

✔ **Congratulations! You passed!**

Grade received **100%** To pass 80% or higher

[Go to next item](#)

Hands-On Activity: Clean data using SQL

Total points 2

1.



1 / 1 point

Activity overview

In previous lessons, you learned about the importance of being able to clean your data where it lives. When it comes to data stored in databases, that means using SQL queries. In this activity, you will create a custom dataset and table, import a CSV file, and use SQL queries to clean automobile data.

In this scenario, you are a data analyst working with a used car dealership startup venture. The investors want you to find out which cars are most popular with customers so they can make sure to stock accordingly.

By the time you complete this activity, you will be able to clean data using SQL. This will enable you to process and analyze data in databases, which is a common task for data analysts.



What you will need

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` file to download it. Or you may download the CSV file directly from the attachments below.

Link to data: [automobile_data](#)

OR

Download download data:



automobile_data
CSV File

[Download file](#) ↓



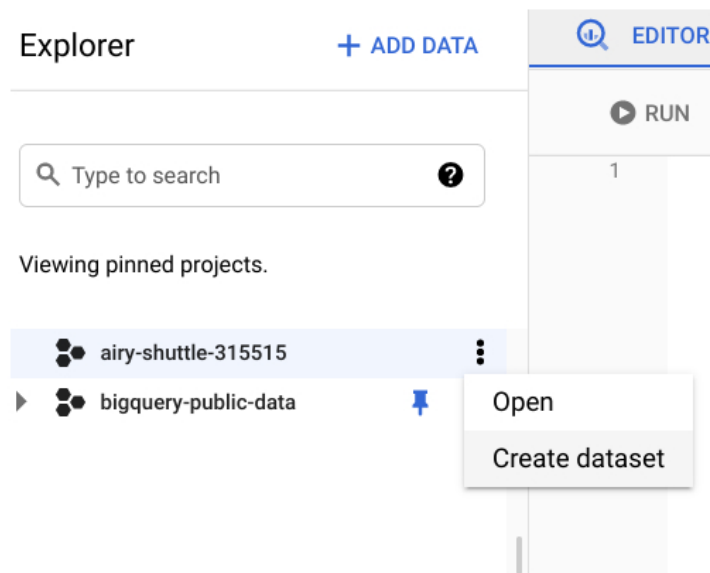
Upload your data

Similarly to a previous BigQuery activity, you will need to create a dataset and a custom table to house your data. Then, you'll be able to use SQL queries to explore and clean it. Once you've downloaded the `automobile_data` file, you can create your dataset.

Step 1: Create a dataset

Go to the **Explore** page in your workspace and click the three dots next to your pinned project to open the menu. From

Go to the **explorer** pane in your workspace and click the three dots next to your pinned project to open the 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 leave the Data location as Default. Then **click CREATE DATASET**.

Create dataset

Dataset ID *

cars

Letters, numbers, and underscores allowed

Data location

Default

Default table expiration

☐ Enable table expiration

Default maximum table age

Days

Encryption

☒ Google-managed encryption key

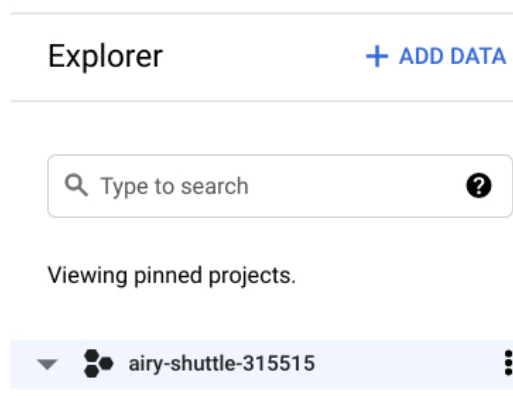
No configuration required

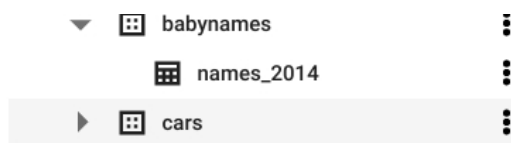
☐ Customer-managed encryption key (CMEK)

Manage via Google Cloud Key Management Service

CREATE DATASET CANCEL

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

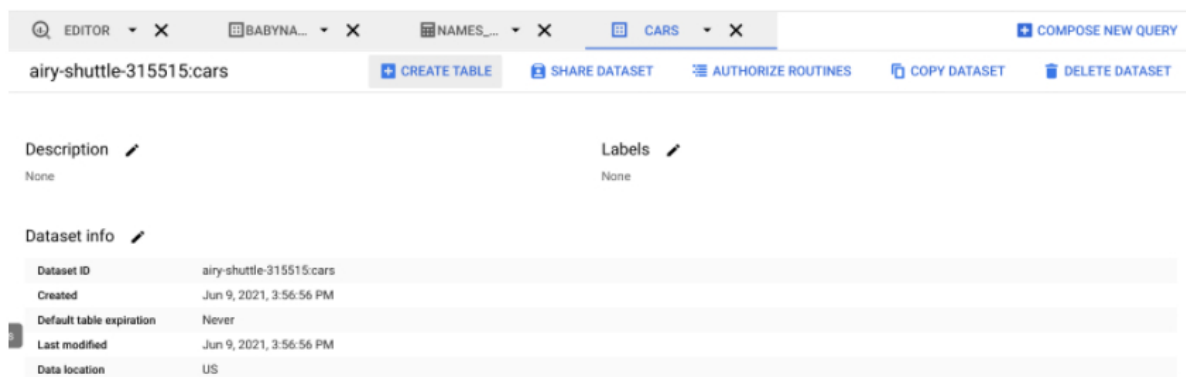




Step 2: Create table

After you open your newly created dataset, you will be able to add a custom table for your data.

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



Under Source, upload the automobile_data CSV. Under Destination, make sure you are uploading into your cars dataset and **name your table `car_info`.** You can **set the schema to Auto-detect**. Then, **click Create table**.

Create table

Source

Create table from: Select file: Browse File format:

Destination

☒ Search for a project ☐ Enter a project name

Project name: Dataset name: Table type:

Table name:

Schema

Auto detect

☒ Schema and input parameters

Partition and cluster settings

Partitioning:

Clustering order (optional):

After creating your table, it will appear in your Explorer pane. You can **click on the table to explore the schema and preview your data**. Once you have gotten familiar with your data, you can start querying it.

Cleaning your data

Your 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 your analysis, you'll need to make sure your data is clean. If you



analyze dirty data, you could end up presenting the wrong list of cars to the investors. That may cause them to lose money on their car inventory investment.

Step 1: Inspect the fuel_type column

The first thing you want to do is inspect the data in your table so you can find out if there is any specific cleaning that needs to be done. According to the [data's description](#), 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**:

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

This returns the following results:

Query results		 SAVE RESULTS	 EXPLORE DATA ▾
Query complete (0.6 sec elapsed, 1 KB processed)			
Job information		<u>Results</u>	JSON Execution details
Row	fuel_type		
1	gas		
2	diesel		

This confirms that the fuel_type column doesn't have any unexpected values.

Step 2: Inspect the length column

Next, you will inspect a column with numerical data. The length column should contain numeric measurements of the cars. So you will check that the minimum and maximum lengths in the dataset align with the [data description](#), which states that the lengths in this column should range from 141.1 to 208.1. **Run this query to confirm**

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

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

Row	min_length	max_length
1	141.1	208.1

Step 3: Fill in missing data

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

You can **check to see 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 any rows with missing data for the num_of_doors column and return them in your results table. You should get two results, one Mazda and one Dodge:

Row	make	fuel_type	num_of_doors	body_style
1	dodge	gas	<i>null</i>	sedan
2	mazda	diesel	<i>null</i>	sedan

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

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

You should get a message telling you that three rows were modified in this table. To make sure, you can **run the previous query again:**

```
SELECT
*
FROM
cars.car_info

WHERE

num_of_doors IS NULL;
```

Now, you only have one row with a NULL value for num_of_doors. **Repeat this process to replace the null value for the Mazda.**

If you are using the BigQuery Sandbox, you can skip these UPDATE queries; they will not affect your ability to complete this activity.

Step 4: Identify potential errors

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

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

After running this, you notice that there are one too many rows. **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 misspelling for all rows, you can run this query if you have the BigQuery free trial:

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

You will get a message alerting you that one row was modified after running this statement. To **check that it worked, you can run the previous query again:**

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

Next, you 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, you can use MIN and MAX to check if that's correct:

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

Notice that **this returns a maximum of 70**. But you 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.

```
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 aligns with the data description. So you'll want to correct the 70 value. You check with the sales manager again, who says that this row was made in error and should be removed. Before you delete anything, you should check to see how many rows contain this erroneous value as a precaution so that you don't end up deleting 50% of your data. If there are too many (for instance, 20% of your rows have the incorrect 70 value), then you 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 below 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 there is only one row with the erroneous /0 value. So you can **delete that row using this query**:

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

If you are using the BigQuery sandbox, you can replace DELETE with SELECT to see which row would be deleted.

Step 5: Ensure consistency

Finally, you want to check your 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**:

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

It appears that 4wd appears twice in results. However, because you 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.

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

To check if this is the case, you can **use a LENGTH statement** to determine the length of how long 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, you can **use the TRIM function to remove all extra spaces in the drive_wheels column if you are using the BigQuery free trial**:

```
UPDATE

  cars.car_info

SET

  drive_wheels = TRIM(drive_wheels)

WHERE TRUE;
```

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

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

And now there should only be three unique values in this column! Which means your data is clean, consistent, and ready for analysis!

Confirmation and reflection

What is the maximum value in the price column of the car_info table?

- ☐ 5,1180
- ☒ 45,400
- ☐ 16,430
- ☐ 12,978

✓ **Correct**

To ensure that the values in the price column fell within the expected range, you used the MIN and MAX functions to determine that the maximum price was 45, 400. Knowing this, you were able to clean this column and prepare for analysis. Going forward, you will continue to check columns with numeric data in BigQuery to make sure your data is clean. This will help you quickly identify issues with your data that might cause errors during analysis.

2. In the text box below, write 2-3 sentences (40-60 words) in response to each of the following questions:

1 / 1 point

- Why is cleaning data before your analysis important?
- Which of these cleaning techniques do you think will be most useful for you in the future?

Why is cleaning data before your analysis important?

It is important in order to have reliable data to start with the analysis.

Which of these cleaning techniques do you think will be most useful for you in the future?

I really liked using SQL. All this knowledge will be very useful in the future.

✓ **Correct**

Congratulations on completing this hands-on activity! In this activity you checked your data for errors and fixed any inconsistencies. A good response would include that cleaning data is an important step of the analysis process that will save you time and help ensure accuracy in the future.

Cleaning data where it lives is incredibly important for analysts. For instance, you were able to use SQL to complete multiple cleaning tasks, which allows you to clean data stored in databases. In upcoming activities, you will use your cleaning skills to prepare for analysis!