Project Design + Brief Project Implementation

I first imported the cars.csv file as a pandas DataFrame to view the data in the cars.csv file in a two-dimensional table in Python. Afterward, I explored the cars.csv data. I first identified the column names and the type of data stored in each of these columns. Then, I checked for missing values. If there were missing values, I decided to replace these values with the mean value for the data in the corresponding column. However, there were no missing values.

Then, I executed the necessary imports and established a connection to my local MySQL server.

The goal was to first partition the cars.csv file into multiple, smaller .csv files. The user had the opportunity to decide how many partitions to implement. Then, the data in each of the partitioned files would be stored in corresponding tables on the local MySQL server. Each time a partitioned data would need to be stored, the old table would be dropped and a new table will be created.

There were six search or analytics functions I sought to implement. The first one was to identify the total number of different car bodies for gas cars. The second one was similar. However, for the second function, the user can select which attribute to group the car data on and what attribute the corresponding total number would depend on. The third function allows users to identify cars within a user-inputted price range. The fourth function allows users to find the minimum or maximum price of cars with a certain attribute value. An example would be finding

the maximum price of gas vehicles. The fifth function would find the car IDs of cars with a certain attribute value. The sixth function would expand upon the fifth function in that the sixth function allows the user to identify the name of the car corresponding to a user-inputted car ID.

For each search or analytics function, the same map function, mapPartition(), was called. The purpose of this function was to map the user-constructed SQL statement to each of the MySQL tables containing the partitioned data. A global variable would then store the output from each table/partition. This output would then be accessed and processed by the corresponding reduce function that was called by the search or analytics function. As a result, this mapPartition() function simulates partition-based mapping behavior.

I decided to include 4 reduce functions. Each reduce function would reduce different types of input and return different outputs. Each search or analytics would call one of the reduce functions depending on the final goal of the search or analytics function. One reduce function, reduceGroupByCount(), would handle output from the map function from SQL statements containing the GROUP BY clause. So, for each distinct value of the attribute in the GROUP BY clause, the reduce function would sum the local, partition totals. The reduce function reduceSumTotal() would sum the integer, representing the total count, outputted by each partition. The third reduce function, reduceMaxOrMin(), would identify the maximum or minimum, depending on the user input, of the values outputted by each partition. Lastly, the fourth reduce function, reduceListOfValues(), would combine the list of values outputted by each partition.

Then, I wanted to incorporate Jupyter Widgets to allow the user to input filter characteristics for the search or analytics functions. The user would be able to customize, for instance, the SQL statement WHERE and GROUP BY clauses to gather the insights they desired. Additionally, I added user interaction for the user to indicate whether or not they would like the input and output for the mapping function and the search or analytics function's corresponding reduce function to be displayed. The added user interface would allow the app to be more engaging and allow the user to gain customizable, specific insight.

More Detailed Project Implementation

To complete the partitioning and storing of data, the parent method csvPartitionData() is called. This method has two parameters, the cars.csv file path and the number of desired partitions. The function calls upon the two helper methods described below to execute the partitioning and storing responsibilities. The output of this method, the names of the .csv files with the partitioned data, is stored in a list for future use.

The first helper method, called partitionData(), has two parameters, the cars.csv file and the number of partitions. This helper method's main responsibility is to take in the cars.csv file as a parameter and output the two partitioned, smaller CSV files. I implemented this partitioning using the Pandas library in Python, specifically leveraging the pandas.read_csv() function and the .to_csv() function after splitting the cars.csv based on the calculated chunk size. The chunk size is calculated by identifying the total number of rows in cars.csv and dividing that total by the number of desired partitions. I am also leveraging enumerate() to make the data splitting and partitioning simpler.

The second helper method, called savePartitionedToSQL(), has one parameter, a .csv file with partitioned data. This helper method is responsible for taking in the partitioned CSV file and saving the data in each CSV to the corresponding MySQL table. If the table for that CSV file has not been created yet, a MySQL table is created via another helper function called dropAndCreateTable(). Then, the data in the CSV file will be read. Each line of data will be inserted into the corresponding MySQL table. Once all the data for the partitioned file has been

saved and committed to the corresponding MySQL table, the program control will return to the parent method, csvPartitionData().

To complete the partition-based mapping, I created one function called mapPartition() that has four parameters. The first parameter is the name of the MySQL table containing data from one partition. The second parameter is a string containing the user-inputted SELECT clause for the mapped SQL statement. The third parameter is a string containing the user-inputted filtering (WHERE, GROUP BY, etc.) for the mapped SQL statement. The fourth parameter is a Boolean value on whether the user would like to see the inputs and outputs of this mapPartition() function. The function constructs one SQL statement, loops through each of the MySQL tables containing the partitioned data, and appends the output of the SQL statement to a list storing all the results of this function. The list storing the output of each partitioned table is declared outside the function to be accessed by other functions later on in the application.

To complete the partition-based reduce, I created 4 different reduce functions. The first reduce function, reduceGroupByCount(), would handle output from the map function from SQL statements containing the GROUP BY clause. The function has two parameters. The first parameter is the list of outputs from each MySQL table from the partition-based mapping. The second parameter is a Boolean value on whether the user would like to see the inputs and outputs of this reduceGroupByCount() function. A centralized dictionary would be initialized within this function. The function would then loop through each dictionary outputted by each MySQL table. If the key is not present in the centralized dictionary, the key and value are added. If the key is

already present, the new value is added to the existing value. The new dictionary is returned. Additionally, if the Boolean parameter is True, the inputs and outputs of the function are printed.

The second reduce function, reduceSumTotal(), has two parameters. The first parameter is the list of outputs from each MySQL table from the partition-based mapping. The second parameter is a Boolean value on whether the user would like to see the inputs and outputs of this reduceGroupByCount() function. A local integer variable is initialized. This reduce function would add the integer outputted by each MySQL table containing the partitioned table to the local integer variable. The sum is returned. Additionally, if the Boolean parameter is True, the inputs and outputs of the function are printed.

The third reduce function, reduceMaxOrMin(), has three parameters. The first parameter is the list of outputs from each MySQL table from the partition-based mapping. The second parameter is whether the user would like to know the maximum or minimum value. The third parameter is a Boolean value on whether the user would like to see the inputs and outputs of this reduceGroupByCount() function. A list is first initialized. Then, for instance, all the prices in the output from each MySQL table will be accessed and stored in that list. Then, if the user would like to find the maximum, a local integer variable is initialized. Each value in the local list will be looped through and the index of the maximum value will be returned. The same logic applies if the user would like to find the minimum value. Additionally, if the Boolean parameter is True, the inputs and outputs of the function are printed.

The fourth reduce function, reduceListOfValues(), has two parameters. The first parameter is the list of outputs from each MySQL table from the partition-based mapping. The second parameter is a Boolean value on whether the user would like to see the inputs and outputs of this reduceGroupByCount() function. A local list is initialized. The values outputted by each MySQL table containing the partitioned data are added to this local list if the value is not in the list already. In other words, this function would combine the list of values outputted by each partition. The local list with all the combined values will be returned. Additionally, if the Boolean parameter is True, the inputs and outputs of the function are printed.

Each of the search or analytics functions is described above. However, to add on, each search or analytics function calls the mapPartition() function and one reduce function depending on the end goal of the function. Before executing each function, user input on whether or not to display the inputs and outputs of the map and the corresponding reduce function. Additionally, user input on what to include in the SELECT clause and/or the WHERE or GROUP BY clauses for the mapped SQL statements is collected. Within each search or analytics function, the "global" list containing the results of the output from each partitioned data set is cleared to then populate the list with the new output. In each search or analytics function, the list containing the names of the partitioned .csv files is looped through to access the corresponding MySQL tables containing the partitioned data.

This section concludes the high-level and low-level overview of the project design and implementation.

Learning Experiences

I learned multiple different pieces of information from this project. On a technical level, I had the opportunity to furnish my pandas skills, specifically with reading in a .csv file and using enumerate() and .to_csv() to create partitioned data files. Additionally, I learned how to set up and connect to a local MySQL database using Python. This skill will benefit me immensely, especially when working on personal projects. In summary, I learned how to partition data without using in-built methods. By coding the partition and data saving to a MySQL database function, I was better able to understand their inner workings. Moreover, I learned about the importance and benefits of partitioning data. Partitioning can make data querying easier and faster to optimize performance and memory space.

Additionally, I better understood the concept of partition-based map and reduce through hands-on learning. These two functions enable scalability, can allow computations to be executed more efficiently at a larger scale, improve availability, and ease data management by enhancing controllability. Moreover, coding the app taught me how to achieve all of these benefits without showing the user how the data has been partitioned (unless the user selected the feature to display the inputs and outputs to the map and reduce functions). As a result, I was able to realize the power of map and reduce. Moreover, reduce is significant. Without reduce, the output from just a map function would confuse the user as the expected or accurate result will not be displayed. Map and reduce must coexist and work together to some degree. By coding these functions without using in-built methods, I was able to understand their inner logic on a more sophisticated level.

I also learned about the importance of user interaction. Without user interaction, the app would not be engaging or productive, as the user would not be able to gather specific insight. User interaction allowed the analysis to be customizable and thus have more value. Additionally, I learned that it is essential to think of the user's perspective when designing the app to identify what features or capabilities the user would benefit from when engaging with the app. For instance, I realized that adding a feature allowing the user to select whether or not they would like the inputs and outputs for the map and reduce functions to be displayed would assist in helping them look at the raw output from each partition and verify the inputs.

Lastly, by creating a timeline of the estimated completion dates of each task within this project, I improved my time management skills.

In summary, I learned on a technical, conceptual, and personal level by successfully designing and developing this app that allows for customizable search and analytics.

Project Code + Video

https://drive.google.com/drive/folders/1xoePD0vRa6_f21brDh5brqfEJPFPfNvb?usp=sharing

- The cars.csv file path is based on where it is stored on my machine, please update the file path for your machine. Additionally, as indicated in the video below, I used a Jupyter Notebook.

https://www.youtube.com/watch?v=fi0j4WR-Nk0

- YouTube link to my video of the project demo/design/implementation