Design Document: CarValue Trial Project

Sr.no	Name	Modified at
1	Sarim Sikander	10/04/2024
2		
3		

Contents

Database Schema

- **Primary Key (id)**:
  - The id column uniquely identifies each record in the vehicles table. It is indexed to ensure efficient retrieval and operations based on the primary key.
- **Indexes for Improved Performance**:
  - **year, make, model, trim**: Indexed to improve performance when searching for vehicles by their make, model, or year.
  - **dealer_city, dealer_state**: Indexed to allow efficient searches when filtering by geographic location.
  - **listing_price, listing_mileage**: Indexed to support efficient searching for vehicles within certain price or mileage ranges.
  - **vin**: Indexed and unique, as VIN is a unique identifier for vehicles. It ensures fast retrieval based on VIN searches.
  - **fuel_type**: Indexed to improve performance when searching for vehicles based on their fuel type.
- **Data Types**:
  - **Integer, String, Boolean**: Appropriate data types have been chosen for each field to reflect their real-world data. For example, Boolean is used for fields like used and certified, which can be either True or False.
  - **String(500)**: Used for fields that could potentially have long text values, such as dealer_name and engine.
- **Nullable Fields**:
  - Most columns are marked as nullable=True, allowing them to store missing data if certain fields are not available for a vehicle. This helps to handle the variability in the completeness of vehicle data.
- **Composite Indexes**:
  - To further improve performance for combined searches, composite indexes have been created:
    - **ix_vehicle_make_model**: Improves performance when querying vehicles by both make and model.
    - **ix_vehicle_price_mileage**: Helps when searching for vehicles based on price and mileage filters.

Use cases

- **Searching Vehicles by Attributes**:
  - Users can search for vehicles using attributes such as make, model, year, fuel_type, etc. The indexes ensure that these queries are fast and efficient.
- **Filtering by Location**:
  - Attributes like dealer_city and dealer_state are indexed to allow users to quickly filter vehicles based on geographic location, which is helpful for local searches.
- **Performance Optimization**:
  - The indexed columns were chosen based on common search criteria that users would likely use, such as vehicle make, model, year, price, and mileage.
  - Using a combination of individual and composite indexes helps optimize database operations without significantly impacting write performance.
- **Data Integrity**:
  - The vin column is marked as unique, ensuring that duplicate entries for the same vehicle cannot exist in the database.
  - Nullable fields allow flexibility in data entry, accommodating situations where some details might not be available.

DataFlow in the Application

The application has a clear data flow, which involves taking input data (e.g., vehicle information), processing it, storing it in a database, and ultimately presenting it through a web server. Below is a detailed description of each stage of the data flow:

Input Data Collection

The input data typically originates from an external source, such as a .txt file containing vehicle data, API input, or manual data entry through a web interface. Here's how the input data collection works:

- **Source**: The input data is provided in the form of a .txt file, CSV, or API payload, containing fields like year, make, model, listing_price, etc.
- **Format**: The file is delimited by a specific character (e.g., |), and contains rows representing different vehicles, with columns for various attributes.

Data Parsing and Validation

The collected input data is then parsed and validated before being stored in the database. This stage involves:

- **Reading the Data**: The file is read using Python (e.g., with pandas), and the data is stored in a DataFrame.
- **Validation**: The data is validated to ensure:
    - Required fields are not missing.
    - Fields match expected formats (e.g., year should be an integer).
    - Special characters or erroneous entries are handled.
    - NaN values are replaced or handled appropriately to ensure they do not cause issues during insertion.

Database Interaction (Data Insertion)

Once validated, the data is ready for storage in the database. Here's how this is handled:

- **SQLAlchemy ORM (Object-Relational Mapper)**: SQLAlchemy is used to interact with the database in an abstract way, converting Python objects into database records.
- **Session Management**: An SQLAlchemy session is created (get_session()) to manage the connection to the database.
- **Insert Data**:
    - The Vehicle model, representing the table schema, is used to create new instances for each row of the data.
    - These instances are added to the session and then committed to persist the changes in the database.
    - The startup script (startup_populate_data) checks whether the vehicles table already contains data. If empty, it will populate the database with data from the file.

Data Storage in the Database

The vehicle data is stored in a MySQL database in a table called vehicles. Key characteristics include:

- **Indexed Columns**: Frequently used fields (year, make, model, etc.) are indexed to optimize search performance.
- **Data Types**: Different columns have appropriate types (String, Integer, Boolean, etc.) to facilitate efficient storage and queries.
- **Integrity Constraints**: Columns like vin are marked as unique to ensure no duplicates exist.

Processing for Web Server Output

The stored data is queried and processed for presentation on the web server. This stage involves:

- **API Endpoint for Data Retrieval**:
    - **FastAPI Controller**: The data is exposed through an API endpoint implemented in FastAPI. For example, the /estimate endpoint allows users to search for vehicle estimates based on attributes such as year, make, and model.
    - **Controller Logic**: The EstimateController handles business logic for data retrieval. It interacts with the EstimateRepository to query the database based on input parameters, such as year and make.
    - **Mileage Adjustment**: If the mileage input is provided, the average price is adjusted to account for depreciation based on the mileage value using a formula (adjusted_price = base_price * max(adjustment_factor, 0)).
- **Data Filtering and Aggregation**:
    - SQLAlchemy queries are used to fetch data from the vehicles table, filtering based on user input.
    - The response is processed to calculate averages, apply adjustments (e.g., mileage adjustments), and select relevant samples for display.

Ideas to Improve Market value estimates

To improve the market value estimates for vehicles based on the provided data, there are several methods you can use that leverage additional features, better data handling, and advanced calculations.

Incorporate More Vehicle Attributes

The current model considers basic attributes like year, make, model, and mileage. To improve the accuracy of market value estimates, additional factors should be considered:

1. **Trim Level (trim)**:
    - Different trim levels often have significantly different features and market values. Including the trim information in the estimation model will increase accuracy.
2. **Vehicle Condition (used and certified)**:
    - A certified pre-owned (certified) vehicle typically has a higher market value compared to a non-certified used vehicle. The model should take into account whether the car is certified or not.
3. **Style and Configuration (style, driven_wheels, engine)**:
    - These attributes provide more insight into the vehicle's capability and desirability, influencing its price.
    - For example, SUVs generally have a higher value compared to sedans, and all-wheel drive (AWD) models may be more desirable in certain regions.
4. **Exterior and Interior Colors**:
    - Certain colors may be more popular or rare, which can have a positive or negative effect on value depending on market trends.
5. **Geographical Location (dealer_city, dealer_state)**:

    - The value of a vehicle can be influenced by where it is being sold. For instance, vehicles with all-wheel drive may have a higher demand in colder regions, while convertibles are more in demand in warmer climates.
6. **Seller Type (dealer_name, dealer_state)**:
    - Vehicles sold by well-known dealers or in certain regions may have higher prices. Consider creating a factor that adjusts the value based on the popularity or reputation of the dealership.

Mileage Adjustment with Enhanced Depreciation Formula

Currently, the depreciation rate used (0.000003 per mile) is static. You could improve accuracy by:

1. **Dynamic Depreciation Rate**:
    - **Mileage Buckets**: Different depreciation rates for different mileage brackets, such as:
        - 0 - 30,000 miles: Lower depreciation rate (the vehicle is still relatively new).
        - 30,001 - 70,000 miles: Moderate depreciation rate.
        - 70,001+ miles: Higher depreciation rate (as the vehicle ages, wear and tear become more significant).
2. **Depreciation by Age**:
    - Vehicles tend to lose value based on their age in addition to mileage. Adding a time-based depreciation component can improve accuracy. For example:
        - **First 3 Years**: High depreciation.
        - **Years 4-10**: Slower depreciation.
        - **10+ Years**: Minimal depreciation (if the vehicle is well-maintained).

Incorporate External Market Data

External data sources can provide a wealth of information to refine the estimates:

1. **Market Demand Data**:
    - Incorporate data from platforms like Kelley Blue Book, Edmunds, or TrueCar to adjust prices based on market trends.
    - Analyze whether certain models have increasing or decreasing market interest over time.
2. **Local Market Factors**:
    - Analyze recent sales data for similar vehicles in the local market.
    - Consider seasonal effects (e.g., convertibles may be in higher demand during the summer).
3. **Fuel Prices**:
    - Vehicles that are more fuel-efficient generally have better resale values during times of high fuel prices. Incorporate data on current fuel prices to adjust the value based on fuel type.

Price Trends and Vehicle Popularity

1. **Time Series Analysis**:
    - Perform a time-series analysis on historical price data for the same make and model to detect trends and predict future value more accurately.
    - Analyze how the market value of a specific vehicle changes over time based on factors such as the introduction of newer models.
2. **Vehicle Popularity**:
    - Utilize popularity data for the make and model. If a particular vehicle is trending or has a strong consumer following, adjust its value accordingly.