REPORT. CAR DATA

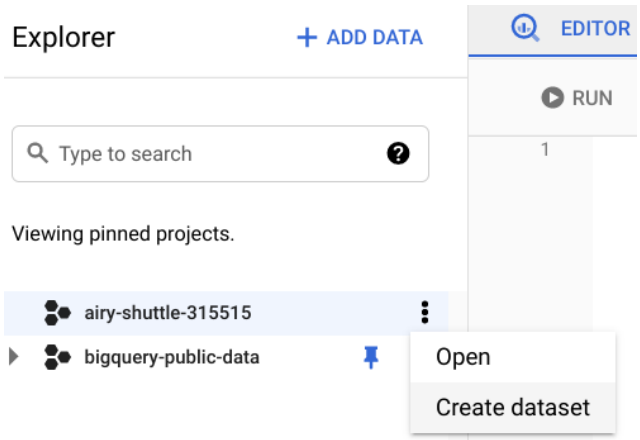
In this report, I outline the steps taken to explore and clean the selected data.

Selecting a Dataset

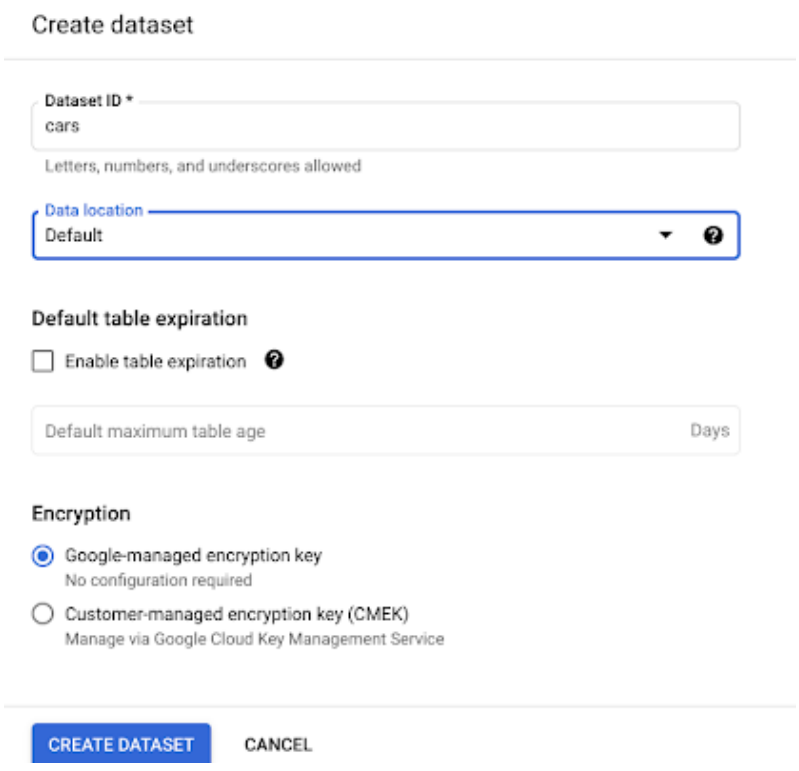
Using Big Query, I search for a dataset and create a custom table to store it, in this case `automobile_data`, to use SQL queries to explore and clean the data.

Step 1: Create a Dataset

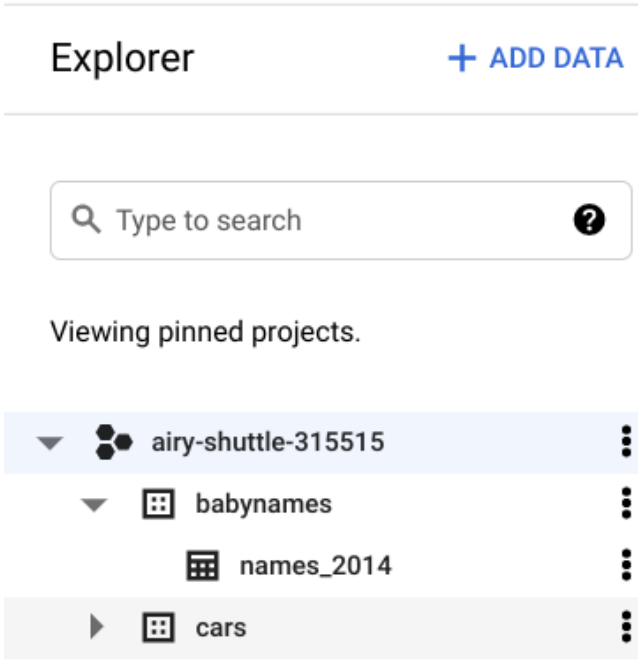
Once we open Big Query and download the `automobile_data` file, in the Explorer panel, we select *Create Dataset*.



From the *Create Dataset* menu, we complete the information about the dataset. Enter the Dataset ID as `cars` and click *CREATE DATASET*.



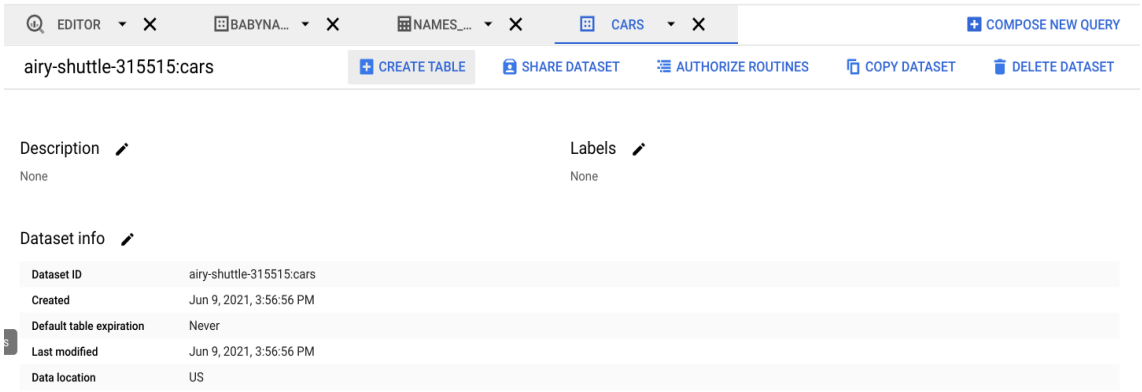
The cars dataset should appear under your project in the Explorer panel as shown below. Click on the three dots next to the `cars` dataset to open it.



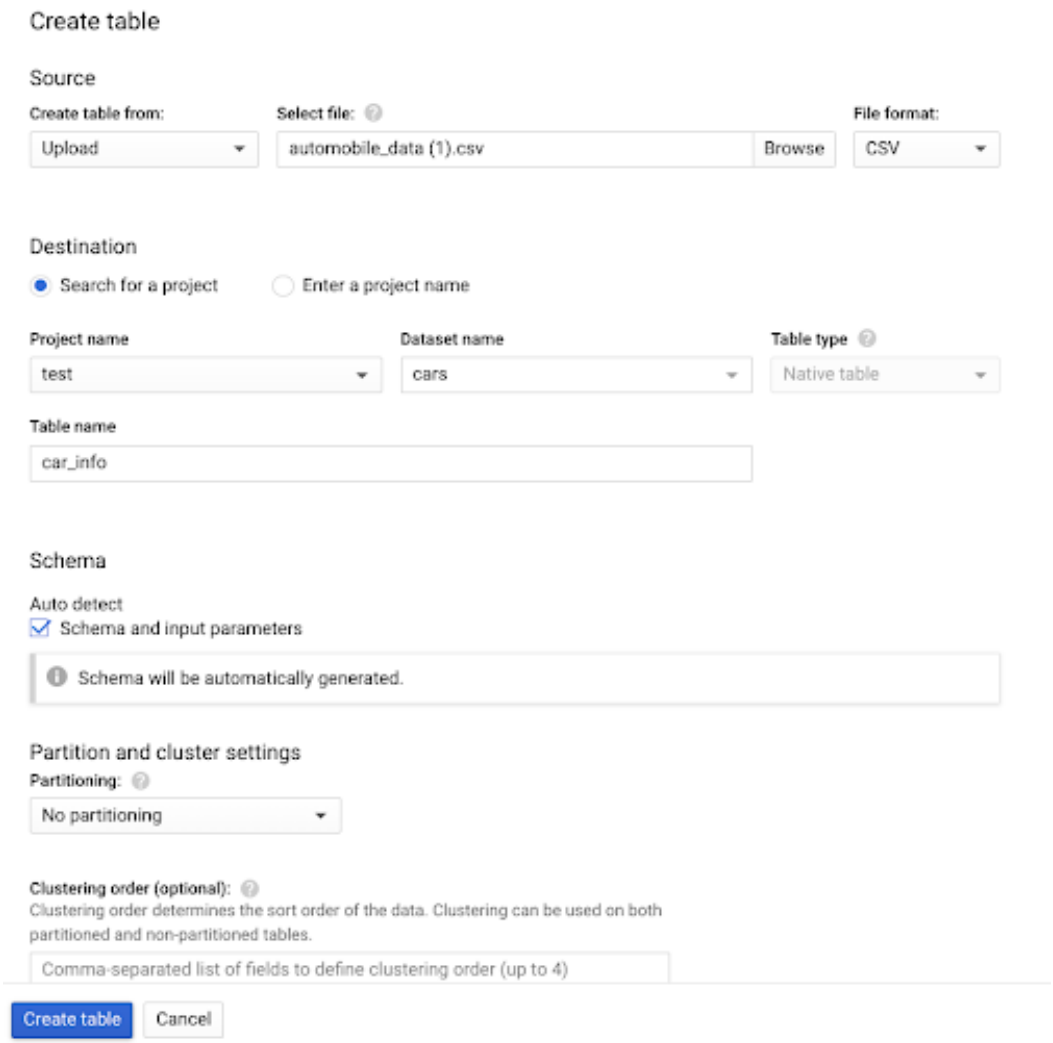
Step 2: Create a Table

After opening the newly created dataset, you can add a custom table for your data.

From the `cars` dataset, click *CREATE TABLE*.



In *Source*, upload the `automobile_data` CSV. In *Destination*, ensure that you are loading it into your `cars` dataset and name your table `car_info`. You can set the schema to *Auto-detect*. Then click *Create Table*.



Once the table is created, it will appear in the Explorer panel. You can click on the table to explore the schema and preview the data.

Data Cleaning

The new dataset contains historical sales information, including details such as car features and prices.

We will use this data to find the top 10 most popular cars and trims. But before performing the analysis, we must ensure that the data is clean, as analyzing dirty data could lead to presenting the wrong list of cars to investors, potentially causing them to lose money in their car inventory investment.

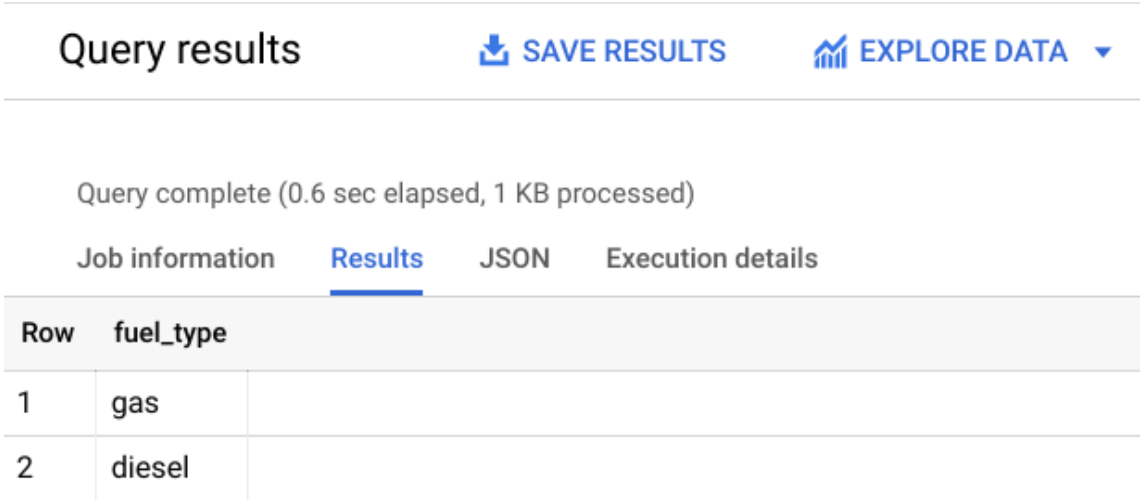
Step 1: Inspect the Data

The first thing we need to do is inspect the data in the table to see if any specific cleaning is necessary.

According to the data description, the `fuel_type` column should only have two string values: diesel or gas. To verify this, we run the following query:

```
SELECT DISTINCT fuel_type FROM cars.car_info;
```

This returns the following results:



Row	fuel_type
1	gas
2	diesel

This confirms that the `fuel_type` column does not have unexpected values.

Next, we inspect the `length` column, which should contain the minimum and maximum lengths of cars, which should match the data description. The lengths should range between 141.1 and 208.1.

We run this query to confirm:

```
SELECT MIN(length) AS min_length, MAX(length) AS max_length
FROM cars.car_info;
```

The results should confirm that 141.1 and 208.1 are the minimum and maximum values, respectively, for this column.

Row	min_length	max_length
1	141.1	208.1

Step 2: Fill Missing Data

Missing values can lead to errors or skewed results during analysis. It is necessary to check the data for null or missing values. These values can appear as a blank cell or the word "null."

I check if the `num_of_doors` column contains null values using this query:

```
SELECT *
FROM cars.car_info
WHERE num_of_doors IS NULL;
```

This will select all rows with missing data for the `num_of_doors` column and display them in the results table.

Row	make	fuel_type	num_of_doors	body_style
1	dodge	gas	*null*	sedan
2	mazda	diesel	*null*	sedan

We find that both Dodge and Mazda have incomplete data. To fill in these values, we check that all Dodge gasoline sedans and all Mazda diesel sedans sold had four doors.

To update, we use the following query:

```
UPDATE cars.car_info SET num_of_doors = "four"
WHERE make = "dodge" AND fuel_type = "gas" AND body_style = "sedan";
```

You should receive a message indicating that three rows have been modified in this table. To verify, you can re-run the previous query:

```
SELECT *
FROM cars.car_info
WHERE num_of_doors IS NULL;
```

Now, only one row has a NULL value for `num_of_doors`. We repeat this process to replace the null value for Mazda.

Step 3: Identify Possible Errors

Once you've ensured that there are no missing values in your data, we check for other potential errors.

In this case, we look at the `num_of_cylinders` column and use this query:

```
SELECT DISTINCT num_of_cylinders FROM cars.car_info;
```

After running this, we find that there is an extra row. There are two entries for two cylinders: rows 6 and 7. But the "two" in row 7 is misspelled.

Row	num_of_cylinders
1	four
2	six
3	five
4	three
5	twelve
6	two
7	tow
8	eight

To correct the spelling for all rows, we use the query:

```
UPDATE cars.car_info SET num_of_cylinders = "two"
WHERE num_of_cylinders = "tow";
```

You will receive a message alerting that one row has been modified after executing this statement. To check that it worked, we re-run the previous query:

```
SELECT DISTINCT num_of_cylinders FROM cars.car_info;
```

Next, we check the `compression_ratio` column. According to the data description, the values in the column should range from 7 to 23. As you did when checking length values, we use MIN and MAX to check if it's correct.

```
SELECT MIN(compression_ratio) AS min_compression_ratio,
MAX(compression_ratio) AS max_compression_ratio
FROM cars.car_info;
```

We find that the maximum is 70, but we know this is an error because the maximum value for this column should be 23. So, most likely, the 70 should be a 7.0. Re-run the previous query without the row with 70 to make sure the rest of the values are within the expected range of 7 to 23.

```
SELECT MIN(compression_ratio) AS min_compression_ratio,
MAX(compression_ratio) AS max_compression_ratio
FROM cars.car_info
WHERE compression_ratio <> 70;
```

Now, the highest value is 23, which matches the data description, so we correct the value of 70. We check with the sales manager, who says this row was entered by mistake and should be deleted. Before deleting anything, we check how many rows contain this incorrect value as a precaution to avoid accidentally deleting 50% of the data. If there are too many (for example, 20% of the rows have the incorrect value of 70), then you should check again with the sales manager to ask if they should be deleted or if the 70 should be updated to another value. Use the following query to count how many rows you would be deleting.

```
SELECT COUNT(*) AS num_of_rows_to_delete
FROM cars.car_info
WHERE compression_ratio = 70;
```

It turns out that only one row has the incorrect value of 70. So we can delete that row using the query:

```
DELETE FROM cars.car_info
WHERE compression_ratio = 70;
```

Step 4: Ensure Consistency

Finally, we must check that the data does not present inconsistencies that could cause errors. These inconsistencies can be difficult to detect; sometimes, even something as simple as an extra space can cause a problem.

We check if the `drive_wheels` column has inconsistencies by running a query with a SELECT DISTINCT statement:

```
SELECT DISTINCT drive_wheels FROM cars.car_info;
```

It seems that "4wd" appears twice in the results. However, since we used a SELECT DISTINCT statement to get unique values, it is likely that there is an extra space in one of the "4wd" entries, making it different from the others.

Row	drive_wheels
1	rwd
2	fwd
3	4wd
4	4wd

To check if this is the case, we use the LENGTH function to determine the length of each of these string variables:

```
SELECT DISTINCT drive_wheels, LENGTH(drive_wheels) AS string_length
FROM cars.car_info;
```

According to these results, some instances of the "4wd" string have four characters instead of the expected three ("4wd" has 3 characters). In that case, we use the TRIM function to remove all extra spaces in the `drive_wheels` column:

```
UPDATE cars.car_info SET drive_wheels = TRIM(drive_wheels)
WHERE TRUE;
```

Next, we re-run the SELECT DISTINCT statement to ensure that there are only three distinct values in the `drive_wheels` column:

```
SELECT DISTINCT drive_wheels
FROM cars.car_info;
```

In this case, the results return three unique values in this column.

After executing these steps, we obtain data that is clean, consistent, and ready for analysis.