

Clean Data using SQL

231030063

Tenzing Pema Thungon

Introduction

In this project, we focus on cleaning and preparing a dataset related to automobile sales using SQL. The objective is to identify the most popular car models by ensuring the data is accurate, consistent, and ready for analysis. Data cleaning is a crucial step because real-world datasets often contain errors, missing values, or inconsistencies, which can compromise the quality of analysis and decision-making. By meticulously cleaning the data, we create a reliable foundation for accurate insights, such as determining which cars a dealership should prioritize in their inventory.

Objectives

- **Ensure Data Accuracy:** Identify and correct errors, outliers, and inconsistencies in the dataset to maintain high data quality.
- **Handle Missing Data:** Detect and appropriately address missing values to ensure completeness of the dataset.
- **Optimize Data for Analysis:** Prepare the dataset for efficient querying and analysis by standardizing and cleaning data fields.
- **Enhance Query Performance:** Implement best practices in SQL to optimize the performance of data queries, particularly in large datasets.
- **Support Data-Driven Decision Making:** Provide a clean and reliable dataset that can be used to generate actionable insights, such as identifying the most popular car models for a dealership's inventory.

Methodology

Using Google Cloud BigQuery API

Link of the automobile_data spreassheets:

<https://docs.google.com/spreadsheets/d/13llypx5qtQEfpHpA7SjcI8rrGkHFfk389XBc7DNoluSA/edit?usp=sharing>

Step 1: Understand how data is measured

To get started, download the *automobile_data.csv* file. This is data from an external source that contains historical sales data on car prices and their features.

Click the link to the *automobile_data.csv* file to download it. Or you may download the .csv file directly from the attachments below.

Step 2: Create a dataset

Once you've downloaded the automobile_data.csv file, create your dataset.

Go to the Explorer pane in your workspace and click the three dots next to your personal project name to open the drop-down menu. From here, select Create dataset.

From the Create dataset menu, fill out some information about the dataset. Input the Dataset ID as cars you can keep the Location type as Multi-region, US (multiple regions in United States), and the Encryption as Google-managed encryption key default settings. Then, click the CREATE DATASET button.

The screenshot shows the BigQuery Studio interface within the Google Cloud Platform. The left sidebar contains navigation links for Analysis, Migration, and Administration. The main area features an 'Explorer' tab with a search bar and a 'Welcome to BigQuery Studio.' message. A dropdown menu is open over the 'Create new' button, showing options like 'Create data set', 'Refresh contents', 'Upload to project', and 'Change my default code region'. Below this, there are two promotional cards: 'Try the Google Trends demo query' and 'Try the Colab demo notebook'. The 'Create data set' dialog box is overlaid on the interface, containing fields for 'Dataset ID' (set to 'cars'), 'Location' (set to 'Multi-region, US'), and 'Encryption' (set to 'Google-managed encryption key'). Other options like 'Case insensitive.', 'Default Collation', 'Default rounding mode', 'Storage billing model', and 'Time travel window' are also visible. At the bottom right of the dialog are 'CREATE DATA SET' and 'CANCEL' buttons.

The screenshot shows the Google Cloud BigQuery Studio interface. The left sidebar has sections like Analysis, BigQuery Studio, Data transfers, Scheduled queries, Analytics Hub, Dataform, Partner Centre, Orchestration, Migration, Assessment, SQL translation, Administration, and Release notes. The main area is titled 'Welcome to BigQuery Studio.' It shows an 'Explorer' pane with a tree view under 'my-project1-433108' containing 'Queries', 'Notebooks', 'Data canvases', 'Data preparations', 'External connections', and the newly created 'cars' dataset. Below the tree view are buttons for 'SQL QUERY', 'PYTHON NOTEBOOK', and 'DATA CANVAS'. To the right, there's a section for 'Add your own data' with options for 'Local file', 'Google Drive', and 'Google Cloud Storage', each with a 'LAUNCH THIS GUIDE' button. A 'SUMMARY' tab at the bottom shows a 'Job history' with a message 'cars created.' and a 'GO TO DATA SET' button.

The **cars** dataset appears under our project in the Explorer pane as shown above.

Step 3: Create a table

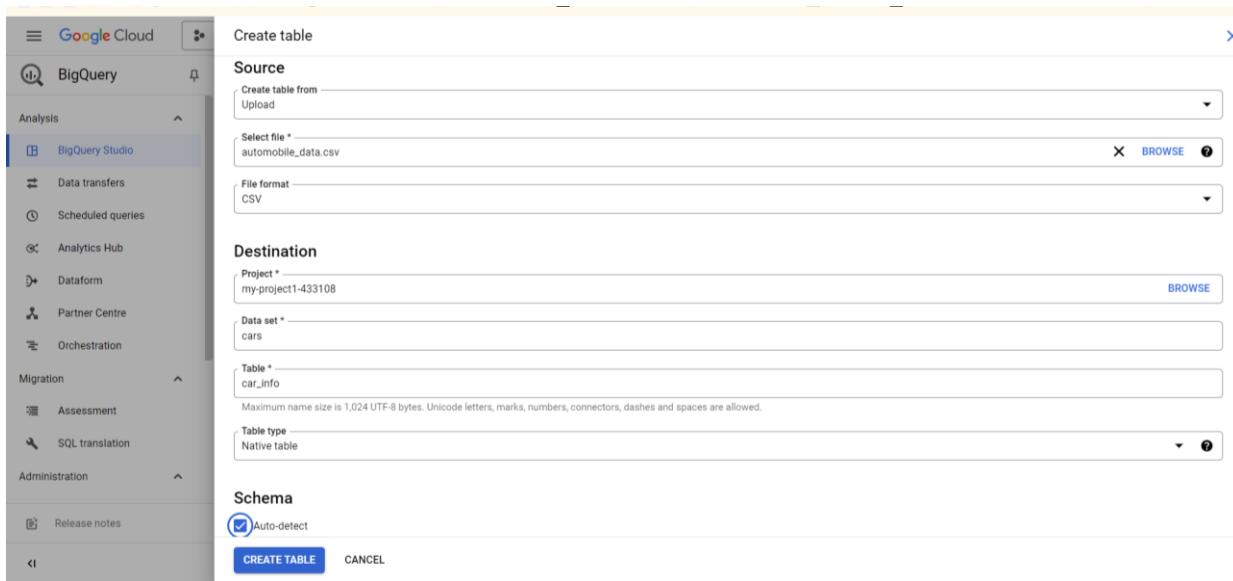
Now that we've created a dataset. We'll create a custom table to house our data. This will enable us to use SQL queries to explore and clean data.

After clicking on cars to open our newly created dataset, you will be able to add a custom table for the insertion of your downloaded data.

From the cars dataset info window, click CREATE TABLE.

This screenshot shows the 'cars' dataset info window in the BigQuery Studio. The top navigation bar includes 'Google Cloud', 'My Project1', a search bar, and 'DISMISS UPGRADE' buttons. The left sidebar is identical to the previous screenshot. The main area shows the 'cars' dataset details. A context menu is open over the dataset, with 'Create table' highlighted. Other options in the menu include 'Open', 'Share', 'Copy ID', 'Refresh contents', and 'Delete'. To the right of the dataset details, there are tabs for 'CREATE TABLE', 'SHARING', 'COPY', 'DELETE', and 'REFRESH'. Below the dataset details, there are sections for 'Dataset replica info' (with a 'PREVIEW' button) and 'Primary location' (set to US). At the bottom of the window, there are 'EDIT DETAILS' and 'VIEW REPLICAS' buttons.

Within the Create table window, upload the automobile_data.csv by clicking the drop-down arrow under Source and choosing the Upload option. Click the BROWSE button and navigate to the folder where your .csv document is located and notice the File format will automatically change to CSV. Ensure the dataset name is cars and name your table car_info. Set the schema to Auto-detect, and finally click the Create table button.



After creating our table, it will appear in our Explorer pane as we can see in the above image. We can click on the newly created table, car_info, to explore the SCHEMA and DETAILS buttons within your data page. Once you have gotten familiar with your data, you can start querying it.

Field name	Type	Mode	Key	Collation	Default value	Policy tags	Description
make	STRING	NULLABLE	-	-	-	-	-
fuel_type	STRING	NULLABLE	-	-	-	-	-
num_of_doors	STRING	NULLABLE	-	-	-	-	-
body_style	STRING	NULLABLE	-	-	-	-	-
drive_wheels	STRING	NULLABLE	-	-	-	-	-
engine_location	STRING	NULLABLE	-	-	-	-	-
wheel_base	FLOAT	NULLABLE	-	-	-	-	-
length	FLOAT	NULLABLE	-	-	-	-	-
width	FLOAT	NULLABLE	-	-	-	-	-

Here we could see SCHEMA, DETAILS,PREVIEW etc.

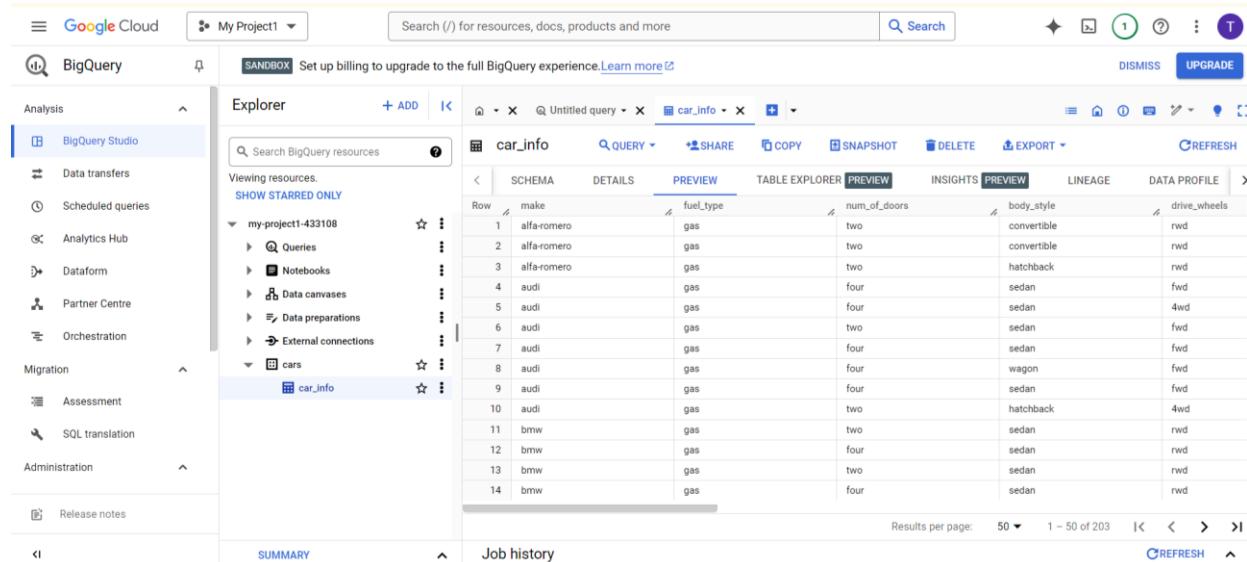
Step 4: Understanding why we need to clean our data

Our new dataset contains historical sales data, including details such as car features and prices. You can use this data to find the top 10 most popular cars and trims. But before you can perform our analysis, we'll need to make sure your data is clean. If we analyze dirty data, we could end up presenting the wrong list of cars to the investors. That may cause them to lose money on their car inventory investment.

Continue to next step to clean our data.

Step 5: Inspect the fuel_type column

The first thing we want to do is inspect the data in our table so we can find out if there is any specific cleaning that needs to be done. Get an initial understanding of the data table by clicking on the PREVIEW tab that sits below the car_info toolbar.



The screenshot shows the Google BigQuery interface. On the left, the sidebar includes sections like BigQuery Studio, Data transfers, Scheduled queries, Analytics Hub, Dataform, Partner Centre, Orchestration, Migration, Assessment, SQL translation, Administration, and Release notes. The main area is titled 'car_info' under 'Sandbox'. The toolbar at the top of the preview window has tabs for SCHEMA, DETAILS, and PREVIEW, with PREVIEW selected. Below the toolbar is a table with 14 rows of data. The columns are: Row, make, fuel_type, num_of_doors, body_style, and drive_wheels. The data is as follows:

Row	make	fuel_type	num_of_doors	body_style	drive_wheels
1	alfa-romero	gas	two	convertible	rwd
2	alfa-romero	gas	two	convertible	rwd
3	alfa-romero	gas	two	hatchback	rwd
4	audi	gas	four	sedan	fwd
5	audi	gas	four	sedan	4wd
6	audi	gas	two	sedan	fwd
7	audi	gas	four	sedan	fwd
8	audi	gas	four	wagon	fwd
9	audi	gas	four	sedan	fwd
10	audi	gas	two	hatchback	4wd
11	bmw	gas	two	sedan	rwd
12	bmw	gas	four	sedan	rwd
13	bmw	gas	two	sedan	rwd
14	bmw	gas	four	sedan	rwd

This is the PREVIEW table.

We know that the **fuel_type** column should only have two unique string values: **diesel** and **gas**. To check and make sure that's true, run the following query. You can generate the default query setup by clicking on the **QUERY** button and selecting the **In split tab**.

Data analytics

The screenshot shows the BigQuery interface with the 'car_info' table selected in the 'TABLE EXPLORER' view. The table has columns: Row, make, num_of_doors, body_style, and drive_wheels. The data preview shows rows 37 to 50, with values like 'honda', 'four', 'wagon', and 'fwd'. A tooltip indicates the table can also be viewed 'In Python notebook' or 'In data canvas'.

The screenshot shows the BigQuery interface with the 'car_info' table selected in the 'PREVIEW' view. The table has columns: Row, make, fuel_type, and body_style. The data preview shows rows 37 to 50, with values like 'honda', 'gas', and 'sedan'. To the right, an 'Untitled query' workspace is open, displaying the SQL query: `1 SELECT FROM my-project1-433108.cars.car_info LIMIT 1000`.

This shows us a dual view of the info window and the query.

Next, we can generate the first query in the workspace:

The screenshot shows the Google BigQuery interface. On the left, the sidebar includes sections for Analysis, BigQuery Studio, Data transfers, Scheduled queries, Analytics Hub, Dataform, Partner Centre, Orchestration, Migration, Assessment, SQL translation, Administration, and Release notes. The main area displays an 'Explorer' window for the 'car_info' table in the 'my-project1-433108' dataset. The 'SCHEMA' tab is selected, showing columns: Field name (make, fuel_type, num_of_doors, body_style, drive_wheels, engine_location, wheel_base, length, width), Type (STRING or FLOAT), Mode (NULLABLE), Key (-), and Coll (-). To the right, a query editor window titled 'Untitled query' contains the following SQL code:

```

1 #SELECT FROM `my-project1-433108.cars.car_info` LIMIT
2 1000
3 SELECT
4   DISTINCT fuel_type
5 FROM
6   `my-project1-433108.cars.car_info`
7 LIMIT 1000

```

The results section shows two rows of data under the 'RESULTS' tab:

fuel_type
gas
diesel

NOTE: Within the **FROM** clause of the syntax above, you will need to begin the **Table ID** line with your personalized project name, period, the dataset name, period, and end with the table name. It's important to understand that the personal project name will be unique to each learner. You can also locate and copy the full **Table ID** filename by clicking on the **DETAIL** option tab in your **car_info Table info** window. Once copied, paste it after the **FROM** clause and run the above query.

This returns the following results:

The screenshot shows the Google BigQuery interface, identical to the one above but with a different URL in the browser bar: 'console.cloud.google.com/bigquery?project=my-project1-433108&ws='.

The main area displays an 'Explorer' window for the 'car_info' table in the 'my-project1-433108' dataset. The 'SCHEMA' tab is selected, showing columns: Field name (make, fuel_type, num_of_doors, body_style, drive_wheels, engine_location, wheel_base, length, width), Type (STRING or FLOAT), Mode (NULLABLE), Key (-), and Coll (-). To the right, a query editor window titled 'Untitled query' contains the same SQL code as before:

```

1 #SELECT FROM `my-project1-433108.cars.car_info` LIMIT
2 1000
3 SELECT
4   DISTINCT fuel_type
5 FROM
6   `my-project1-433108.cars.car_info`
7 LIMIT 1000

```

The results section shows two rows of data under the 'RESULTS' tab:

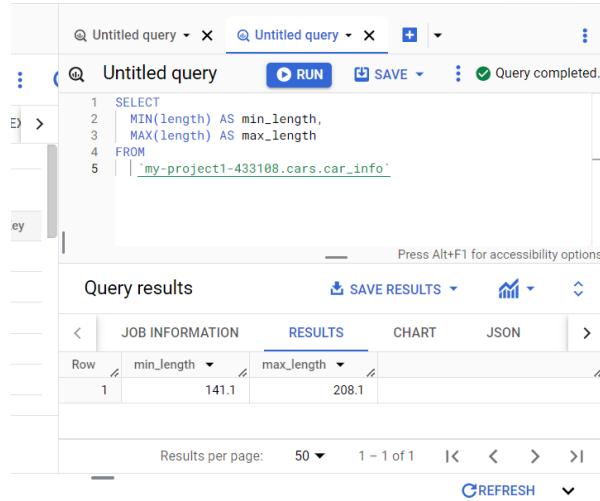
fuel_type
gas
diesel

We could see gas and diesel results at the bottom right.

This confirms that the **fuel_type** column doesn't have any unexpected values. Also note that the default **LIMIT 1000** is added to your query, but in this case, BigQuery is only returning two distinct fuel types.

Step 6: Inspect the length column

We inspect a column with numerical data. The **length** column should contain numeric measurements of the cars. So we will check that the minimum and maximum lengths in the dataset align with the [data description](#).



The screenshot shows the BigQuery interface with a query editor and a results viewer. The query is:

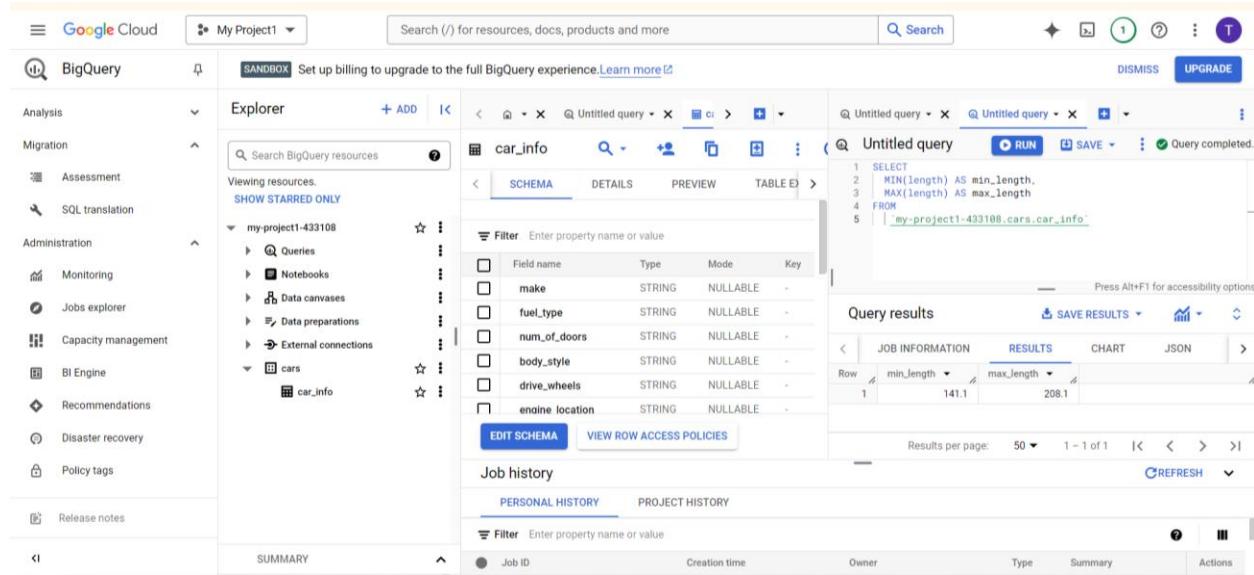
```

1 SELECT
2   MIN(length) AS min_length,
3   MAX(length) AS max_length
4 FROM
5   `my-project1-433108.cars.car_info`

```

The results table shows one row with the following data:

Row	min_length	max_length
1	141.1	208.1



The screenshot shows the Google Cloud BigQuery interface. On the left, the navigation pane includes Analysis, Migration, Assessment, SQL translation, Administration, Monitoring, Jobs explorer, Capacity management, BI Engine, Recommendations, Disaster recovery, Policy tags, and Release notes. The main area shows the Explorer tab for the project 'my-project1-433108'. A table named 'car_info' is selected, showing its schema with fields: make (STRING, NULLABLE), fuel_type (STRING, NULLABLE), num_of_doors (STRING, NULLABLE), body_style (STRING, NULLABLE), drive_wheels (STRING, NULLABLE), engine_location (STRING, NULLABLE). Below the schema, there is a 'EDIT SCHEMA' button and a 'VIEW ROW ACCESS POLICIES' button. To the right, the results of the previous query are displayed in a results viewer, identical to the one above.

Our results confirm that 141.1 and 208.1 are the minimum and maximum values respectively in this column.

Step: Fill in missing data

Missing values can create errors or skew our results during analysis. We're going to want to check our data for null or missing values. These values might appear as a blank cell or the word null in BigQuery.

We can check to see if the **num_of_doors** column contains null values using this query:

The screenshot shows the Google Cloud BigQuery interface. On the left, the sidebar includes sections for Analysis, Migration, Assessment, SQL translation, Administration, Monitoring, Jobs explorer, Capacity management, BI Engine, Recommendations, Disaster recovery, Policy tags, and Release notes. The main area shows an 'Explorer' view for the 'car_info' table under the 'my-project1-433108' dataset. The schema for 'car_info' is displayed, showing columns: Field name (make, fuel_type, num_of_doors, body_style, drive_wheels, engine_location, wheel_base), Type (STRING, STRING, STRING, STRING, STRING, STRING, FLOAT), Mode (NULLABLE, NULLABLE, NULLABLE, NULLABLE, NULLABLE, NULLABLE, NULLABLE), and Key (-, -, -, -, -, -, -). A query editor window titled 'Untitled query' contains the following SQL code:

```

1 SELECT
2   *
3   FROM
4     `my-project1-433108.cars.car_info`
5   WHERE
6     num_of_doors IS NULL;

```

The 'Query results' section shows two rows of data from the 'car_info' table:

Row	make	fuel_type	num_of_doors
1	dodge	gas	null
2	mazda	diesel	null

This will select rows with missing data for the **num_of_doors** column and return them in our results table. We get two results, one Mazda and one Dodge as shown in the above image.

In order to fill in these missing values, we check with the sales manager, who states that all Dodge gas sedans and all Mazda diesel sedans sold had four doors. If we are using the BigQuery free trial, we can use this query to update our table so that all Dodge gas sedans have four doors: we use UPDATE, SET, WHERE

The screenshot shows the Google Cloud BigQuery interface. The sidebar and table structure are identical to the previous screenshot. The query editor window titled 'Untitled query' contains the following SQL code:

```

1 UPDATE
2   `my-project1-433108.cars.car_info`
3   SET
4     num_of_doors = "four"
5   WHERE
6     make = "dodge"
7     AND fuel_type = "gas"
8     AND body_style = "sedan";

```

The 'Query results' section shows a message indicating that the statement modified 3 rows in 'car_info'. A 'GO TO TABLE' button is available to view the updated data.

We get a message telling that three rows were modified in this table. To make sure, we can run the previous query again: using SELECT, FROM, WHERE

The screenshot shows the Google BigQuery interface. On the left, the sidebar includes sections like Analysis, Migration, Assessment, SQL translation, Administration, Monitoring, Jobs explorer, Capacity management, BI Engine, Recommendations, Disaster recovery, Policy tags, and Release notes. The main area displays the 'car_info' table schema and its contents. The schema includes fields: Field name (make, fuel_type, num_of_doors, body_style, drive_wheels, engine_location, wheel_base, length), Type (STRING or FLOAT), Mode (NULLABLE or nullable), and Key (indicated by a key icon). The results table shows one row: make (mazda), fuel_type (diesel), and num_of_doors (null). A query editor window is open with the following SQL code:

```

1 SELECT
2 *
3 FROM
4 `my-project1-433108.cars.car_info`
5 WHERE
6 num_of_doors IS NULL;
    
```

The results panel shows the same row with num_of_doors as null.

Results show that we only have one row with a null value for **num_of_doors**. Repeat this process to replace the null value for the Mazda.

The screenshot shows the Google BigQuery interface. The sidebar and table schema are identical to the previous screenshot. The query editor contains an UPDATE statement:

```

1 UPDATE
2 `my-project1-433108.cars.car_info`
3 SET
4 num_of_doors = "four"
5 WHERE
6 make = "mazda"
7 AND fuel_type = "diesel"
8 AND body_style = "sedan";
    
```

The results panel shows a message: "This statement modified 2 rows in car_info." A "GO TO TABLE" button is also present.

We get a message telling that two rows were modified in this table. To make sure, we can run the previous query again: using SELECT, FROM, WHERE

The screenshot shows the Google BigQuery interface. On the left, the sidebar includes sections for Analysis, Migration, Assessment, SQL translation, Administration, Monitoring, Jobs explorer, Capacity management, BI Engine, Recommendations, Disaster recovery, Policy tags, and Release notes. The main area displays the schema for a table named 'car_info' with fields: Field name (make, fuel_type, num_of_doors, body_style, drive_wheels, engine_location, wheel_base, length), Type (STRING, STRING, STRING, STRING, STRING, STRING, FLOAT, FLOAT), and Mode (NULLABLE, NULLABLE, NULLABLE, NULLABLE, NULLABLE, NULLABLE, NULLABLE, NULLABLE). A query editor window titled 'Untitled query' contains the following SQL code:

```

1 SELECT
2   *
3   FROM
4     `my-project1-433108.cars.car_info`
5   WHERE
6     num_of_doors IS NULL;

```

The results pane below shows the message: "There is no data to display." with a results per page dropdown set to 50.

We now see that there is no data to display.

Step 8: Identify potential errors

Once we have finished ensuring that there aren't any missing values in our data, we'll want to check for other potential errors. We can use **SELECT DISTINCT** to check what values exist in a column. We can run this query to check the **num_of_cylinders** column:

The screenshot shows the Google BigQuery interface. The schema for the 'car_info' table is identical to the previous screenshot. The query editor window now contains the following SQL code:

```

1 SELECT
2   DISTINCT num_of_cylinders
3   FROM
4     `my-project1-433108.cars.car_info`;

```

The results pane shows the following data:

Row	num_of_cylinders
1	six
2	eight
3	three
4	two

Results per page: 50

After running this, we notice that there are one too many rows. There are two entries for two cylinders: rows 4 and 5. But the *two* in row 5 is misspelled.

The screenshot shows the Google BigQuery web interface. On the left, there's a sidebar with navigation links like Google Cloud, My Project1, Analysis, Migration, Assessment, SQL translation, Administration, Monitoring, Jobs explorer, Capacity management, BI Engine, Recommendations, Disaster recovery, Policy tags, and Release notes. Below the sidebar, there's a search bar and a 'Filter' input field.

The main area has two tabs: 'JOB INFORMATION' and 'RESULTS'. The 'RESULTS' tab is selected, displaying a table titled 'Query results' with one column 'num_of_cylinders' and eight rows containing values: six, eight, three, two, tow, twelve, four, and five. Below the table are buttons for 'SAVE RESULTS', 'EXPLORE DATA', and 'EXECUTION DETAILS'. At the bottom, there are buttons for 'EDIT SCHEMA' and 'VIEW ROW ACCESS POLICIES'.

Below the main table, there's a 'Job history' section with tabs for 'PERSONAL HISTORY' and 'PROJECT HISTORY'. There's also a 'REFRESH' button and a help icon.

To correct the misspelling for all rows, we can run this query if you have the BigQuery free trial:
Query contains UPDATE, SET, WHERE

This screenshot shows the Google BigQuery interface with a different view. The left sidebar includes 'BigQuery' under 'Analysis' and 'car_info' under 'Viewing resources'. The main area features a 'Untitled query' editor with the following SQL code:

```

1 UPDATE
2   `my-project1-433108.cars.car_info`
3   SET
4     num_of_cylinders = "two"
5   WHERE
6     num_of_cylinders = "tow";

```

Below the query editor, there's a 'Query results' section with tabs for 'JOB INFORMATION', 'RESULTS', 'EXECUTION DETAILS', and 'EXECUTION GRAPH'. A message indicates that the statement modified 1 row in car_info. There are also 'GO TO TABLE' and 'REFRESH' buttons.

The results show this statement modified one row in car_info. To check that it worked, we can run the previous query having SELECT, DISTINCT again:

The screenshot shows the Google Cloud BigQuery interface. On the left, the navigation pane includes sections like Analysis, Migration, Assessment, SQL translation, Administration, Monitoring, Jobs explorer, Capacity management, BI Engine, Recommendations, Disaster recovery, Policy tags, and Release notes. The main area displays the schema for the 'car_info' table, which has fields: make (STRING, NULLABLE), fuel_type (STRING, NULLABLE), num_of_doors (STRING, NULLABLE), body_style (STRING, NULLABLE), drive_wheels (STRING, NULLABLE), engine_location (STRING, NULLABLE), wheel_base (FLOAT, NULLABLE), length (FLOAT, NULLABLE), and num_of_cylinders (FLOAT, NULLABLE). Below the schema, a query results table is shown with the following data:

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

The results show that correction were carried out successfully.

Next, we can check the **compression_ratio** column. According to the [data description](#), the **compression_ratio** column values should range from 7 to 23. Just like when you checked the length values, we can use **MIN** and **MAX** to check if that's correct:

The screenshot shows the Google Cloud BigQuery interface. The schema for the 'car_info' table is identical to the previous screenshot. A new query is run to find the minimum and maximum values of the 'compression_ratio' column:

```

1 SELECT
2   MIN(compression_ratio) AS min_compression_ratio,
3   MAX(compression_ratio) AS max_compression_ratio
4 FROM
5   `my-project1-433108.cars.car_info`

```

The resulting table shows the minimum and maximum compression ratio values:

min_compression_ratio	max_compression_ratio
7.0	70.0

We find that **this returns a maximum of 70**. But we know this is an error because the maximum value in this column should be 23, not 70. So the 70 is most likely a 7.0. Run the above query again without the row with 70 to make sure that the rest of the values fall within the expected range of 7 to 23.

The screenshot shows the Google BigQuery interface. On the left, the navigation pane includes 'Analysis', 'Migration', 'Assessment', 'SQL translation', 'Administration', 'Monitoring', 'Jobs explorer', 'Capacity management', 'BI Engine', 'Recommendations', 'Disaster recovery', 'Policy tags', and 'Release notes'. The main area displays a schema for a 'car_info' table with fields: make (STRING, NULLABLE), fuel_type (STRING, NULLABLE), num_of_doors (STRING, NULLABLE), body_style (STRING, NULLABLE), drive_wheels (STRING, NULLABLE), engine_location (STRING, NULLABLE), wheel_base (FLOAT, NULLABLE), length (FLOAT, NULLABLE), and price (FLOAT, NULLABLE). A query titled 'Untitled query' is running, showing the following SQL code:

```

1 SELECT
2   MIN(compression_ratio) AS min_compression_ratio,
3   MAX(compression_ratio) AS max_compression_ratio
4 FROM
5   `my-project1-433108.cars.car_info`
6 WHERE
7   compression_ratio >= 70;

```

The 'Query results' section shows the output of the query:

	min_compression_ratio	max_compression_ratio
1	7.0	23.0

Results per page: 50 | 1 - 1 of 1 | Refresh

Now the highest value is 23, which aligns with the data description. So we'll want to correct the 70 value. We check with the sales manager again, who says that this row was made in error and should be removed. Before we delete anything, we should check to see how many rows contain this erroneous value as a precaution so that you don't end up deleting 50% of our data. If there are too many (for instance, 20% of our rows have the incorrect 70 value), then we would want to check back in with the sales manager to inquire if these should be deleted or if the 70 should be updated to another value. Use the query `SELECT COUNT` below to count how many rows we would be deleting:

The screenshot shows the Google BigQuery interface. The schema for the 'car_info' table is identical to the previous screenshot. A new query titled 'Untitled query' is running, showing the following SQL code:

```

1 SELECT
2   COUNT(*) AS num_of_rows_to_delete
3 FROM
4   `my-project1-433108.cars.car_info`
5 WHERE
6   compression_ratio = 70;

```

The 'Query results' section shows the output of the query:

	num_of_rows_to_delete
1	1

Results per page: 50 | 1 - 1 of 1 | Refresh

Turns out there is only one row with the erroneous 70 value. So we can delete that row using this query `DELETE WHERE`:

The screenshot shows the Google BigQuery interface within the Google Cloud Platform. On the left, the navigation pane includes sections like Analysis, Migration, Assessment, SQL translation, Administration, Monitoring, Jobs explorer, Capacity management, BI Engine, Recommendations, Disaster recovery, Policy tags, and Release notes. The main area displays a table named 'car_info' with columns: Field name, Type, and Mode. The table contains several rows of data. A query editor window is open with the following SQL code:

```

1 DELETE `my-project1-433108.cars.car_info`
2 WHERE compression_ratio = 70;

```

The 'Query results' section indicates that the statement removed 1 row from the 'car_info' table.

We find that this statement removed 1 row, we can also check using the previously used query
SELECT DISTINCT

Step 9: Ensure Consistency

Finally, we want to check our data for any inconsistencies that might cause errors. These inconsistencies can be tricky to spot—sometimes even something as simple as an extra space can cause a problem.

Check the **drive_wheels** column for inconsistencies by running a query with a **SELECT DISTINCT** statement:

The screenshot shows the Google BigQuery interface within the Google Cloud Platform. The navigation pane and table structure are identical to the previous screenshot. The query editor window now contains the following SQL code:

```

1 SELECT
2   DISTINCT drive_wheels
3   FROM
4   `my-project1-433108.cars.car_info`;

```

The 'Query results' section displays the distinct values found in the 'drive_wheels' column:

drive_wheels
rwd
fwd
4wd
4wd

It appears that 4wd appears twice in results. However, because we used a **SELECT DISTINCT** statement to return unique values, this probably means there's an extra space in one of the 4wd entries that makes it different from the other 4wd.

To check if this is the case, we can use a **LENGTH** statement to determine the length of how long each of these string variables:

The screenshot shows the Google Cloud BigQuery interface. On the left, the sidebar includes Analysis, Migration, Assessment, SQL translation, Administration, Monitoring, Jobs explorer, Capacity management, BI Engine, Recommendations, Disaster recovery, Policy tags, and Release notes. The main area shows the schema for the 'car_info' table, which has columns: Field name (make, fuel_type, num_of_doors, body_style, drive_wheels, engine_location, wheel_base, length), Type (STRING, STRING, STRING, STRING, STRING, STRING, FLOAT, FLOAT), and Mode (NULLABLE, NULLABLE, NULLABLE, NULLABLE, NULLABLE, NULLABLE, NULLABLE, NULLABLE). A query editor window is open with the following SQL code:

```

1 SELECT
2   DISTINCT drive_wheels,
3   LENGTH(drive_wheels) AS string_length
4 FROM
5   `my-project1-433108.cars.car_info`

```

The results table shows the following data:

drive_wheels	string_length
rwd	3
fwd	3
4wd	3
4wd	4

According to the results, some instances of the 4wd string have four characters instead of the expected three (4wd has 3 characters). In that case, we can use the **TRIM** function to remove all extra spaces in the **drive_wheels** column if you are using the BigQuery free trial:

The screenshot shows the Google Cloud BigQuery interface. The sidebar and schema for the 'car_info' table are identical to the previous screenshot. The query editor window now contains an **UPDATE** statement:

```

1 UPDATE
2   `my-project1-433108.cars.car_info`
3 SET
4   | drive_wheels = TRIM(drive_wheels)
5 WHERE TRUE;

```

The results table shows a message: "This statement modified 202 rows in car_info." A "GO TO TABLE" button is also present.

The results show that this statement modified 202 rows.

Then, we run the **SELECT DISTINCT** statement again to ensure that there are only three distinct values in the `drive_wheels` column:

The screenshot shows the Google BigQuery web interface. On the left, the sidebar includes sections like Analysis, Migration, Assessment, SQL translation, Administration, Monitoring, Jobs explorer, Capacity management, BI Engine, Recommendations, Disaster recovery, Policy tags, and Release notes. The main area has tabs for Explore, Schema, Details, and Preview. A search bar at the top says "Search (/) for resources, docs, products and more". Below it, a query editor window titled "Untitled query" contains the following SQL code:

```

1 SELECT
2   | DISTINCT drive_wheels
3 FROM
4   | `my-project1-433108.cars.car_info`

```

The "Query results" section shows the output of the query:

Row	drive_wheels
1	rwd
2	fwd
3	4wd

Below the table, there are buttons for "SAVE RESULTS" and "EXPLORE DATA". The bottom right corner of the interface shows a refresh button.

And now only three unique values are in this column! Which means our data is clean, consistent, and ready for analysis!

Results:

The data cleaning process successfully identified and corrected several issues within the dataset:

- Fuel Type Validation: Confirmed that the `fuel_type` column contained only expected values (gas and diesel), ensuring reliability in fuel-based analysis.
- Numerical Integrity: Verified the `length` and `compression_ratio` columns, correcting outliers and ensuring that all values fell within the expected ranges, thus preserving the accuracy of car feature metrics.
- Missing Values: Addressed missing data in the `num_of_doors` column by consulting with the sales manager, resulting in accurate and complete records.
- Data Consistency: Rectified inconsistencies in the `drive_wheels` column by removing extraneous spaces, which standardized the entries and improved query performance.

These corrections led to a clean, consistent dataset, ready for in-depth analysis to identify the top 10 most popular car models based on historical sales data.

Through a thorough data cleaning process, we successfully prepared the dataset for analysis, ensuring it was free of errors, missing values, and inconsistencies. This clean dataset enabled accurate identification of the top 10 most popular car models based on historical sales data. These insights are vital for investors, as they provide a clear understanding of customer preferences,

allowing the dealership to optimize inventory and maximize profitability. By achieving this, the project meets the investors' goal of identifying popular car models to guide their stocking decisions.

Conclusion:

By implementing best practices in SQL data cleaning, we transformed a raw, error-prone dataset into a reliable resource for business analysis. This project not only improved data quality but also enhanced query performance and laid a strong foundation for accurate insights. The cleaned data empowers the dealership to make data-driven decisions on inventory stocking, thereby maximizing profitability and meeting customer demands.