# ENGR 10 Final Project

# Project Purpose:

We did the mechanical engineering project

Simulate tools used in real-world automotive engineering.

Analyze time-series vehicle data like speed, fuel use, RPM, and distance.

Visualize performance trends with a PyQT6 dashboard.

```
from PyQt6.QtWidgets import (
    QApplication, QWidget, QLabel, QVBoxLayout, QHBoxLayout,
    QComboBox, QPushButton, QFileDialog, QLineEdit, QMessageBox
)
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.figure import Figure

#basic imports
import sys
import os
import datetime

#importing other files
from data_manager import DataManager
from graph_manager import GraphManager
from generate_vehicle_data import generate_vehicle_test_data
```

# Libraries:

Pandas: Data loading, filtering, and statistics

NumPy: For numerical ops

Matplotlib: Graphing acceleration, speed, fuel trends

PyQt6: GUI framework for interactive dashboard

os, sys: File handling and app launching

datetime: Timestamping generated data files

# data_manager.py: Data Handling & Analysis

Purpose: This module loads and filters vehicle test data and calculates statistical summaries for key performance parameters.

01

## load_data()

Loads vehicle data from a .csv file using Pandas. Extracts unique test IDs, vehicle types, and profile types. Returns True on success, and handles errors gracefully.

02

## get_test_data()

Filters the dataset by TestID, VehicleType, or ProfileType. Returns a DataFrame containing only the selected subset of the data.

03

## calculate_statistics()

Calculates descriptive statistics for a given parameter (e.g., Speed, Acceleration): Mean, median, standard deviation, min, max, range, Q1, Q3. Uses optional filters to narrow down to a specific subset of test data.

04

## summarize_all_parameters()

Automatically calculates statistics for multiple parameters: Speed, Acceleration, EngineRPM, FuelConsumption, Distance. Returns a dictionary of statistics for each parameter.

# graph_manager.py: Visualization Module

Purpose: This module creates plots to visualize trends and statistics from the dataset.

01

## plot_basic_stats()

Bar chart showing summary statistics (mean, median, etc.) for a selected parameter. Helps visualize variability and trends in the data.

02

## plot_acceleration()

Line graph of acceleration over time. Useful for identifying rapid starts or sudden changes in motion.

03

## plot_fuel_efficiency()

Line graph of fuel consumption over time. Reveals how efficiently a vehicle uses fuel under different profiles.

04

## plot_braking_events()

Plots acceleration data with a red line at -2.0 m/s² to show braking thresholds. Helps detect and analyze braking events.

# main.py: GUI Application with PyQt6

Purpose: This file launches the interactive dashboard and connects the UI to analysis features.

01

## VehicleDashboard ()

Uses PyQt6 to provide dropdown menus, buttons, and a plotting canvas. Allows users to load existing CSV files or generate new test data.

02

## UI Elements

File selector for loading data
Dropdowns for vehicle type and driving profile
Buttons for plotting: Basic Stats, Acceleration, Fuel Efficiency, Braking, Speed Motion
A FigureCanvas embeds Matplotlib plots directly into the window.

03

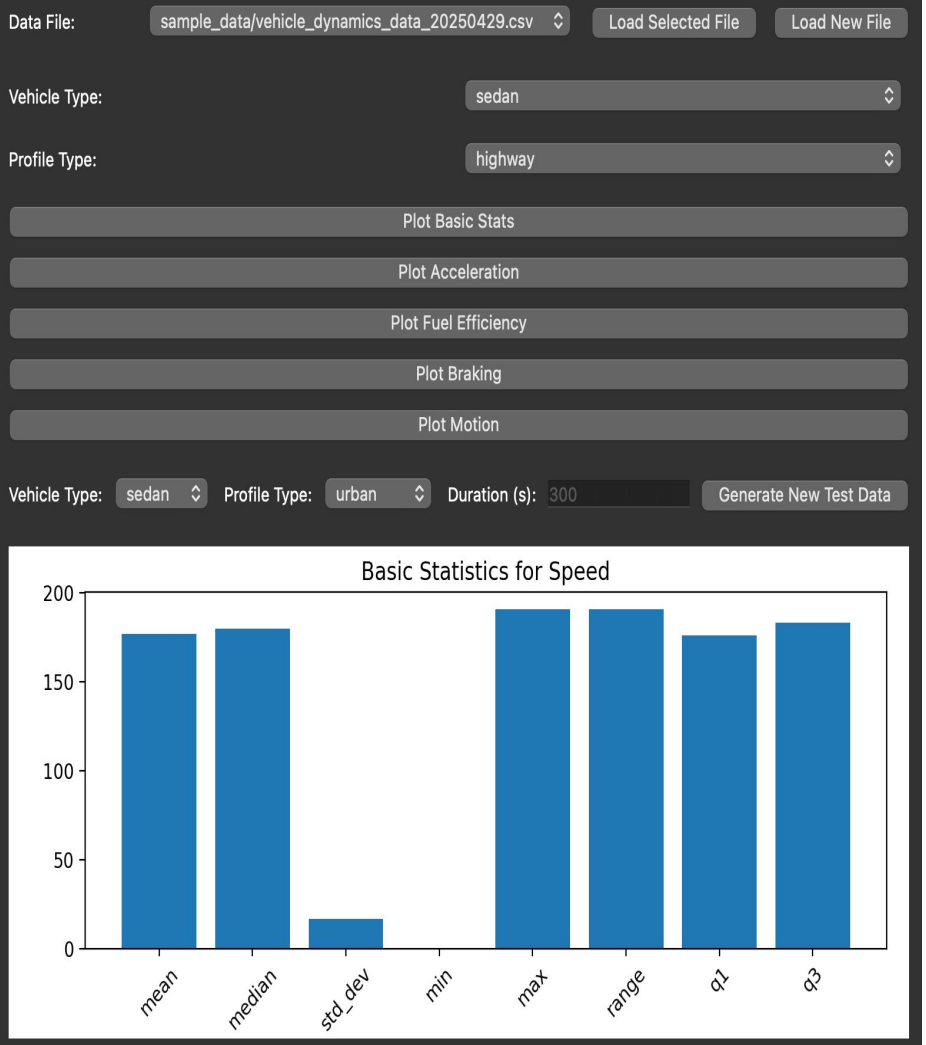
## Additional Features

generate_vehicle_test_data() creates synthetic test data based on user input. update_dropdowns() ensures only valid vehicle-profile combinations are shown. get_filters() dynamically extracts the user's current selection to pass to the plotting methods.

04

## Interactive Features

File Management: Dropdown to choose from available .csv files. "Load New File" button opens a file dialog to import custom data.
Filtering Controls: Vehicle and Profile dropdowns are dynamically updated based on loaded data. Ensures only valid combinations can be selected (e.g., SUV + highway).




# Example:

Here is an example of what our dashboard looks like. Each button corresponds to a graph that can be shown on the dashboard. It is fully interactive and also allows you to import your own CSV and create new specific ones.

```python
class VehicleDashboard(QWidget):  1 usage
    #initial constructor for data etc
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Vehicle Dynamics Dashboard")
        self.setGeometry(100, 100, 800, 600)

        # Initialize DataManager and GraphManager
        self.dm = DataManager()
        self.gm = GraphManager(self.dm)

        self.valid_combinations = []
        self.data_file_combo = QComboBox()
        self.vehicle_combo = QComboBox()
        self.profile_combo = QComboBox()
        self.plot_canvas = FigureCanvas(Figure(figsize=(6, 4)))

        self.init_ui()

        # Load the first file automatically if available
        if self.data_file_combo.count() > 0:
            self.load_selected_data_file()
```

# Code:

Here is an example of what the constructor for the main.py class looks like. It initializes all of the key elements that will be needed for the proper visualization of our graphs.

Throughout 3 Classes we wrote, we had a total of 25 methods. A majority of these methods were in the main.py file, and they just called the graphing methods to get the correct plots.

# GraphManager:

```python
import matplotlib.pyplot as plt

class GraphManager:  2 usages
    def __init__(self, data_manager):
        self.dm = data_manager

    #all of these methods are for plotting different types of graphs
    def plot_basic_stats(self, parameter, **filters):  1 usage
        stats = self.dm.calculate_statistics(parameter, **filters)
        if not stats:
            return None

        fig, ax = plt.subplots(figsize=(6, 4))
        labels = list(stats.keys())
        values = list(stats.values())
        ax.bar(labels, values)
        ax.set_title(f"Basic Statistics for {parameter}")
        ax.tick_params(axis='x', rotation=45)
        fig.tight_layout()
        return fig

    def plot_acceleration(self, **filters):  1 usage
        data = self.dm.get_test_data(**filters)
        if data is None:
            return None
```

# DataManager:

```python
import pandas as pd

class DataManager:  2 usages
    def __init__(self):
        self.data = None
        self.test_ids = []
        self.vehicle_types = []
        self.profile_types = []

    #load data method
    def load_data(self, filepath):  2 usages
        try:
            self.data = pd.read_csv(filepath)
            self.test_ids = self.data['TestID'].unique()
            self.vehicle_types = self.data['VehicleType'].unique()
            self.profile_types = self.data['ProfileType'].unique()
            return True
        except Exception as e:
            print(f"Error loading data: {e}")
            return False


    #get test data method
    def get_test_data(self, test_id=None, vehicle_type=None, profile_type=None):  5 usages (4 dynamic)
        if self.data is None:
            print("No data loaded.")
            return None
```

```python
#updating dropdowns method
def update_dropdowns(self, changed="vehicle"):  2 usages
    if self.user_updating:
        return

    self.user_updating = True

    selected_vehicle = self.vehicle_combo.currentText()
    selected_profile = self.profile_combo.currentText()

    if changed == "vehicle":
        #filter profile types for valid vehicle combo
        valid_profiles = sorted({p for v, p in self.valid_combinations if v == selected_vehicle})
        self.profile_combo.clear()
        self.profile_combo.addItems(valid_profiles)
        if selected_profile in valid_profiles:
            self.profile_combo.setCurrentText(selected_profile)
    elif changed == "profile":
        #filter vehicle types for valid profile combo
        valid_vehicles = sorted({v for v, p in self.valid_combinations if p == selected_profile})
        self.vehicle_combo.clear()
        self.vehicle_combo.addItems(valid_vehicles)
        if selected_vehicle in valid_vehicles:
            self.vehicle_combo.setCurrentText(selected_vehicle)

    self.user_updating = False
```

# Features:

One of our features, as mentioned before, is filtering. Vehicle and Profile dropdowns are dynamically updated based on loaded data. Ensures only valid combinations can be selected (e.g., SUV + highway). This was a key feature in our main method which made sure that the user was not able to try to generate and display data that did not exist.

Perfect — to embed these plots in your **PyQt6 dashboard**, you'll want the `GraphManager` methods to return the `matplotlib.figure.Figure` objects **instead of displaying them with** `plt.show()`.

### ✅ Updated `GraphManager` Methods (Return Plot Objects)

Here's the corrected version of your class:

python Copy Edit

```python
import matplotlib.pyplot as plt

class GraphManager:
    def __init__(self, data_manager):
        self.dm = data_manager

    def plot_basic_stats(self, parameter, **filters):
        stats = self.dm.calculate_statistics(parameter, **filters)
        if not stats:
            return None

        fig, ax = plt.subplots(figsize=(6, 4))
        labels = list(stats.keys())
        values = list(stats.values())
        ax.bar(labels, values)
        ax.set_title(f"Basic Statistics for {parameter}")
        ax.tick_params(axis='x', rotation=45)
        fig.tight_layout()
        return fig

    def plot_acceleration(self, **filters):
        data = self.dm.get_test_data(**filters)
        if data is None:
            return None

        fig, ax = plt.subplots(figsize=(10, 4))
        ax.plot(data['Time'], data['Acceleration'])
        ax.set_title("Acceleration Over Time")
        ax.set_xlabel("Time (s)")
        ax.set_ylabel("Acceleration (m/s²)")
        ax.grid(True)
        fig.tight_layout()
```

# Uses of AI:

The main use of AI in our project was for helping with the PyQT6 library. For example, one thing we prompted ChatGPT was for one of our bugs. The prompt was, "I need these to return the plots themselves. Please help me fix it so it can be displayed in the PyQT6 GUI." AI was very helpful in our project with debugging and also giving us ideas and tips to improve our code. Most of the other code did not require the use of AI except for improvements and debugging.

# Thank you!