Design Document

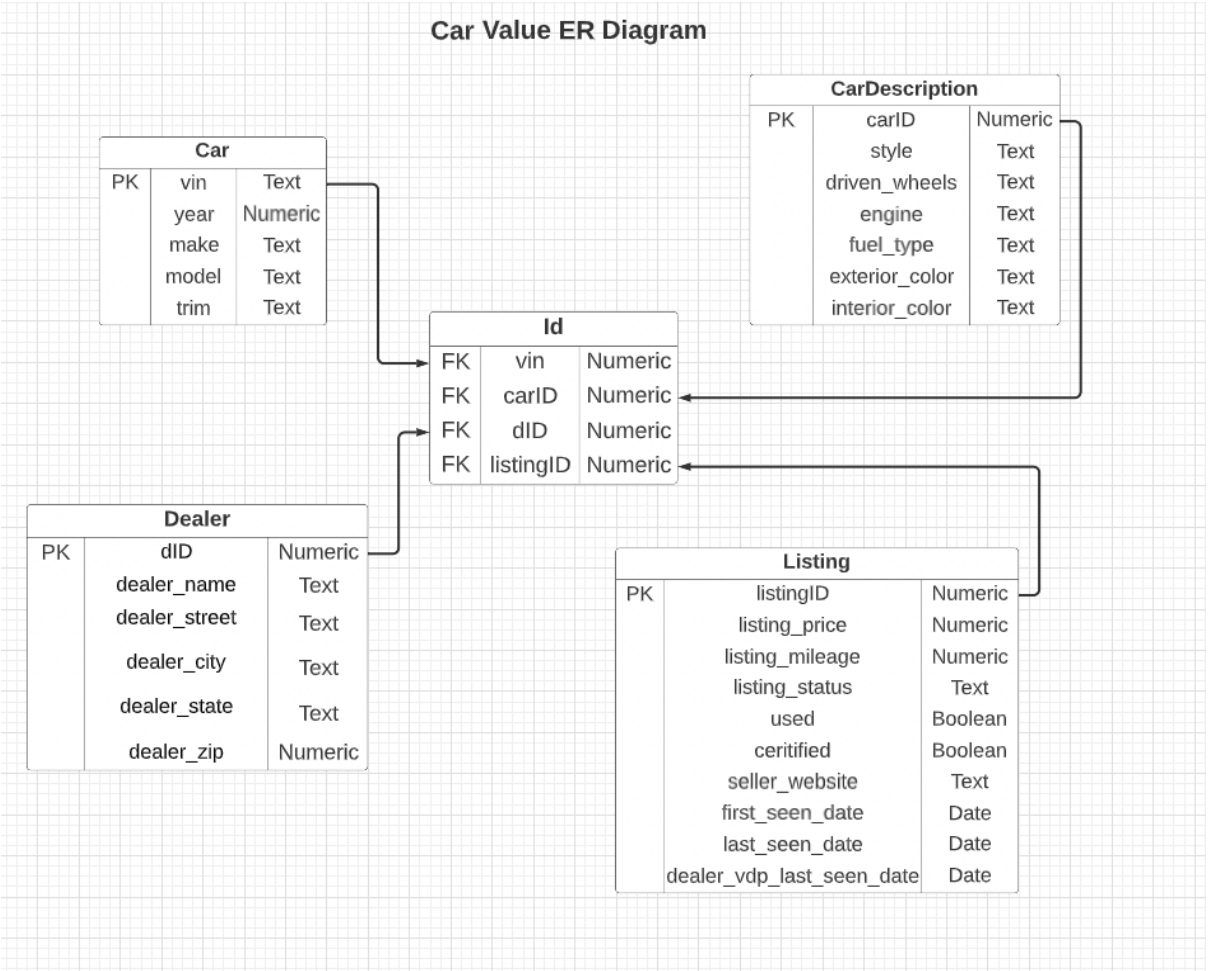
Connecting the MySQL database in Python

The first task was to connect the MySQL database in Python and import necessary libraries like pandas, mysql.connector.

Creating the database schema

Downloaded the Full Market Data File that was provided. Spend time understanding the data file and skim reading the file. Created a Normalised Relational Schema keeping in mind all the Normal forms (1NF, 2NF, 3NF, BCNF) The schema has tables and relationships between different entities.
There were 25 attributes in all. Five tables were created namely :
Car - vin, year, make, model, trim
CarDescription - carID, style, driven_wheels, engine, fuel_type, exterior_color, interior_color
Dealer - dID, dealer_name, dealer_street, dealer_city, dealer_state, dealer_zip
Listing - listingID, listing_price, listing_mileage, listing_status, used, certified, seller_website, first_seen_date, last_seen_date, dealer_vdp_last_seen_date
Id - vin(FK), carID(FK), dID(FK), listingID(FK)

All tables have unique primary keys. The Id table has three foreign keys which are referenced to primary keys in the respective table. Below is the **Entity Relationship Diagram**.



Creating the database
The database tables were created based on the schema and with the necessary data types for all fields.

Reading the data file
The entire text data file was imported in python.

Data Cleaning
Cleaning the data was one of the most important steps. The null values were removed from the data. The duplicate values were dropped and all the special characters were ignored.

Creating dataframes according to the schema
To transfer the entire data file into the database that was created, the text data had to be converted to rows and columns and this was done by creating dataframes for each entity.

- Creating unique IDs for CarID(using all fields), ListingID(seller website), dID(dealer_name and dealer_zip)

As there were three primary keys that were defined individually for each table and were not present in the datafile. The primary keys had to be given some unique values. To create unique values for the primary different approaches were followed as follows :
carID - The carID was created using all the fields of the CarDescription table (“style”_ “driven_wheels”_”engine_”fuel_type”_”exterior_color”_”interior_color”)
dID - The dealer was created using two fields of the Dealer table (“dealer_name”_”dealer_zip”)
listingID - The listingID was created using the seller_website domain name alone replacing the remaining URL.

Importing Data Frames to Database tables.
The final step for setting up the database was importing the data frames to the database tables. This datafile was very large so I created a sample test file containing the first 1000 rows and did all the testing on the smaller file and later when everything worked fine I did it on the big file which took a lot of time to process.

Creating a flask application
Now once the database setup was completed, a new flask application was created and a static User Interface was created to start working on. Once that was created the next task was to connect to the database (CarValue) that had all rows and columns. The data was then fetched from the database to display on the web page.

The Search.html page
The search.html page consists of the HTML code for fields to be entered by the user and the search buttons. It also has the jinja code that creates the table where the data will be displayed. The webpage has been designed keeping in mind that it should be mobile friendly.

The Index.css
An index.css file was created that handled all the styling of the web page. Including the tables, the headings and the formatting of the page.

The App.py file
The code in the app.py file creates the first endpoint to the search page. It renders the search.html page from the templates folder. Once the user enters the required details and presses the search button, it will use the value entered in the search field and perform the SQL query given and the data retrieved from the database is rendered via the POST request.

Testing the Frontend
The frontend has been tested on various user inputs :

- When the user enters only the make or the year or the model of the car alone the error is handled and the user is advised to provide the missing details.
- If the user enters a year which is not in the present or will be upcoming in the future, that is, any year greater than 2023 will throw an error stating to enter a valid year.
- If the user does not enter any value and directly clicks on the search button then the user is suggested to enter all details to proceed further.
- If the user enters a car mileage value which is negative or is zero then throw an error mentioning that the car mileage is invalid

Bonus
The following two factors/features are looked upon widely while buying a car :

- **Car trim** - It defines a group of features for a particular car. The base model is the standard having a lower cost value whereas the higher trims are costly and have additional features, technology, and performance enhancements.

- **Car Engine** - While buying a car the buyer looks into the type of car engine and decides on the basis of his use if he/she wants a smaller engine to save up on fuel and travel smaller distances or one has to drive the vehicle to greater limits and want a bigger engine.

Incorporating the **trim, engine factor** in the search will give a more accurate market value estimate.