Normal features:

1. Tabulate to create a table and create a nice UI for cars displayed at the database.
2. ANSI colour scheme for make the code more presentable using colours.
3. Error handling to work on wrong and duplicate inputs (ValueErrors)
4. Getter/setter methods for storing the data according to category.
5. Create an option to enter! to exit the code whenever they want to
6. Created an prompt so that EOFerrors, would display error message (ctrl + D)
7. Bubble sort in ascending order
8. Insertion sort in descending order
9. Created choices for user to move to and for.
10. Created invalid choices to prompt the code again until user chooses the correct option.

Additional features:

1. Created a search function for engine number to find the car and its details.
2. Created a filter function to filter the car by ANY category.
3. Created a function to delete a car from the database whenever.
4. Created a function to edit a cars detail from ANY category.
5. All of this having error handling with no issues (I hope there isn’t any more please don’t find any more)
6. Tabulate to store data in the database.
7. ANSI colour scheme to make the code more presentable using colours.

ICA #2:

Normal Features:

1. Modularity:
   1. The code is organized into specific functions that handle distinct tasks, such as display\_cars, edit\_car\_details, or cocktail\_sort\_by\_id. This modular approach makes it easier to understand, debug, and expand upon the existing functionality.
2. User Experience:
   1. Coloured Output: By using escape sequences to apply colour to the console output, the code enhances the user experience by making the interface visually appealing and more readable.
   2. Clear Prompts: The user is given clear instructions for every action they can take, reducing the chances of errors and confusion.
   3. Consistent Feedback: The system provides feedback after almost every action, informing the user of successful operations or explaining errors.
3. Error Handling & Input Validation:
   1. The application consistently checks the user's input for correctness. For instance, it validates car engine numbers, customer IDs, and email formats.
   2. Appropriate error messages guide the user in case of incorrect inputs, ensuring smooth interaction and data integrity.
4. Data Structuring & Representation:
   1. Tabular Data Display: The use of the tabulate library offers a structured and aesthetic view of data, making it easy for users to read and understand the displayed information.
   2. Object-Oriented Approach: Classes like Car and Customer allow for a clean representation of real-world entities, making data management and manipulation more intuitive.
5. Comprehensive Functionality:
   1. The application covers a wide range of functionalities from basic CRUD operations (Create, Read, Update, Delete) to advanced actions like sorting and filtering.
6. Multiple Sorting Methods:
   1. It's notable that the code incorporates several sorting algorithms like Bubble Sort, Insertion Sort, and Merge Sort. This provides flexibility in how data can be organized and can serve as an educational tool for understanding various sorting methods.
7. Intuitive Search & Filter Options:
   1. Users can search cars by engine numbers or filter them by categories, offering flexibility in data retrieval. Such features enhance usability and efficiency for the end-users.
8. Integration of Separate Modules:
   1. The system doesn't just manage cars; it also handles customer requests. The integration of both modules in a single application showcases a holistic approach to solving real-world problems.
9. Initial Data Setup:
   1. The application comes with sample car data. This makes it easier for first-time users to understand the system's functionality without needing to input data immediately.
10. ASCII Art for Branding:
    1. The ASCII header is not just visually appealing but can also be considered a branding or identity for the application, adding a unique touch.
11. Structured Menu System:
    1. The menu-driven approach is systematic and straightforward, guiding users step-by-step through the available functionalities.
12. Exit Mechanisms:
    1. The system provides clear exit points ('!' or option 'Exit'), ensuring users can easily terminate their sessions when needed.

Additional Features:

1. Added function Radix Sort for ANY category for choosing between ASCENDING or DESCENDING (sorts according to the bit size of the attribute.
2. Added Linear Search to search for an attribute. Fastest search for this code.
3. Split the customer requests with the management’s requests.
4. Created a Game utilising PyGame to show car traffic and earn points.
5. Cocktail sort to sort customer ID by ascending or descending.
6. Used Json and re to create a persistent database with CRUD attributes (to note: external libraries are needed to be installed and is the fastest database retrieval out of all programs
7. Customer requests are retrieved using the enumerate function to retrieve the number itself
8. Used ANSI code to colour the code from the output. Highlights all the important attributes and give the code a glow.
9. Error validation: ensured all codes are working properly without errors and prevent any wrong prompts into the code.
10. Used tabulate to sort and display the database and options in a table. This ensures the code can be easily seen and viewed.
11. Used a python package known as validate\_email to ensure that any email that was input in the code must be a registered email and cannot be a email that has never been created before. It also ensures that the prompt is in the right email format.