Data Preprocessing
   - Handle missing values (impute or remove).
   - Normalize or scale numerical features if required.

5. Data Splitting
   - Split the dataset into training and testing subsets (e.g., 80% training, 20% testing).

6. Machine Learning Modeling
   - Choose multiple machine learning algorithms (e.g., Decision Tree, Random Forest, Support Vector Machine).
   - For each algorithm:
     - Initialize the model.
     - Train the model using the training dataset.
     - Predict target labels for the testing dataset.

7. Evaluation using Confusion Matrix
   - For each trained model:
     - Generate a confusion matrix using predicted and actual labels.
     - Calculate accuracy, precision, recall, and F1-score from the confusion matrix.

8. Graphical Representation
   - Plot confusion matrices as heatmaps for each model.
   - Display bar charts comparing model performance metrics (accuracy, precision, recall, F1-score).
   - Create other relevant graphs or visualizations to showcase results.

9. Model Selection and Tuning
   - Analyze model performance metrics to select the best-performing algorithm.
   - Perform hyperparameter tuning if necessary to improve model accuracy.

10. Conclusion
    - Summarize the performance of different machine learning models.
    - Select the best model based on evaluation metrics.

11. Future Enhancements (Future Scope)
    - Discuss potential future improvements, such as incorporating more advanced algorithms, fine-tuning hyperparameters, and expanding the dataset.

This algorithm provides a structured approach to driver behavior pattern analysis, starting from data initialization and exploration, to machine learning modeling, evaluation, and visualization.

CHAPTER 7
CODE AND OUTPUT

Driver Pattern Code

```
#Importing Libraries
import math
import numpy as np
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_selection import mutual_info_classif
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import preprocessing
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import cross_val_score,KFold, StratifiedKFold
%matplotlib inline
```

1. **Data Acquisition**

2.1 Load the data

In [2]:

```
driver_data = pd.read_csv("driver_profile_data.csv")
```

2.2 Observations

In [3]:

```
#finding the Size and shape of the dataset
print("Size of the dataset is {}".format(driver_data.size))
print("The dataset contains {} rows and {} columns\n".format(driver_data.shape[0],
driver_data.shape[1]))
#finding the attributes of the dataset
print("The attributes and their data-types are as follows:")
driver_data.info()
```

3. Exploratory Data Analysis

3.1) Dataset Description

In [5]:

```
#Descriptive statistics
driver_data.describe().T
```

From the desciption of the dataset, we can find that some of the columns are having constant values with no variance.

In [6]:

```
#Finding columns with constant values and no variance
desc = driver_data.describe().T
const_cols = desc[desc['min']==desc['max']].index.to_list()
print("Columns with constant values: ", const_cols)
```

3.2) Finding missing values

In [7]:

```
# Finding the percentage of null values
driver_data.isnull().sum().sort_values(ascending = False)/driver_data.shape[0]*100
```

3.3) Finding the duplicate values

In [8]:

```
#Get duplicate rows
driver_data[driver_data.duplicated()]
```

Out[8]:

3.4) Visualization

In [9]:

```
#Bar plot for showing the target data
var = driver_data["Class"]
varValue = var.value_counts()
plt.bar(varValue.index,varValue)
plt.xticks(varValue.index,varValue.index.values)
plt.ylabel("Frequeny")
plt.title('Driver')
plt.show()
print("{}".format(varValue))
```

Histograms

In [10]:

```
fig, axes = plt.subplots(11,5, figsize=(20,40))
count=0
```

```
x,y=0,0
for col in driver_data.columns:
    axes[x][y].hist(driver_data[col], bins=50)
    axes[x][y].set_xlabel(col)
    count+=1
    x=math.floor(count/5)
    y=count%5
plt.show()
```

Boxplots

In [11]:

```
fig, axes = plt.subplots(14,4, figsize=(20,40))
count=0
a,b=0,0
for col in driver_data.drop('Class', axis=1).columns:
    sns.boxplot(data=driver_data, y=col, ax=axes[a, b])
    axes[a][b].set_xlabel(col)
    count+=1
    a=math.floor(count/4)
    b=count%4
plt.show()
```

From the boxplots of the features, we can see that there are outliers for most of the features and hence these needs to be removed.

4. Data Preparation

Perform the data preprocessing that is required for the data.

Separate the Features and Target

In [12]:

```
#separating features and target variable to perform feature selection
features = driver_data.drop(['Class'],axis=1)
target = driver_data[["Class"]]
```

4.1) Cleaning the dataset

4.1.1) Dealing insignificant features

The insignificant columns are removed

In [13]:

```
# Removing insignificant columns
features.drop(const_cols, axis=1, inplace=True)
```

4.1.2) Missing Values

The dataset has no missing values

4.1.3) Duplicate values

The dataset has no duplicates

4.1.4) Outlier Removal

In [14]:

```
features.describe(percentiles=[0.01,0.05,0.10,0.25,0.50,0.75,0.85,0.9,0.95])
```

Capping and Flooring of outliers

In [15]:

```
def outlier_cap(x):
    x=x.clip(lower=x.quantile(0.05))
    x=x.clip(upper=x.quantile(0.95))
    return(x)
```

In [16]:

```
features = features.apply(lambda x: outlier_cap(x))
```

In [17]:

```
features.describe(percentiles=[0.01,0.05,0.10,0.25,0.50,0.75,0.85,0.9,0.95])
```

4.2) Feature Encoding

There are no categorical features

Encoding the target variable

In [18]:

```
#Label encoding
le = preprocessing.LabelEncoder()
Y = le.fit_transform(target["Class"])
```

4.3 Feature Selection

We removed identical and extraneous features. For instance, Engine torque value is identical to correction of Engine torque value. After deleting redundant features, we performed feature selection to exclude highly correlated features for improving the performance in terms of the accuracy and speed. We selected 15 features from 51 features.

In [19]:

```
#Mutual information between features and target values are calculated
importances = mutual_info_classif(features, target['Class'].to_numpy())
```

In [20]:

```
#Plotting mutual information values
feat_importances = pd.Series(importances, features.columns)
feat_importances.sort_values(ascending=False, inplace=True)
feat_importances.plot(kind='barh',color='teal',figsize=(10,15))
plt.show()
#Selecting the best 15 fatures
X = features[feat_importances.index[:15]]
```

4.4 Split the main data into training set and testing set

In [22]:

```
#Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,random_state=0)
```

In [23]:

```
print("Shape of Training Data", X_train.shape)
print("Shape of Testing Data", X_test.shape)
print("Response Rate in Training Data", y_train.mean())
print("Response Rate in Testing Data", y_test.mean())
```

4.5 Data Normalization

In [24]:

```
from sklearn.preprocessing import MinMaxScaler
mmc = MinMaxScaler()
X_train = mmc.fit_transform(X_train)
X_test = mmc.transform(X_test)
```

5. Training the model

5.1 Train the model

5.1.1 Decision Tree

In [25]:

```
# Building a Decision Tree Model
dtree=DecisionTreeClassifier(criterion='gini',random_state=101)
```

Hyperparameter Tuning

In [26]:

```
param_dist = {'max_depth': [3, 5, 6, 7, 8, 9, 10, 11, 12],
            'min_samples_split': [5,10,20,30,50, 100, 150, 200, 250] }
tree_grid = GridSearchCV(dtree, cv = 10, param_grid=param_dist,n_jobs = 3)
tree_grid.fit(X_train,y_train)
print('Best Parameters using grid search: \n', tree_grid.best_params_)
```

Model Training with Best parameters

In [40]:

```
dtree=DecisionTreeClassifier(criterion='gini',random_state=0,max_depth=12,min_samples_split
=5)
dtree.fit(X_train,y_train)
```

5.1.2 Random Forest

In [41]:

```
# Building a Random Forest Model with best parameters from Decision Tree
rf=RandomForestClassifier(criterion='gini',random_state=101,max_depth=12,min_samples_split
=5)
rf.fit(X_train,y_train)
```

5.1.3 KNN

In [29]:

```
# Building a KNN Model
knn = KNeighborsClassifier()
```

Hyperparameter Tuning

In [30]:

```
grid_params = { 'n_neighbors' : [5,7,9,11,13,15],
               'weights' : ['uniform','distance'],
               'metric' : ['minkowski','euclidean','manhattan']}
gs = GridSearchCV(KNeighborsClassifier(),grid_params,verbose=1,cv=3,n_jobs=-1)
gs_res = gs.fit(X_train,y_train)
print('Best Parameters using grid search: \n', gs_res.best_params_)
Fitting 3 folds for each of 36 candidates, totalling 108 fits
Best Parameters using grid search:
 {'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'distance'}
```

Model Training with Best parameters

In [31]:

```
knn =
KNeighborsClassifier(n_neighbors=5,weights='distance',algorithm='brute',metric='manhattan')
knn.fit(X_train,y_train)
```

Evaluating models

In [32]:

```
# Creating the metrics
def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        clf_report = pd.DataFrame(classification_report(y_train, pred, output_dict=True))
        print("Train Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
    elif train==False:
        pred = clf.predict(X_test)
        clf_report = pd.DataFrame(classification_report(y_test, pred, output_dict=True))
        print("Test Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
```

Decision Tree

In [42]:

```
#Decision tree
print_score(dtree,X_train, y_train, X_test, y_test, train=True)
print_score(dtree,X_train, y_train, X_test, y_test, train=False)
```

Random Forest

In [43]:

```
#random forests
print_score(rf,X_train, y_train, X_test, y_test, train=True)
print_score(rf,X_train, y_train, X_test, y_test, train=False)
```

KNN

In [35]:

```
#KNN
print_score(knn,X_train, y_train, X_test, y_test, train=True)
print_score(knn,X_train, y_train, X_test, y_test, train=False)
```

6. Cross Validation

random forest is the best model in terms of accuracy score as well as f1-score. Therefore, we will be testing our random forest model using the hold-out dataset

In [36]:

```
kf=KFold(n_splits=5, shuffle=True, random_state=42)
```

In [37]:

```
rf=RandomForestClassifier(criterion='gini',random_state=101,max_depth=12,min_samples_split
=20)
```

In [38]:

```
score=cross_val_score(rf, X, Y, cv=kf)
```

In [39]:

```
print("Cross Validation Scores are {}".format(score))
print("Average Cross Validation score :{}".format(score.mean()))
```

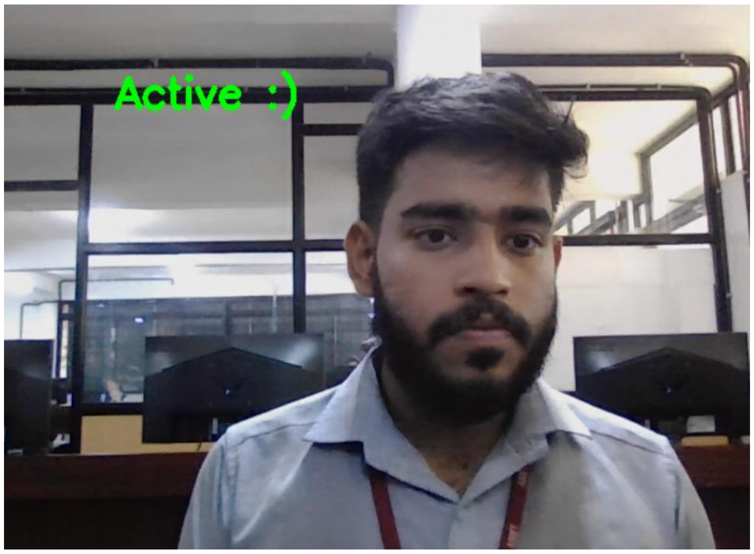
<u>Driver Drowsiness System</u>

```
#Importing OpenCV Library for basic image processing functions
import cv2
# Numpy for array related functions
import numpy as np
# Dlib for deep learning based Modules and face landmark detection
import dlib
#face_utils for basic operations of conversion
from imutils import face_utils


#Initializing the camera and taking the instance
cap = cv2.VideoCapture(0)

#Initializing the face detector and landmark detector
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

#status marking for current state
sleep = 0
drowsy = 0
active = 0
status=""
color=(0,0,0)

def compute(ptA,ptB):
  dist = np.linalg.norm(ptA - ptB)
  return dist

def blinked(a,b,c,d,e,f):
  up = compute(b,d) + compute(c,e)
  down = compute(a,f)
  ratio = up/(2.0*down)

  #Checking if it is blinked
  if(ratio>0.25):
    return 2
  elif(ratio>0.21 and ratio<=0.25):
    return 1
  else:
    return 0
```

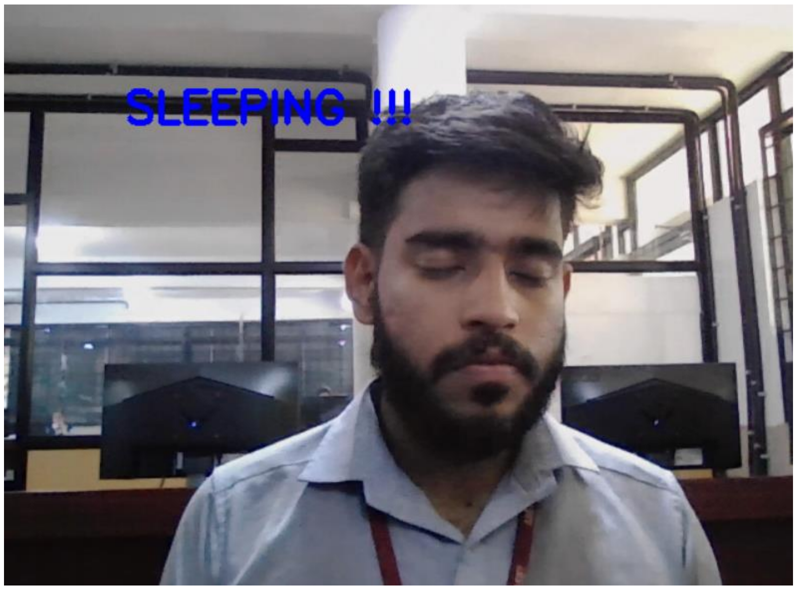
```python
while True:
    _, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces = detector(gray)
    #detected face in faces array
    for face in faces:
        x1 = face.left()
        y1 = face.top()
        x2 = face.right()
        y2 = face.bottom()

        face_frame = frame.copy()
        cv2.rectangle(face_frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

        landmarks = predictor(gray, face)
        landmarks = face_utils.shape_to_np(landmarks)

        #The numbers are actually the landmarks which will show eye
        left_blink = blinked(landmarks[36],landmarks[37],
          landmarks[38], landmarks[41], landmarks[40], landmarks[39])
        right_blink = blinked(landmarks[42],landmarks[43],
          landmarks[44], landmarks[47], landmarks[46], landmarks[45])

        #Now judge what to do for the eye blinks
        if(left_blink==0 or right_blink==0):
         sleep+=1
         drowsy=0
         active=0
         if(sleep>6):
          status="SLEEPING !!!"
          color = (255,0,0)

        elif(left_blink==1 or right_blink==1):
         sleep=0
         active=0
         drowsy+=1
         if(drowsy>6):
          status="Drowsy !"
          color = (0,0,255)

        else:
         drowsy=0
         sleep=0
         active+=1
         if(active>6):
          status="Active :)"
          color = (0,255,0)

        cv2.putText(frame, status, (100,100), cv2.FONT_HERSHEY_SIMPLEX, 1.2,
color,3)

        for n in range(0, 68):
         (x,y) = landmarks[n]
         cv2.circle(face_frame, (x, y), 1, (255, 255, 255), -1)

    cv2.imshow("Frame", frame)
    # cv2.imshow("Result of detector", face_frame)
    key = cv2.waitKey(1)
    if key == 27:
        break
    elif key == ord('q'):  # Pressing the 'q' key
        break  # Exit the loop if 'q' key is pressed

cap.release()  # Release the camera
cv2.destroyAllWindows()  # Close all OpenCV windows
```

OUTPUT

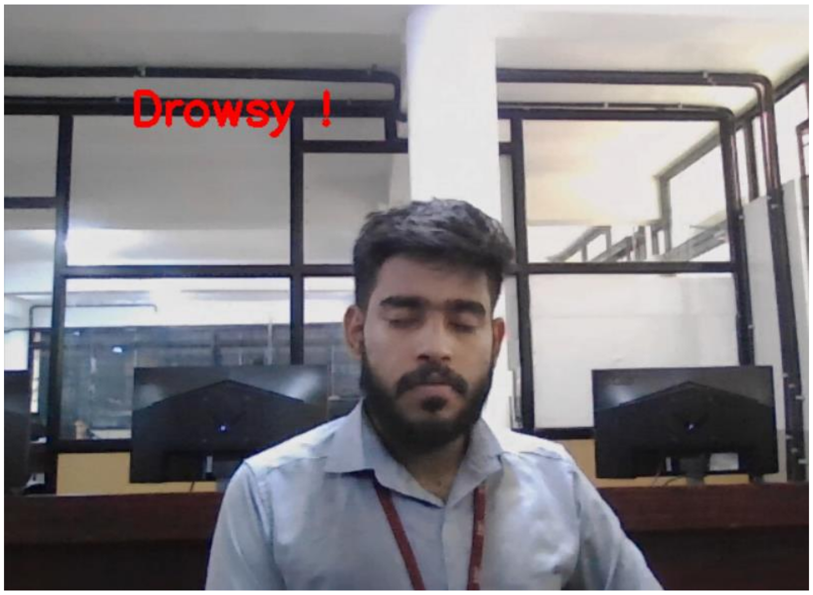
ACTIVE



SLEEPING

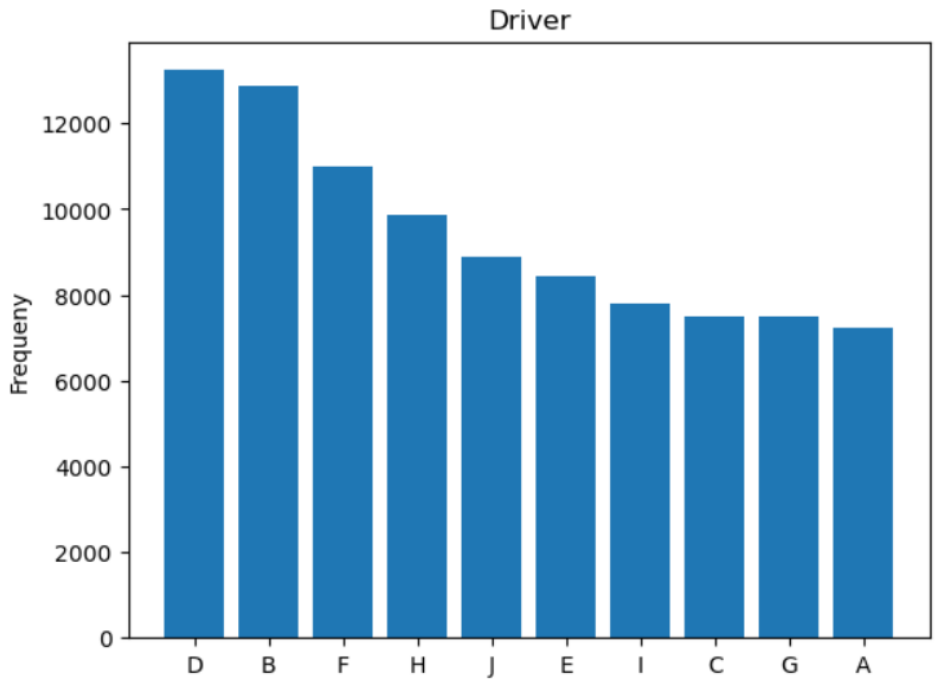


DROWSY



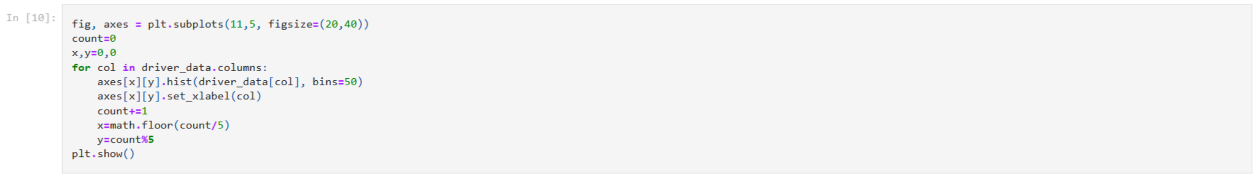
DRIVER PROFILE

```
plt.bar(varValue.index,varValue)
plt.xticks(varValue.index,varValue.index.values)
plt.ylabel("Frequeny")
plt.title('Driver')
plt.show()
print("{}".format(varValue))
```

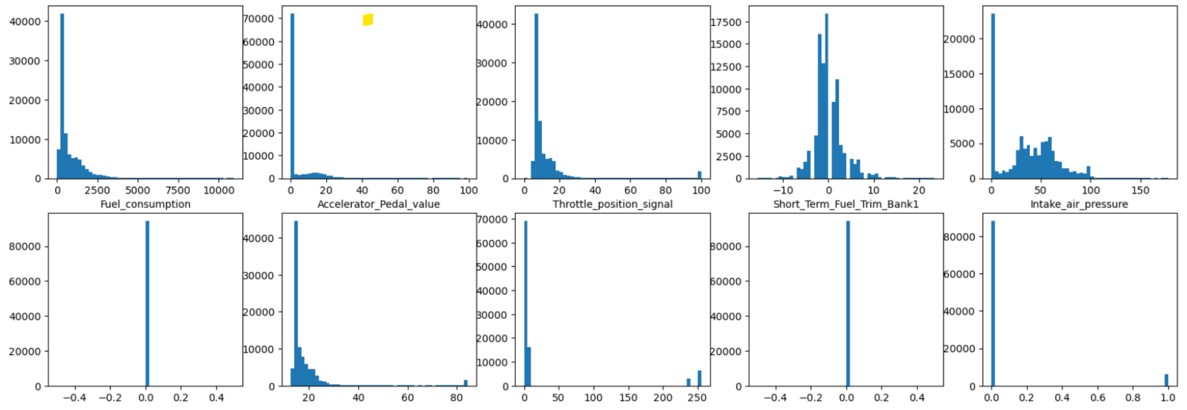


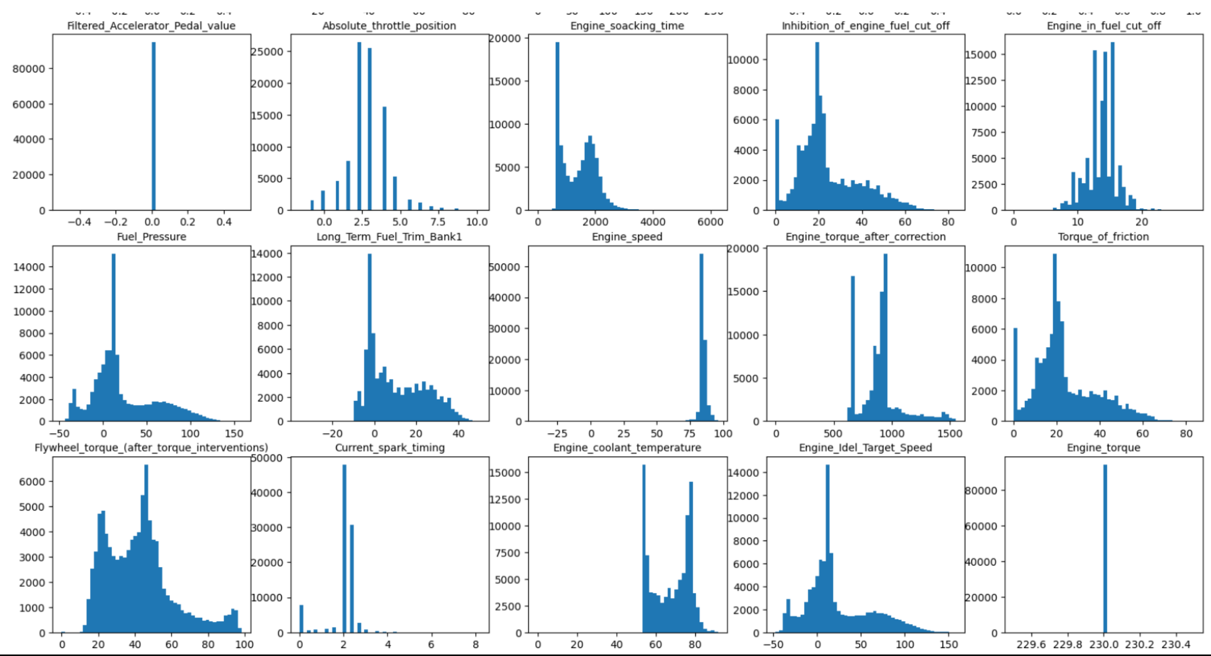
```
D    13244
B    12864
F    11012
H     9880
J     8904
E     8436
I     7808
C     7500
G     7492
A     7240
Name: Class, dtype: int64
```

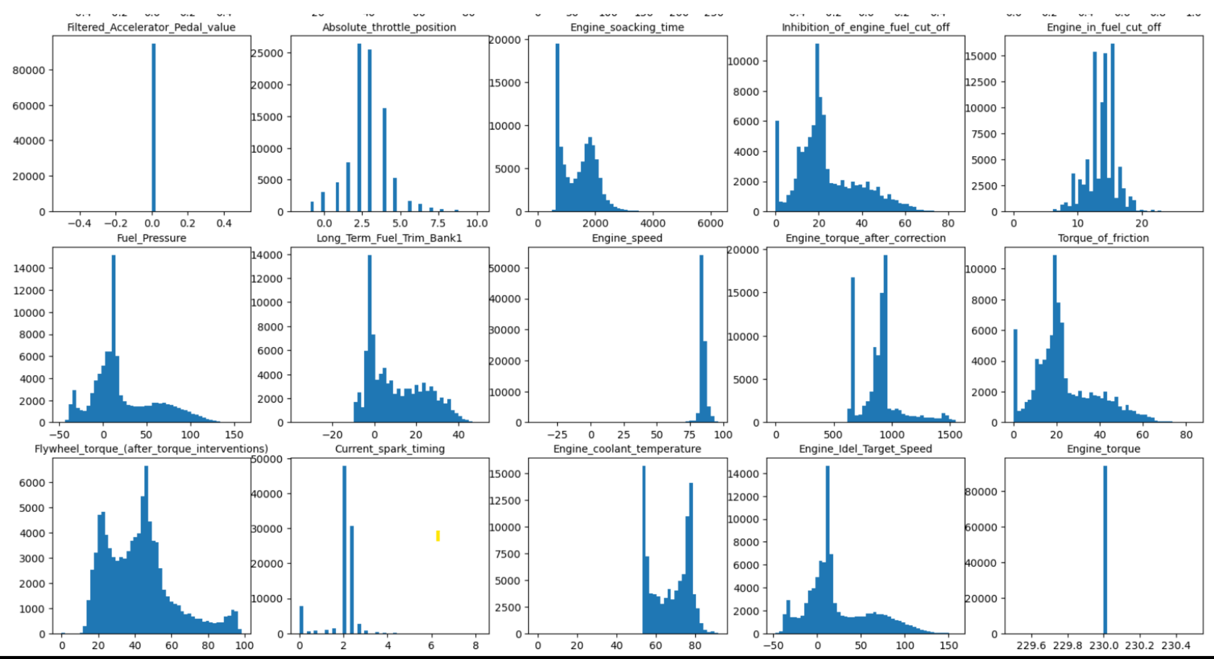
Histograms



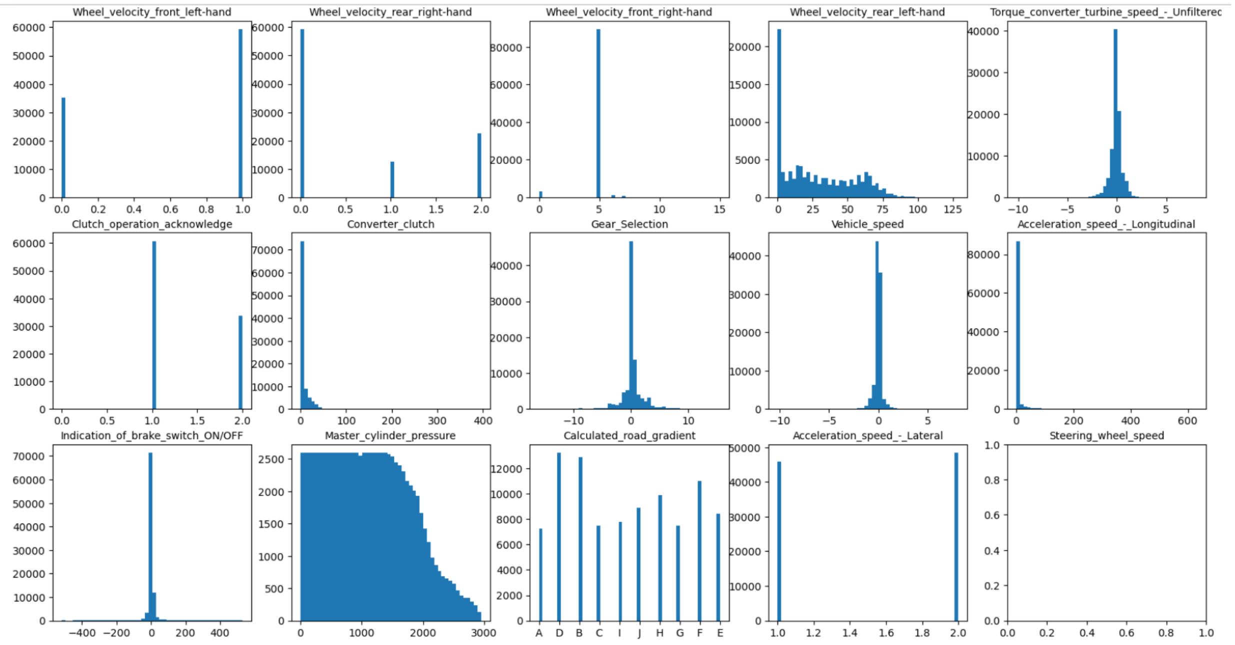
```
In [10]:
fig, axes = plt.subplots(11,5, figsize=(20,40))
count=0
x,y=0,0
for col in driver_data.columns:
    axes[x][y].hist(driver_data[col], bins=50)
    axes[x][y].set_xlabel(col)
    count+=1
    x=math.floor(count/5)
    y=count%5
plt.show()
```





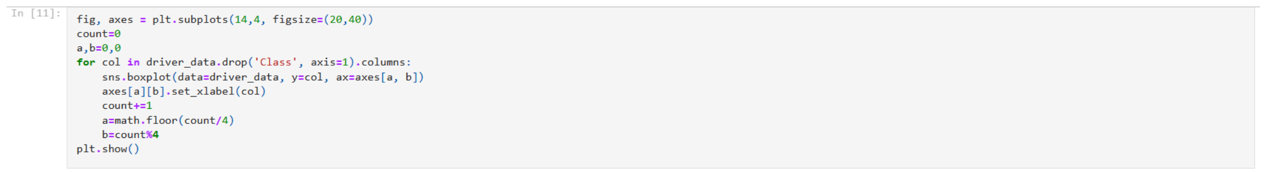


Filtered_Accelerator_Pedal_value
Absolute_throttle_position
Engine_soacking_time
Inhibition_of_engine_fuel_cut_off
Engine_in_fuel_cut_off
Fuel_Pressure
Long_Term_Fuel_Trim_Bank1
Engine_speed
Engine_torque_after_correction
Torque_of_friction
Flywheel_torque_(after_torque_interventions)
Current_spark_timing
Engine_coolant_temperature
Engine_Idel_Target_Speed
Engine_torque

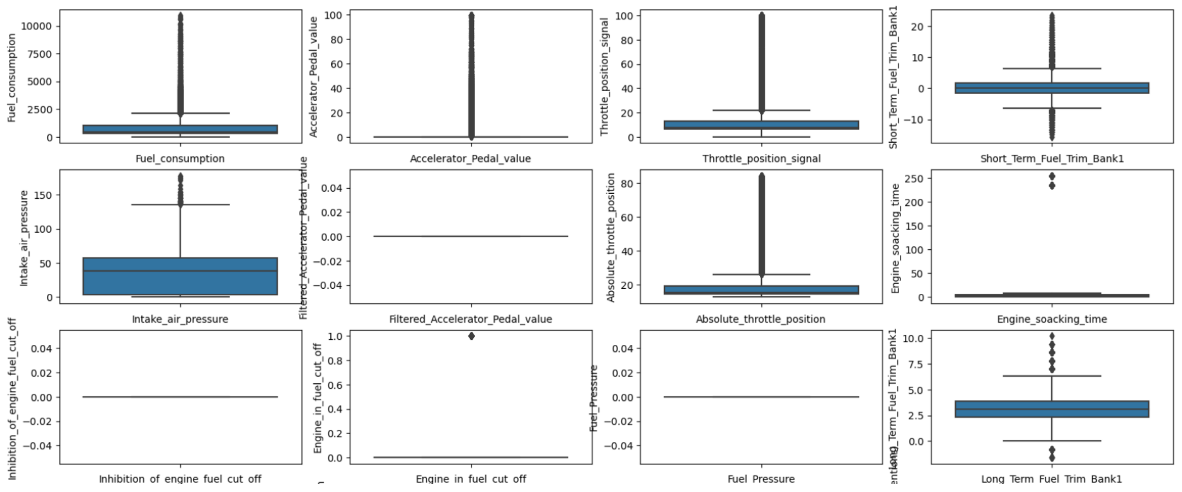


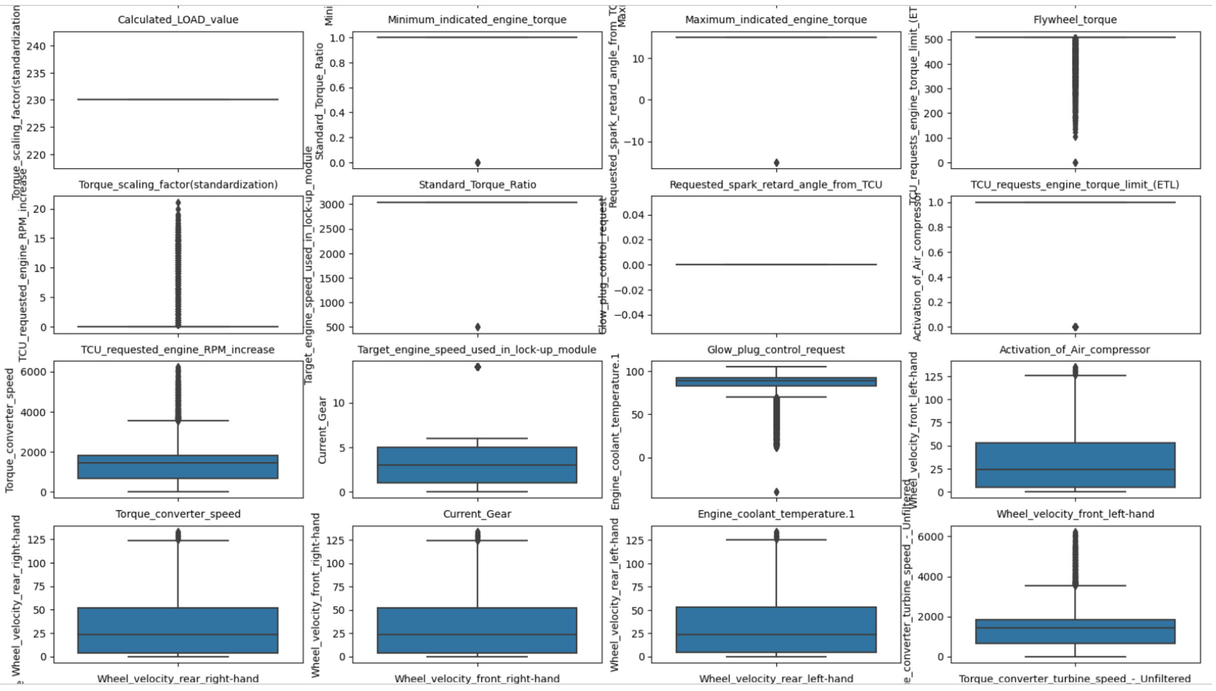
Wheel_velocity_front_left-hand
Wheel_velocity_rear_right-hand
Wheel_velocity_front_right-hand
Wheel_velocity_rear_left-hand
Torque_converter_turbine_speed_-_Unfilterec
Clutch_operation_acknowledge
Converter_clutch
Gear_Selection
Vehicle_speed
Acceleration_speed_-_Longitudinal
Indication_of_brake_switch_ON/OFF
Master_cylinder_pressure
Calculated_road_gradient
Acceleration_speed_-_Lateral
Steering_wheel_speed

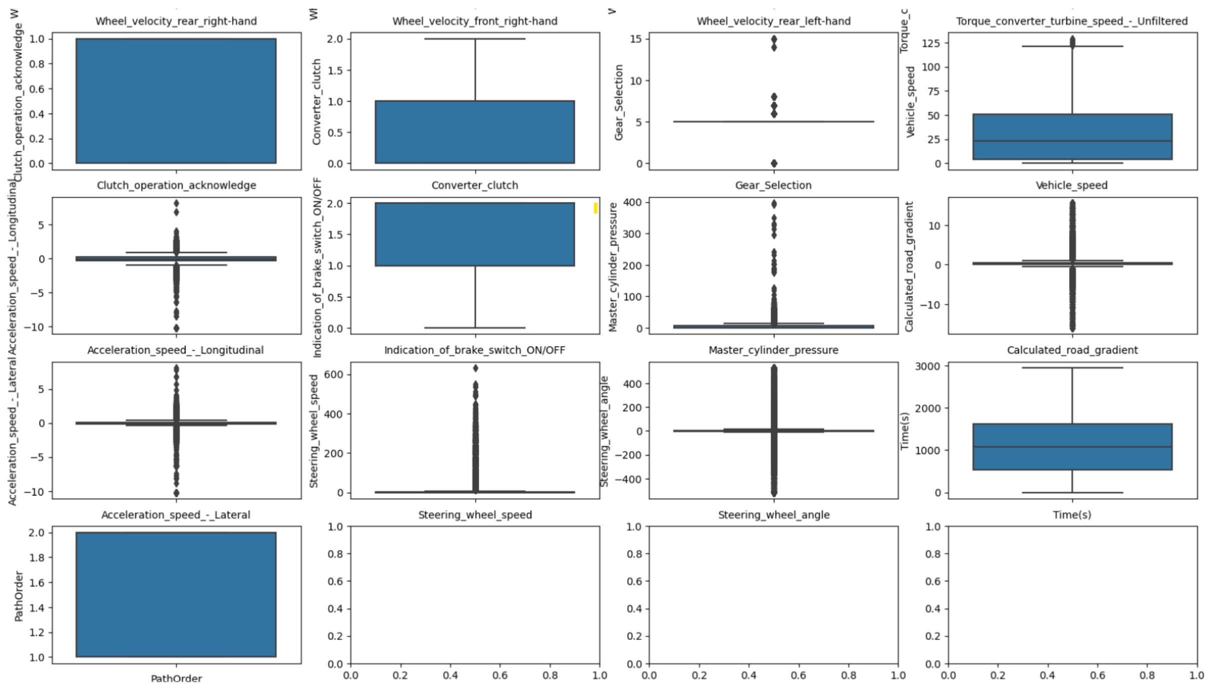
Box plot

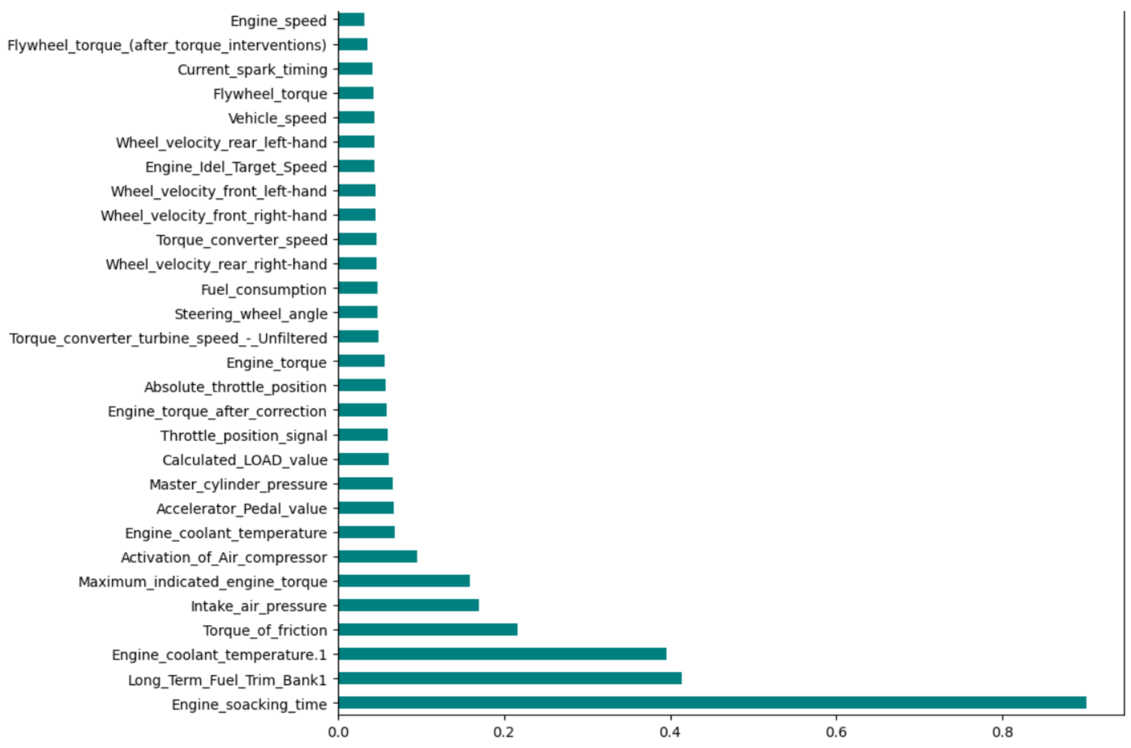


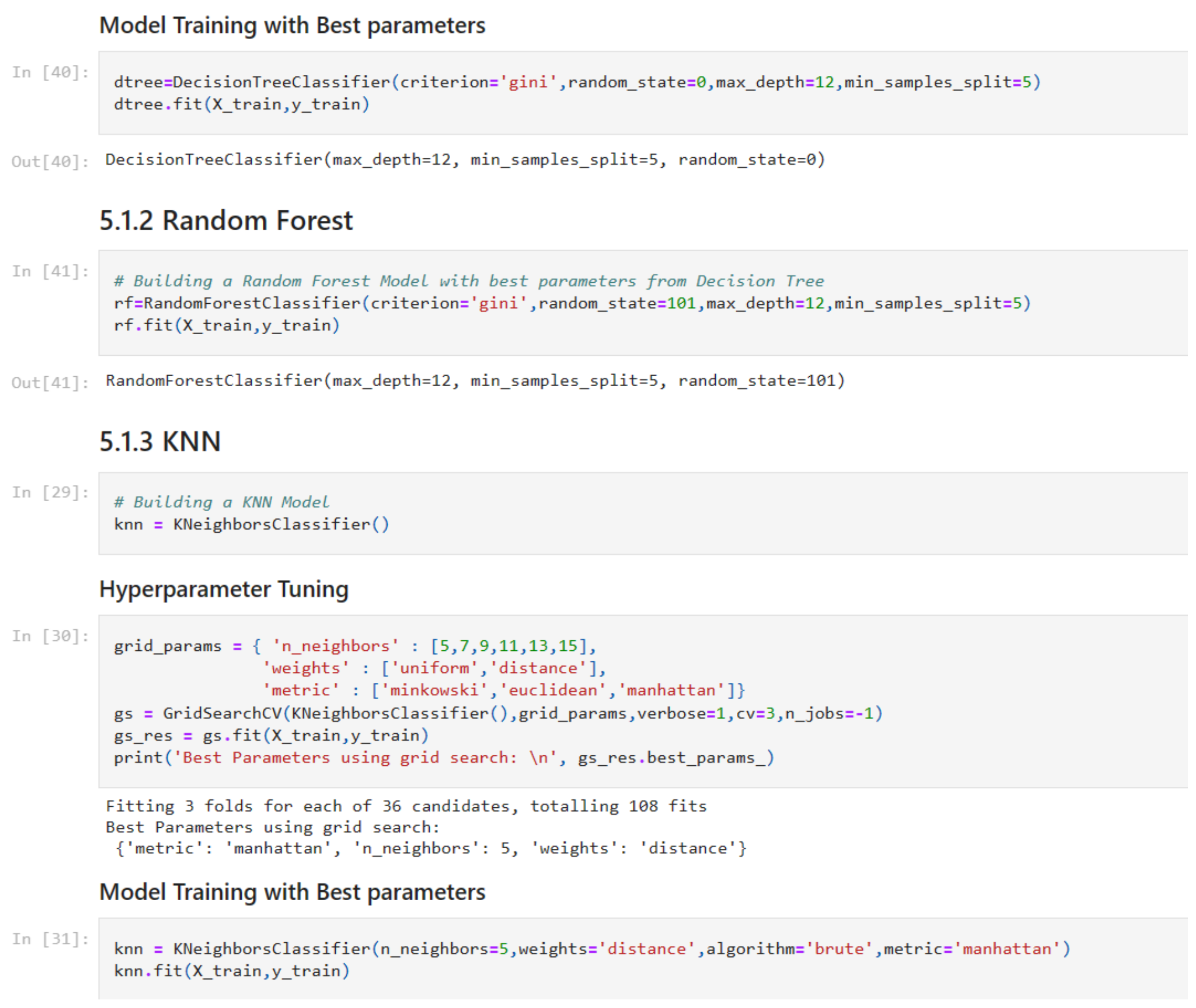
```
In [11]:  fig, axes = plt.subplots(14,4, figsize=(20,40))
          count=0
          a,b=0,0
          for col in driver_data.drop('Class', axis=1).columns:
              sns.boxplot(data=driver_data, y=col, ax=axes[a, b])
              axes[a][b].set_xlabel(col)
              count+=1
              a=math.floor(count/4)
              b=count%4
          plt.show()
```






Wheel_velocity_rear_right-hand
Wheel_velocity_front_right-hand
Wheel_velocity_rear_left-hand
Torque_converter_turbine_speed_-_Unfiltered
Clutch_operation_acknowledge
Converter_clutch
Gear_Selection
Vehicle_speed
Acceleration_speed_-_Longitudinal
Indication_of_brake_switch_ON/OFF
Master_cylinder_pressure
Calculated_road_gradient
Acceleration_speed_-_Lateral
Steering_wheel_speed
Steering_wheel_angle
Time(s)
PathOrder


Engine_speed
Flywheel_torque_(after_torque_interventions)
Current_spark_timing
Flywheel_torque
Vehicle_speed
Wheel_velocity_rear_left-hand
Engine_Idel_Target_Speed
Wheel_velocity_front_left-hand
Wheel_velocity_front_right-hand
Torque_converter_speed
Wheel_velocity_rear_right-hand
Fuel_consumption
Steering_wheel_angle
Torque_converter_turbine_speed_-_Unfiltered
Engine_torque
Absolute_throttle_position
Engine_torque_after_correction
Throttle_position_signal
Calculated_LOAD_value
Master_cylinder_pressure
Accelerator_Pedal_value
Engine_coolant_temperature
Activation_of_Air_compressor
Maximum_indicated_engine_torque
Intake_air_pressure
Torque_of_friction
Engine_coolant_temperature.1
Long_Term_Fuel_Trim_Bank1
Engine_soacking_time
0.0
0.2
0.4
0.6
0.8



Model Training with Best parameters

In [40]:
```
dtree=DecisionTreeClassifier(criterion='gini',random_state=0,max_depth=12,min_samples_split=5)
dtree.fit(X_train,y_train)
```

Out[40]:
```
DecisionTreeClassifier(max_depth=12, min_samples_split=5, random_state=0)
```

5.1.2 Random Forest

In [41]:
```
# Building a Random Forest Model with best parameters from Decision Tree
rf=RandomForestClassifier(criterion='gini',random_state=101,max_depth=12,min_samples_split=5)
rf.fit(X_train,y_train)
```

Out[41]:
```
RandomForestClassifier(max_depth=12, min_samples_split=5, random_state=101)
```

5.1.3 KNN

In [29]:
```
# Building a KNN Model
knn = KNeighborsClassifier()
```

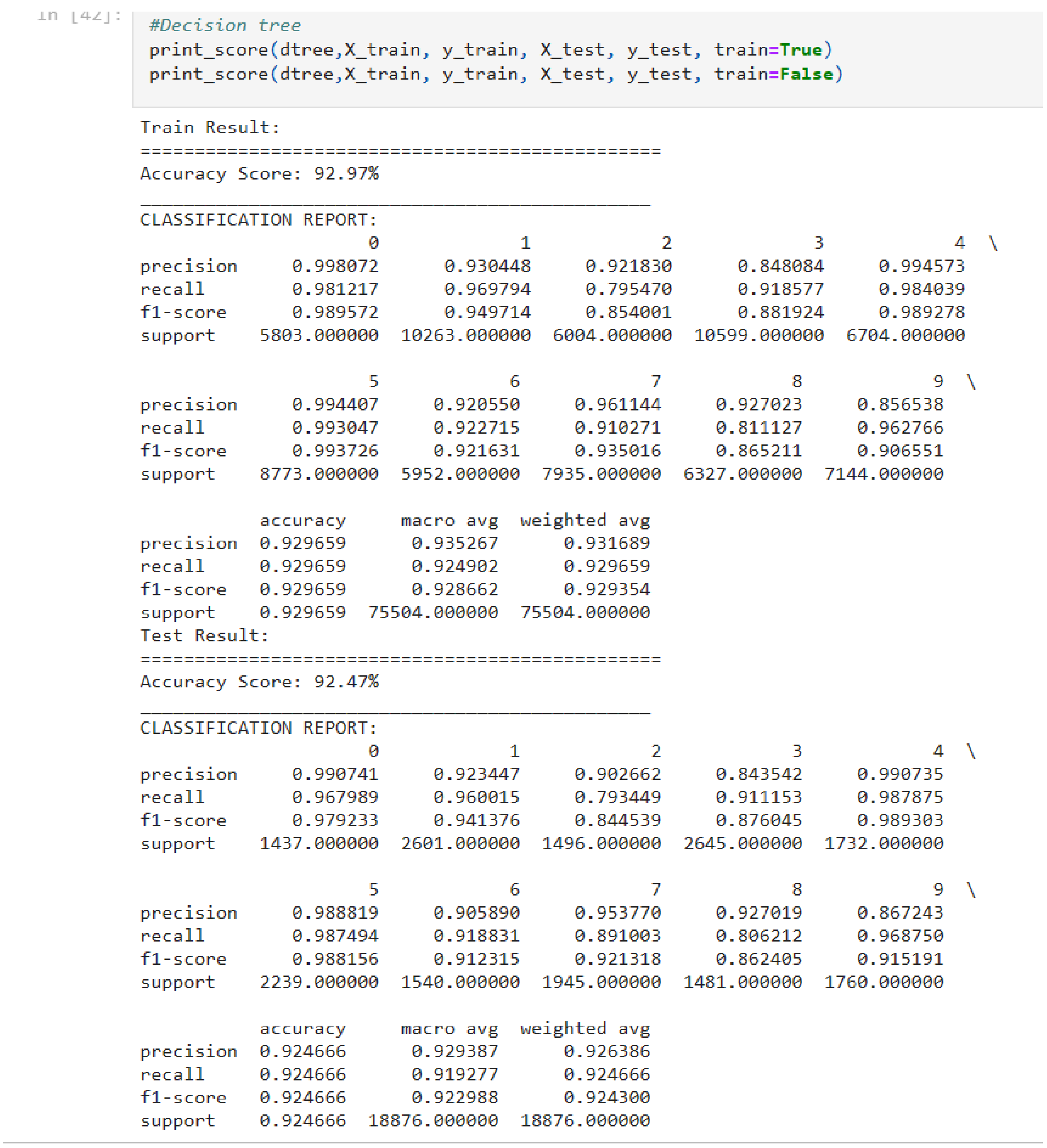
Hyperparameter Tuning

In [30]:
```
grid_params = { 'n_neighbors' : [5,7,9,11,13,15],
               'weights' : ['uniform','distance'],
               'metric' : ['minkowski','euclidean','manhattan']}
gs = GridSearchCV(KNeighborsClassifier(),grid_params,verbose=1,cv=3,n_jobs=-1)
gs_res = gs.fit(X_train,y_train)
print('Best Parameters using grid search: \n', gs_res.best_params_)
```

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits
Best Parameters using grid search:
 {'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'distance'}
```

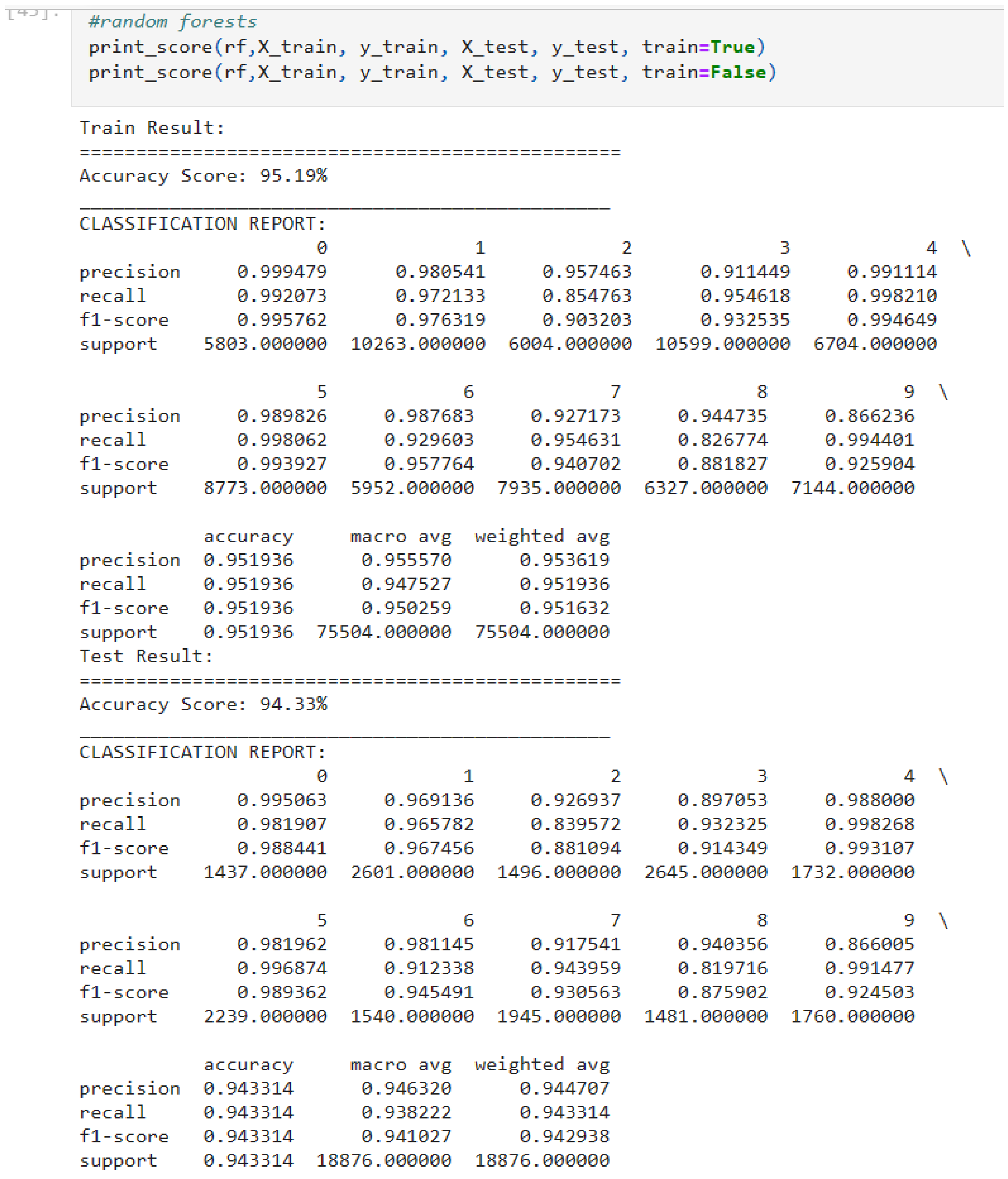
Model Training with Best parameters

In [31]:
```
knn = KNeighborsClassifier(n_neighbors=5,weights='distance',algorithm='brute',metric='manhattan')
knn.fit(X_train,y_train)
```



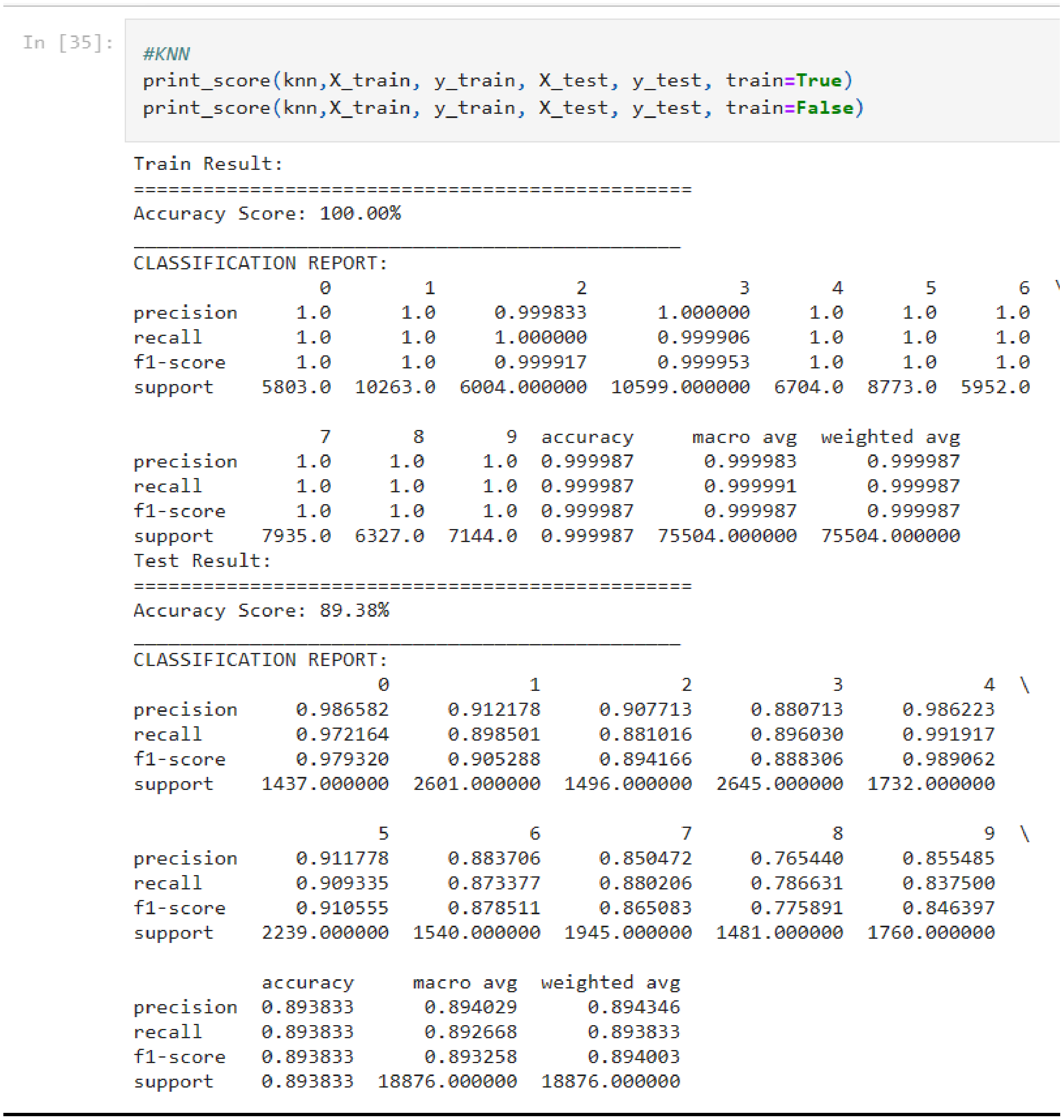
```
In [42]:  #Decision tree
          print_score(dtree,X_train, y_train, X_test, y_test, train=True)
          print_score(dtree,X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 92.97%
_______________________________________________
CLASSIFICATION REPORT:
                     0             1            2             3            4  \
precision     0.998072      0.930448     0.921830      0.848084     0.994573
recall        0.981217      0.969794     0.795470      0.918577     0.984039
f1-score      0.989572      0.949714     0.854001      0.881924     0.989278
support    5803.000000  10263.000000  6004.000000  10599.000000  6704.000000

                     5            6            7            8            9  \
precision     0.994407     0.920550     0.961144     0.927023     0.856538
recall        0.993047     0.922715     0.910271     0.811127     0.962766
f1-score      0.993726     0.921631     0.935016     0.865211     0.906551
support    8773.000000  5952.000000  7935.000000  6327.000000  7144.000000

           accuracy     macro avg  weighted avg
precision  0.929659      0.935267      0.931689
recall     0.929659      0.924902      0.929659
f1-score   0.929659      0.928662      0.929354
support    0.929659  75504.000000  75504.000000
Test Result:
================================================
Accuracy Score: 92.47%
_______________________________________________
CLASSIFICATION REPORT:
                     0            1            2            3            4  \
precision     0.990741     0.923447     0.902662     0.843542     0.990735
recall        0.967989     0.960015     0.793449     0.911153     0.987875
f1-score      0.979233     0.941376     0.844539     0.876045     0.989303
support    1437.000000  2601.000000  1496.000000  2645.000000  1732.000000

                     5            6            7            8            9  \
precision     0.988819     0.905890     0.953770     0.927019     0.867243
recall        0.987494     0.918831     0.891003     0.806212     0.968750
f1-score      0.988156     0.912315     0.921318     0.862405     0.915191
support    2239.000000  1540.000000  1945.000000  1481.000000  1760.000000

           accuracy     macro avg  weighted avg
precision  0.924666      0.929387      0.926386
recall     0.924666      0.919277      0.924666
f1-score   0.924666      0.922988      0.924300
support    0.924666  18876.000000  18876.000000
```



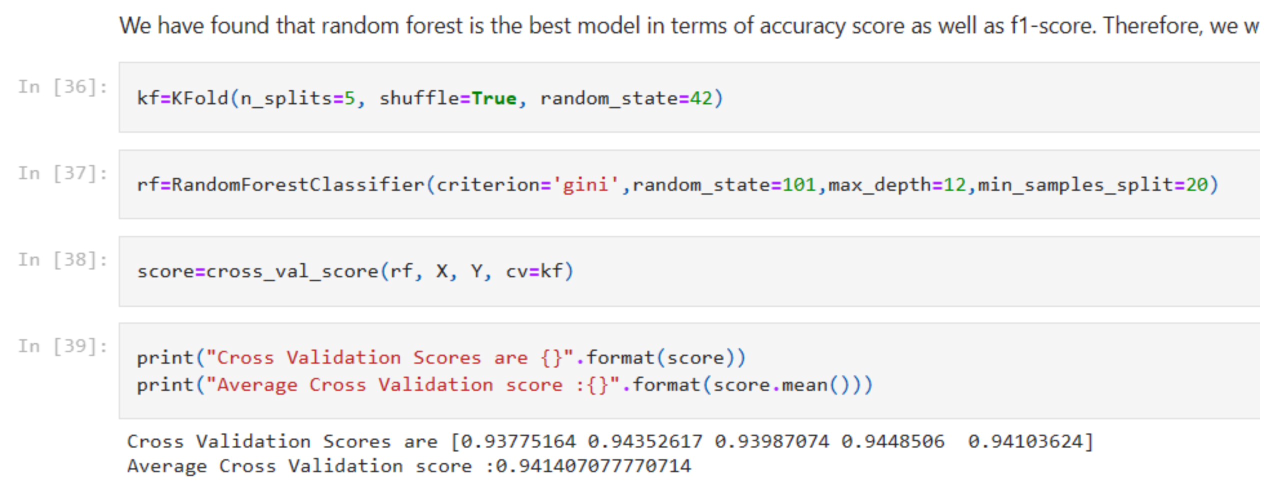
```
#random forests
print_score(rf,X_train, y_train, X_test, y_test, train=True)
print_score(rf,X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 95.19%
_______________________________________________
CLASSIFICATION REPORT:
                     0             1            2             3            4  \
precision     0.999479      0.980541     0.957463      0.911449     0.991114
recall        0.992073      0.972133     0.854763      0.954618     0.998210
f1-score      0.995762      0.976319     0.903203      0.932535     0.994649
support    5803.000000  10263.000000  6004.000000  10599.000000  6704.000000

                     5            6            7            8            9  \
precision     0.989826     0.987683     0.927173     0.944735     0.866236
recall        0.998062     0.929603     0.954631     0.826774     0.994401
f1-score      0.993927     0.957764     0.940702     0.881827     0.925904
support    8773.000000  5952.000000  7935.000000  6327.000000  7144.000000

           accuracy     macro avg  weighted avg
precision  0.951936      0.955570      0.953619
recall     0.951936      0.947527      0.951936
f1-score   0.951936      0.950259      0.951632
support    0.951936  75504.000000  75504.000000
Test Result:
================================================
Accuracy Score: 94.33%
_______________________________________________
CLASSIFICATION REPORT:
                     0            1            2            3            4  \
precision     0.995063     0.969136     0.926937     0.897053     0.988000
recall        0.981907     0.965782     0.839572     0.932325     0.998268
f1-score      0.988441     0.967456     0.881094     0.914349     0.993107
support    1437.000000  2601.000000  1496.000000  2645.000000  1732.000000

                     5            6            7            8            9  \
precision     0.981962     0.981145     0.917541     0.940356     0.866005
recall        0.996874     0.912338     0.943959     0.819716     0.991477
f1-score      0.989362     0.945491     0.930563     0.875902     0.924503
support    2239.000000  1540.000000  1945.000000  1481.000000  1760.000000

           accuracy     macro avg  weighted avg
precision  0.943314      0.946320      0.944707
recall     0.943314      0.938222      0.943314
f1-score   0.943314      0.941027      0.942938
support    0.943314  18876.000000  18876.000000
```



```
In [35]:  #KNN
           print_score(knn,X_train, y_train, X_test, y_test, train=True)
           print_score(knn,X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
CLASSIFICATION REPORT:
                0        1         2             3       4       5       6  \
precision     1.0      1.0  0.999833      1.000000     1.0     1.0     1.0
recall        1.0      1.0  1.000000      0.999906     1.0     1.0     1.0
f1-score      1.0      1.0  0.999917      0.999953     1.0     1.0     1.0
support    5803.0  10263.0  6004.000000  10599.000000  6704.0  8773.0  5952.0

                7       8       9  accuracy     macro avg  weighted avg
precision     1.0     1.0     1.0  0.999987      0.999983      0.999987
recall        1.0     1.0     1.0  0.999987      0.999991      0.999987
f1-score      1.0     1.0     1.0  0.999987      0.999987      0.999987
support    7935.0  6327.0  7144.0  0.999987  75504.000000  75504.000000
Test Result:
================================================
Accuracy Score: 89.38%
_______________________________________________
CLASSIFICATION REPORT:
                     0            1            2            3            4  \
precision     0.986582     0.912178     0.907713     0.880713     0.986223
recall        0.972164     0.898501     0.881016     0.896030     0.991917
f1-score      0.979320     0.905288     0.894166     0.888306     0.989062
support    1437.000000  2601.000000  1496.000000  2645.000000  1732.000000

                     5            6            7            8            9  \
precision     0.911778     0.883706     0.850472     0.765440     0.855485
recall        0.909335     0.873377     0.880206     0.786631     0.837500
f1-score      0.910555     0.878511     0.865083     0.775891     0.846397
support    2239.000000  1540.000000  1945.000000  1481.000000  1760.000000

           accuracy     macro avg  weighted avg
precision  0.893833      0.894029      0.894346
recall     0.893833      0.892668      0.893833
f1-score   0.893833      0.893258      0.894003
support    0.893833  18876.000000  18876.000000
```



We have found that random forest is the best model in terms of accuracy score as well as f1-score. Therefore, we w

In [36]:
```
kf=KFold(n_splits=5, shuffle=True, random_state=42)
```

In [37]:
```
rf=RandomForestClassifier(criterion='gini',random_state=101,max_depth=12,min_samples_split=20)
```

In [38]:
```
score=cross_val_score(rf, X, Y, cv=kf)
```

In [39]:
```
print("Cross Validation Scores are {}".format(score))
print("Average Cross Validation score :{}".format(score.mean()))
```

```
Cross Validation Scores are [0.93775164 0.94352617 0.93987074 0.9448506  0.94103624]
Average Cross Validation score :0.941407077770714
```

CHAPTER 8

CONCLUSION AND FUTURE SCOPE

In conclusion, the developed driver drowsiness and pattern detection system demonstrates promising capabilities in enhancing road safety through real-time monitoring of driver alertness and behavior patterns. The integration of image processing, machine learning algorithms, and user-friendly alerts provides a comprehensive solution for early drowsiness detection and encouraging safer driving habits. By accurately identifying drowsiness and analyzing behavior patterns, the system contributes to reducing accidents and improving overall driving practices.

Future Scope

While the current system offers effective drowsiness detection and pattern analysis, several areas offer potential for further enhancement and expansion:

1. **Advanced Machine Learning Techniques**: Incorporating advanced machine learning models and techniques can enhance the accuracy of drowsiness detection and behavior analysis.

2. **Multi-Sensor Integration:** Integrating additional sensors, such as steering angle sensors or heart rate monitors, can provide a more holistic view of the driver's state.

3. **Cloud-Based Analysis**: Exploring cloud-based analytics can offer scalability for processing large volumes of data and provide insights into long-term driver behavior trends.

4. **Driver Profiling**: Developing personalized driver profiles based on behavior patterns can enable tailored alerts and coaching recommendations.

5. **Real-Time Intervention**: Implementing automated interventions, such as adjusting vehicle settings, can prevent accidents when critical drowsiness is detected.

6. **Fleet Management Integration**: Extending the system for fleet management purposes can support monitoring and coaching for professional drivers.

7. **Data Privacy and Security:** Ensuring robust data privacy mechanisms and encryption techniques to protect sensitive driver information.

8. **Collaboration with OEMs**: Collaborating with automotive manufacturers to integrate the system into vehicles as a built-in safety feature.

The proposed system's potential for continual improvement and expansion ensures that it remains relevant and valuable in promoting road safety and fostering responsible driving habits in the future.

REFERENCES

- Wang, W., & Wu, Z. (2018). A review of driver drowsiness monitoring systems. In *IOP Conference Series: Earth and Environmental Science* (Vol. 123, No. 1, p. 012055). IOP Publishing.
- Kaur, K., & Singh, S. (2020). A review on driver drowsiness detection systems. In *IOP Conference Series: Materials Science and Engineering* (Vol. 895, No. 1, p. 012062). IOP Publishing.
- Zhang, Y., & Wu, X. (2017). A survey on driver behavior analysis for intelligent transportation systems. *IEEE Transactions on Intelligent Transportation Systems*, 19(12), 3812-3825.

- Rakotonirainy, A., & Schroeter, R. (Eds.). (2017). *Driver Behaviour and Training* (Vol. 1). CRC Press.
- Pradhan, A. K. (Ed.). (2016). *Modeling, Simulation, and Optimization of Complex Processes: Proceedings of the International Conference on High Performance Scientific Computing, March 10-14, 2003, Hanoi, Vietnam*. Springer.

- National Highway Traffic Safety Administration (NHTSA). (n.d.). Drowsy Driving. Retrieved from https://www.nhtsa.gov/risky-driving/drowsy-driving
- European Commission - Mobility and Transport. (n.d.). Intelligent Transport Systems (ITS). Retrieved from https://ec.europa.eu/transport/themes/its_en

- International Conference on Intelligent Transportation Systems (ITSC): https://www.ieee-itss.org/its-conferences/itsc
- IEEE Intelligent Vehicles Symposium (IV): https://2021.ieee-iv.org/
- International Conference on Human-Computer Interaction (HCI): https://hcii2022.jinahya.fun/

- Virginia Tech Transportation Institute (VTTI): https://www.vtti.vt.edu/
- Transportation Research Institute - University of Michigan (UMTRI): http://www.umtri.umich.edu/