Data Cleaning With SQL

**Introduction**

You already know that clean data is an essential part of the data analysis process. Now you are going to learn about cleaning data from a database to prepare it for analysis using SQL. You are going to take a "dirty" dataset and clean it up using basic SQL commands right here in this Qwiklab.

Data cleaning corrects or removes incorrect, missing, and faulty data. This is very important. Dirty data can lead to errors and mistakes in your analysis, or worse, to the wrong conclusions. The better your data, the better your results.

Let's check out a real-world situation.

You are working with a used car dealership start-up venture. The investors want to know which cars customers are mostly to purchase so that they know what to keep in stock. From an external source, you find a data set that contains historical sales data on car prices and their related features: [automobile\_data](https://drive.google.com/u/0/uc?id=1cJtuw-6mxZk7BNkcsLYEvfjW0l_PdKxA&export=download). You think these features can be used to give investors a list of the top 10 most popular cars and trims, and help determine which cars should be in the inventory.

**Before doing any analysis or making any predictions, this dataset must be cleaned.** Analyzing bad data could cause investors to reach the wrong conclusions and, ultimately, lose money on their car inventory investment.

What you'll do

For this lab, you will use BigQuery, which is a fully-managed, serverless database management system (DBMS) that supports querying using SQL. Other open-source options include MySQL Community Edition, MongoDB, and PostgreSQL-- just keep in mind that the visual design tool for each DBMS might be different. However, the queries you will perform in this Qwiklab will be similar regardless of the DBMS used.

You will use SQL commands to perform database actions like:

* Create a new SQL table using CREATE TABLE.
* Import a CSV file into a database table
* Use the SELECT statement to view database records and SELECT DISTINCT to view unique database records
* Use the ORDER BY keyword to sort data in different ways
* Use the WHERE clause to filter data in different ways
* Use the UPDATE statement to modify records in a table
* Use the DELETE statement to remove records in a table
* Use the MIN(), MAX(), AVG(), LENGTH() and TRIM() functions to clean data in a table

**You will have 90 minutes to complete this lab.**

Start your lab by signing in to the Console

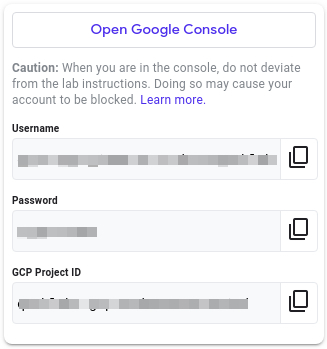
**Before you click the Start Lab button** read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab** , shows how long Google Cloud resources will be made available to you.

This Qwiklab hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

1. Click the **Start Lab** button.

Start Lab

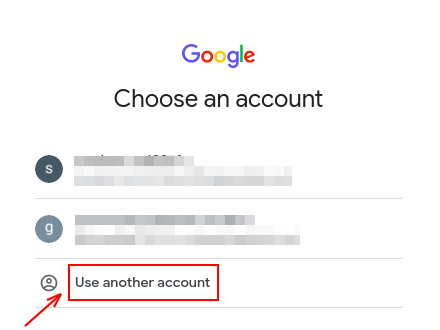
On the left is a panel populated with the temporary credentials that you'll need to use for this lab.



1. Copy the username, then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Choose an account** page.

***Tip:*** Open the tabs in separate windows, side by side.

**Note:**Using a new Incognito window (Chrome) or another browser for the Qwiklabs session is recommended. Alternatively, you can log out of all other Google / Gmail accounts before beginning the labs.

1. On the **Choose an account** page, click **Use another account**. 
2. The **Sign in** page opens. Paste the username that you copied from the **Connection Details** panel. Then copy and paste the password.

***Important:*** You must use the credentials from the **Connection Details** panel. Please do **not** use your Qwiklabs credentials. If you have your own GCP account, do **not** use it for this lab in order to avoid incurring charges.

1. Click through the subsequent pages:
2. Accept the terms and conditions.
3. Do **not** add recovery options or two-factor authentication, since this is a temporary account.
4. Do **not** sign up for free trials.

After a few moments, the GCP console opens in this tab.

What you need

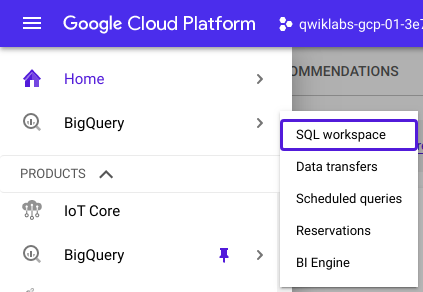
To complete this lab, you need:

* Access to a standard internet browser (Chrome browser recommended).
* Time to complete the lab.

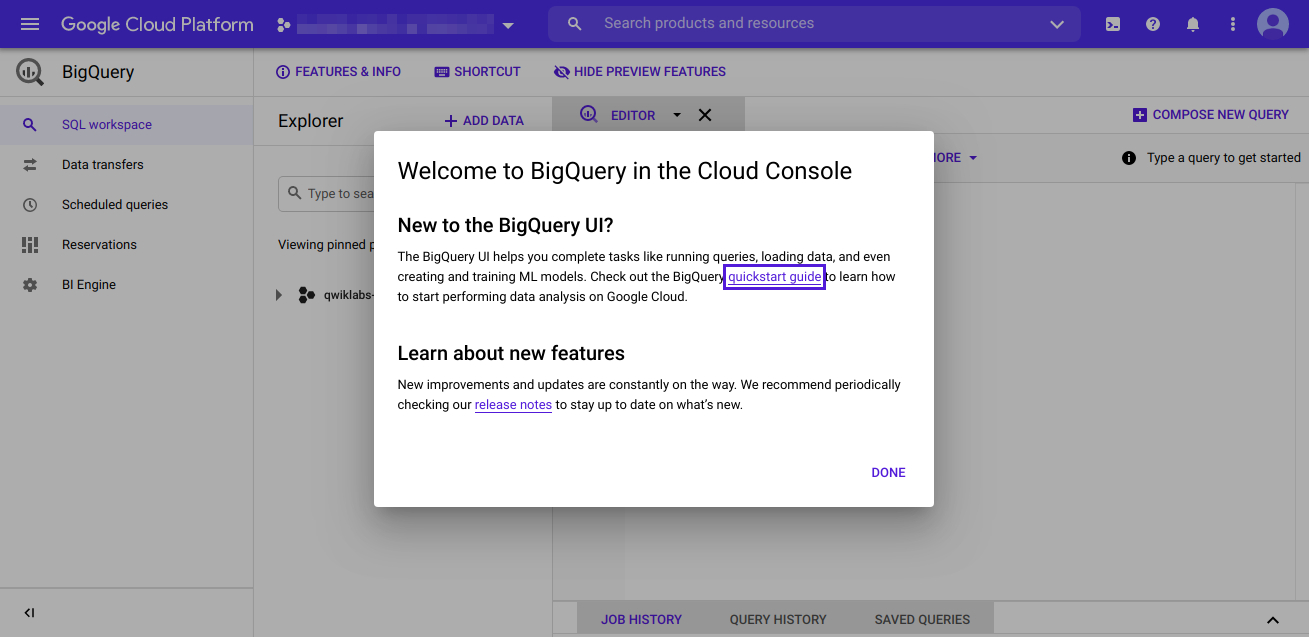
**Note:**If you already have your own personal Google Cloud account or project, do not use it for this lab.

Open BigQuery Console

In the Google Cloud Console, select **Navigation menu** > **BigQuery**:

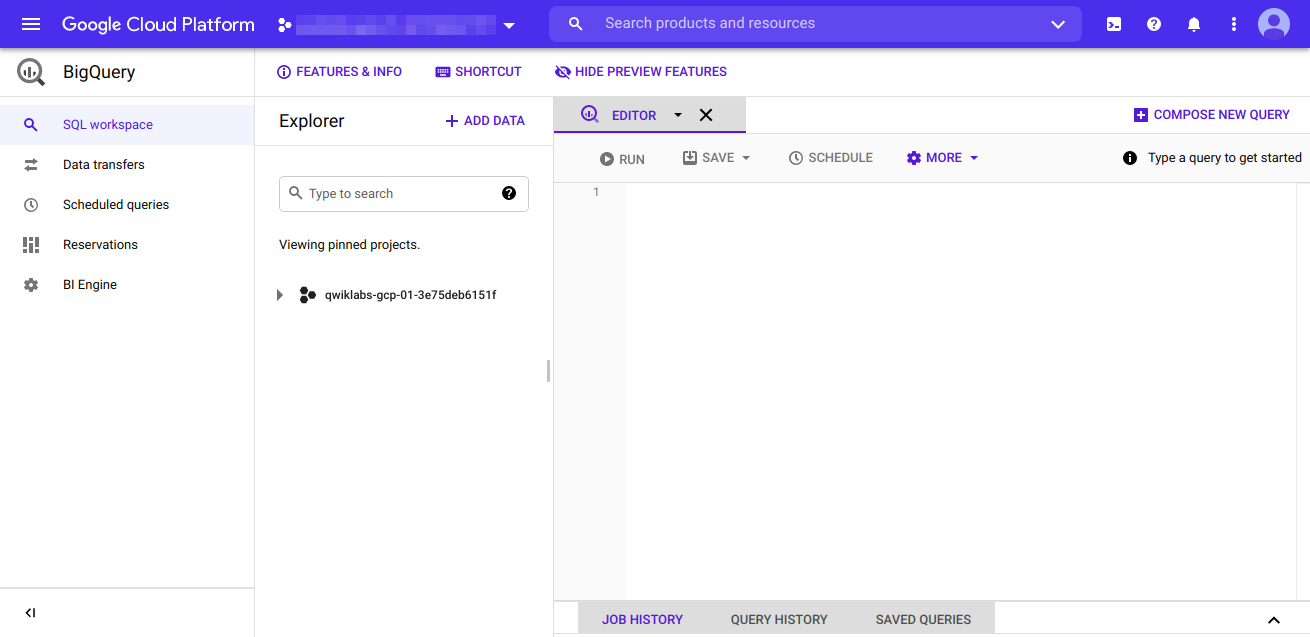


The **Welcome to BigQuery in the Cloud Console** message box opens. This message box provides a link to the quickstart guide and the release notes.



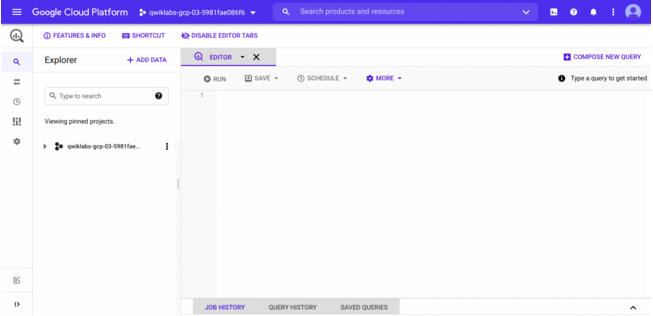
Click **Done**.

The BigQuery console opens.

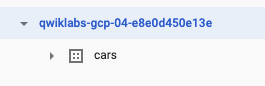


Creating the database, creating the table, and importing the data

To get started, you will need to create the *cars* database where you will store your table. First select your qwiklabs project and click on the View actions icon (three dots) then select **CREATE DATASET**. Then on the popup menu, make the following entries and selections.



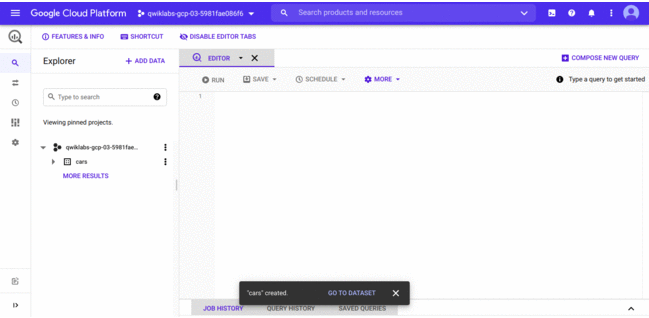
Then click **Create dataset**. You can verify that the cars database was created by selecting the project name in the dropdown menu on the left. The cars database should appear underneath the project.



Next, you need to create a table within the cars database to store your car data.

Since you will be importing data from a CSV file, it will be easier to create this table using the web interface instead of a query. This way, BigQuery will create the table, automatically define the appropriate data types, and import that data all in one step. Other database management systems may require you to run a query to do the import, however.

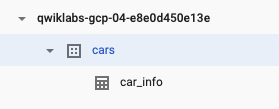
On the left menu pane, click on the **View actions** icon next to the **cars** dataset and select **Open**, then click **CREATE TABLE** from the right side. On the next popup menu, make the following entries and selections.



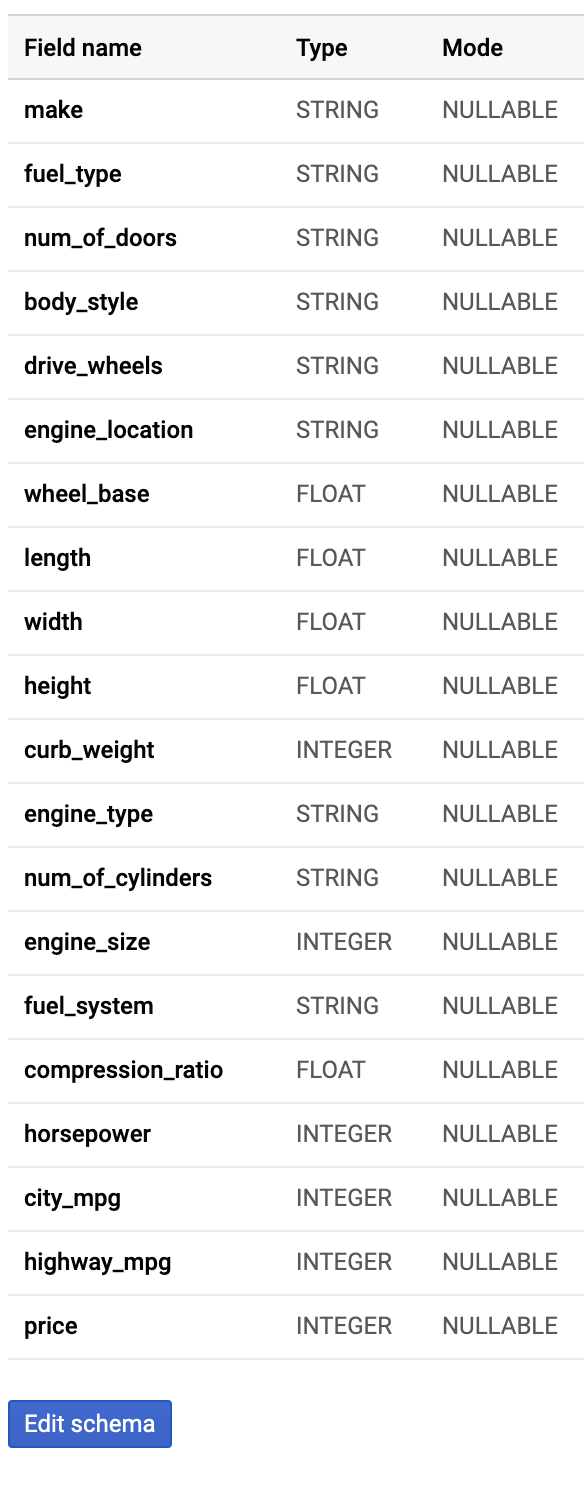
When you click the Browse button shown above, you will be prompted to navigate to your downloaded file. If you didn't rename the file it should be *automobile\_data.csv*. To make the advanced option selections shown above, click the **Advanced options** dropdown.

Next click **Create table**.

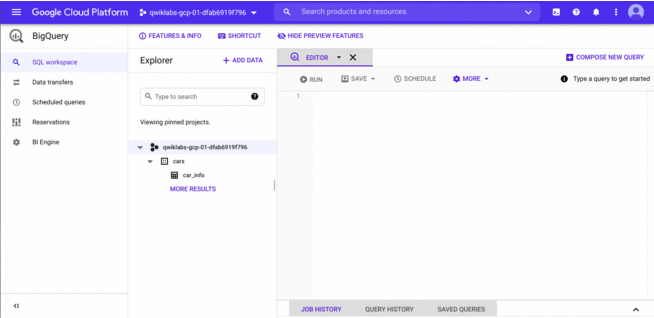
Now the dataset has been imported into the car\_info table within the cars database. You can verify this by selecting the cars database in the left menu. The car\_info table should appear underneath the dropdown.



To view the data column names and data types for the data imported into car\_info, select the car\_info table on this menu. You should get the following schema:



The SELECT statement is used to select data from a table. The data returned is stored in a result table, called the result-set. To view the actual data imported into car\_info, make the following query containing the SELECT statement:



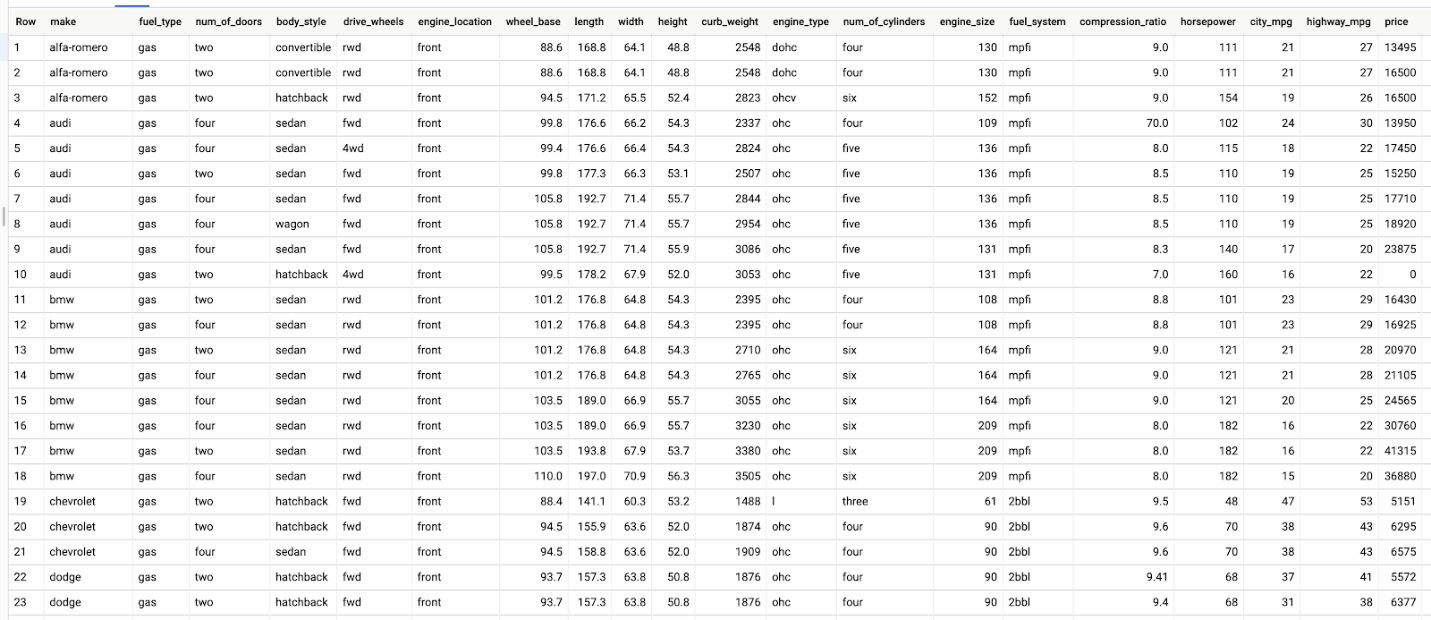
SELECT

\*

FROM

cars.car\_info;

After you enter the query in the Query Editor, click on the run button to run your query. You should get something like this:



Notice that BigQuery requires you to specify the database name and the table name you are querying separated by a period like this: cars.cars\_info.

In SQL, the COUNT() function returns the number of rows that match specific characteristics. Verify that your table has 203 rows by executing the following query containing the COUNT() function:

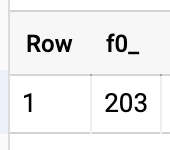
SELECT

COUNT (\*)

FROM

cars.car\_info;

After you enter the query in the Query Editor, click on the run button to run your query. You should get the following table:



**Inspecting the data**

To clean the data, you'll need to know what data values and types are in each column. This is called inspecting the data. A good way to inspect string data type columns is to check the unique values they contain. This will make it easier to find out if there's data that needs to be cleaned.

According to the [data description](https://archive.ics.uci.edu/ml/datasets/Automobile), fuel\_type should contain only two unique string values: diesel and gas. For this reason, this column should be inspected by checking its unique values. In SQL, you can view a column's values by including DISTINCT in the SELECT statement. Make the following query to view the fuel\_type column's unique values:

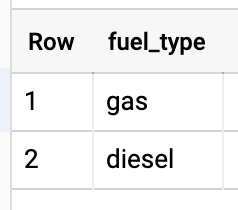
SELECT

DISTINCT(fuel\_type)

FROM

cars.car\_info;

After you enter the query in the Query Editor, click on the run button to run your query. You should get the following:



Since these values agree with the values in the data description, they verify that the data in the fuel\_type column is already clean.

A good way to inspect the data in a column containing numeric values is to sort it. The ORDER BY keyword is used in SQL to sort the result-set in ascending or descending order. The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

The length column contains numeric data, so you can inspect it by sorting in ascending order. To do this, make the following query containing the ORDER BY keyword:

SELECT

length

FROM

cars.car\_info

ORDER BY

length;

Now you can scroll through the data to double-check it. But if a dataset contains many rows of data, it may also be a good idea to inspect the minimum and maximum values of a column rather than scrolling down the returned values to locate them. In SQL, the MIN() function identifies the smallest value of the selected column, and the MAX() function identifies the largest value of the selected column.

According to the [data description](https://archive.ics.uci.edu/ml/datasets/Automobile), the length column values should range from 141.1 to 208.1. Make the following query containing the MIN() and MAX() functions to inspect the minimum and maximum values of this column:

SELECT

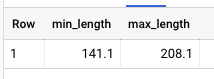
MIN(length) AS min\_length,

MAX(length) AS max\_length

FROM

cars.car\_info;

You should get the following:

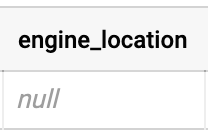


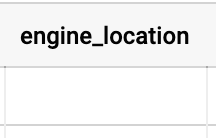
In SQL, the AS command is used to rename a column or table with an alias. In the above query, the columns were renamed as min\_length and max\_length to make it easier to tell which value in the result-set is the minimum and which is the maximum. These results agree with the value range in the data description, making the length column clean.

Filling in missing data

Missing values in data can distort analysis results. One way to inspect a column for missing values in SQL is to use the IS NULL condition.

In SQL, NULL is the term used to represent a missing value. A NULL value in a table means that there is a value in that column that appears to be absent. It is very important to understand that a NULL value is different from a zero value or a blank value. If a value is blank, it will appear in a column as empty. The following are examples of how a NULL and a blank value will appear in a column.





To inspect the num\_of\_doors column for NULL values, you will need to filter this column using the IS NULL condition. In SQL, the WHERE clause is used to filter records. Make the following query containing the WHERE clause and the IS NULL condition:

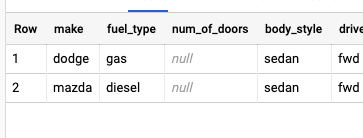
SELECT

\*

FROM

cars.car\_info WHERE num\_of\_doors IS NULL;

You should get the following:



It appears that the number of doors is only missing for one Dodge and one Mazda with sedan body styles. Also, the fuel type equals gas for the Dodge and diesel for the Mazda.

Make the following query to pull all Dodges with matching attributes:

SELECT

make,

fuel\_type,

num\_of\_doors,

body\_style

FROM

cars.car\_info

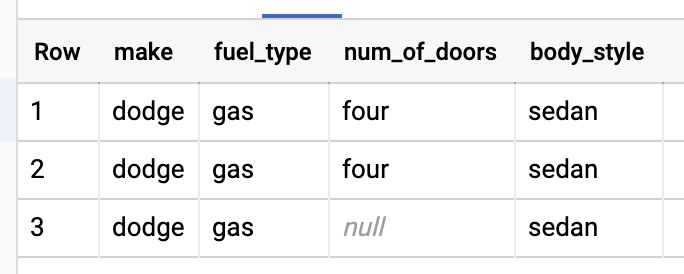
WHERE

make = "dodge"

AND fuel\_type = "gas"

AND body\_style = "sedan";

The results should be like this:



For all records that have the query attributes fuel\_type = gas and body\_style = sedan, the non-missing value is four. Therefore, you can safely conclude that the missing value here should also be four. You'll need to modify this record by replacing the *null* value in the num\_of\_doors column with four.

In SQL, the UPDATE statement is used to modify existing records in a table. To make the replacement, make the following query using the UPDATE statement:

UPDATE

cars.car\_info

SET

num\_of\_doors = "four"

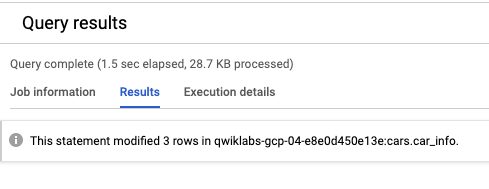
WHERE

make = "dodge"

AND fuel\_type = "gas"

AND body\_style = "sedan";

A message stating that 3 rows were modified should appear.



To verify that the missing value was filled in correctly, make the following query again:

SELECT

make,

fuel\_type,

num\_of\_doors,

body\_style

FROM

cars.car\_info

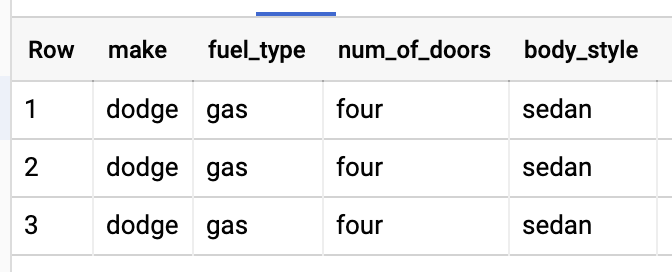
WHERE

make = "dodge"

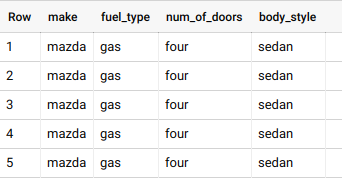
AND fuel\_type = "gas"

AND body\_style = "sedan";content\_copy

The results should appear like this:



Now the value four is filled in for the *null* value in the num\_of\_doors column for the Dodge car. Perform a similar process to fill in the blank in the num\_of\_doors column for the Mazda car.



**Identifying potential errors in the data**

Erroneous data can also distort results. You can sometimes spot erroneous data by inspecting a column's unique values. In SQL, the SELECT DISTINCT statement is used to return only distinct (or unique) values. To check for errors in the num\_of\_cylinders column, make the following query containing the SELECT DISTINCT statement:

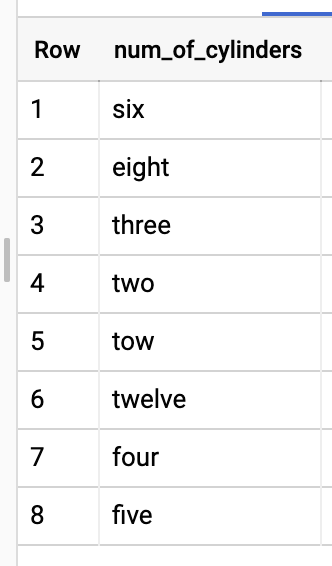
SELECT

DISTINCT(num\_of\_cylinders)

FROM

cars.car\_info;

You should get the following results:



Check out row 5: there are misspelled values in this column. You can safely conclude that "tow" here should probably be "two". Therefore, to fix the misspelling, make this next query:

UPDATE

cars.car\_info

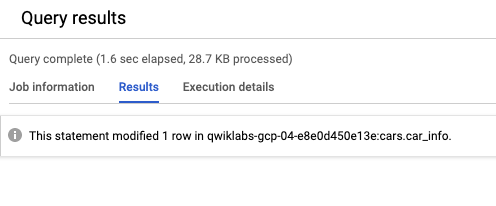
SET

num\_of\_cylinders = "two"

WHERE

num\_of\_cylinders = "tow";

A message stating that 1 row was modified should appear.



To verify that the erroneous values were filled in correctly, make the following query again:

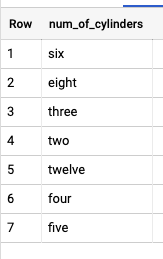
SELECT

DISTINCT(num\_of\_cylinders)

FROM

cars.car\_info;content\_copy

The results should be like this:



The num\_of\_cylinders column is clean now.

Make the following query to inspect the compression\_ratio column:

SELECT

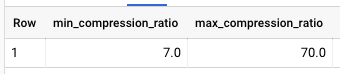
MIN(compression\_ratio) AS min\_compression\_ratio,

MAX(compression\_ratio) AS max\_compression\_ratio

FROM

cars.car\_info;

The results should be like this:



According to the [data description](https://archive.ics.uci.edu/ml/datasets/Automobile), the compression\_ratio column values should range from 7 to 23. That means that the maximum value of 70 is an error.

Let's make the following query to determine if there are any other values in this column that are out of range:

SELECT

compression\_ratio

FROM

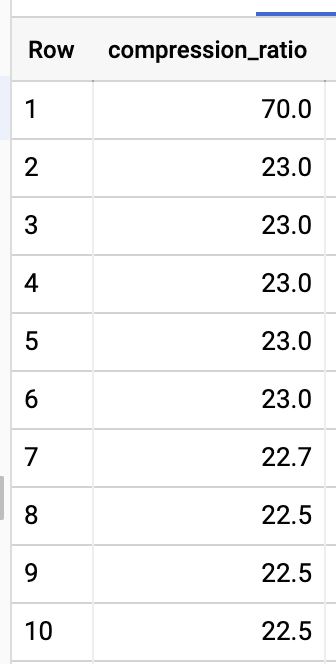
cars.car\_info

ORDER BY

compression\_ratio DESC;

The results should be like this:

There will be 203 results for this query.



It appears that all the other values are within range.

Make the following query to get more insight:

SELECT

\*

FROM

cars.car\_info

WHERE

compression\_ratio = 70;

The result should be like this:



There is one result returned. This means that only one row contains this error. One way to handle erroneous values is by deleting the rows in which they are contained. Keep in mind that deleting data rows can also cause distorted results. If you are deleting no more than 20% of the data rows, then your results should be fine. Because there is only one row you need to delete, it should be fine.

In SQL, the DELETE statement is used to delete existing records in a table. Make the following query containing the DELETE statement to delete the rows containing the identified error:

DELETE

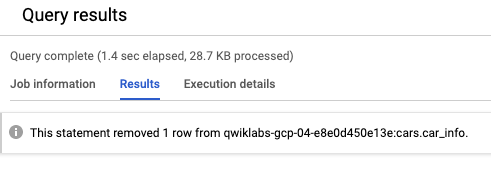
FROM

cars.car\_info

WHERE

compression\_ratio = 70;

A message stating that 1 row was removed should appear.



To verify that the rows containing the erroneous values were deleted correctly, make the following query again:

SELECT

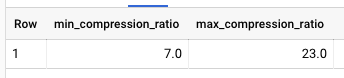
MIN(compression\_ratio) AS min\_compression\_ratio,

MAX(compression\_ratio) AS max\_compression\_ratio

FROM

cars.car\_info;content\_copy

You should get these results:



Since the results are within range, the compression\_ratio column should be clean.

Make the following query to inspect the price column:

SELECT

MIN(price) AS min\_price,

MAX(price) AS max\_price

FROM

cars.car\_info;

Here are the results:



According to the [data description](https://archive.ics.uci.edu/ml/datasets/Automobile), prices should not go below 5,118. Unfortunately, the results don't match that. Sort the data to get more insight by making this next query:

SELECT

price

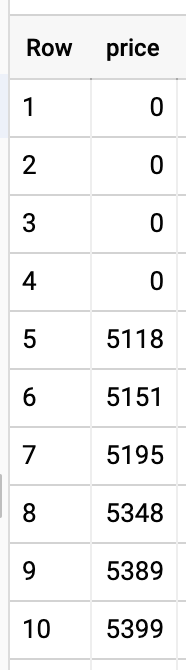
FROM

cars.car\_info

ORDER BY

price ASC;

The results should be this:



It appears that only the 0 values are out of range. Since there are only four rows that contain this error, you could delete them using the same logic as the compression\_ratio column; but let's use mean imputation instead.

Mean imputation is a method in which erroneous values in a column are replaced by the mean (or average) of the other values in that column. This method maintains the dataset size, but some important statistics like variance and standard deviation tend to be minimized.

In SQL, the AVG() function returns the average value of a numeric column. To find the average price, make the following query containing the AVG() function:

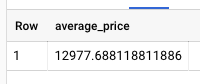
SELECT

AVG(price) AS average\_price

FROM

cars.car\_info;

You should get these results:



Next, replace the 0 values in the price column with the rounded average value by making the following query:

UPDATE

cars.car\_info

SET

price = 12978

WHERE

price = 0;

To verify that the rows containing the erroneous values were filled in correctly, make the following query again:

SELECT

MIN(price) AS min\_price,

MAX(price) AS max\_price

FROM

cars.car\_info;content\_copy

You should get these results:



Because the results are in range, the price column seems clean.

**Ensuring consistency in the data**

Data consistency means that there is consistency in the measurement of variables throughout the data tables. Discrepancies can create inaccurate, unreliable results. This leads to misinformed business decisions.

Data inconsistencies are often overlooked and can sometimes be tricky to spot. One way data inconsistencies can occur within a table is if values that are meant to be the same either are spelled differently or have different character lengths due to extra spaces. You can spot these types of inconsistencies by inspecting a column's unique values.

To check for data inconsistencies in the drive\_wheels column, make the following query containing the SELECT DISTINCT statement:

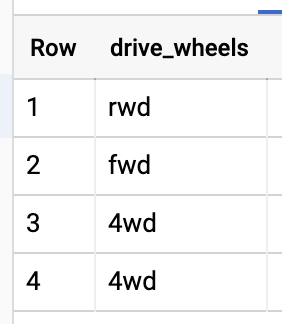
SELECT

DISTINCT drive\_wheels

FROM

cars.car\_info;

You should get these results:



These values appear to agree with the drive\_wheels values in the [data description](https://archive.ics.uci.edu/ml/datasets/Automobile). However, in this unique drive\_wheels listing, 4wd appears twice. This is happening because some 4wd values include extra spaces. To understand this, make the next query containing the LENGTH() function:

SELECT

drive\_wheels,

LENGTH(drive\_wheels) AS length\_drive\_wheels

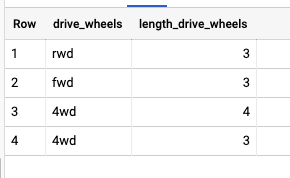
FROM

cars.car\_info

GROUP BY

drive\_wheels;content\_copy

You should get the following results:



In SQL, the LENGTH() function identifies the length of a string of data. Notice that in some instances, 4wd has four characters when there should only be three. To fix this, you need to remove the leading or trailing whitespace from this string. In SQL, the TRIM() function removes spaces from the start or end of a string.

Make the next query containing the TRIM() function:

UPDATE

cars.car\_info

SET

drive\_wheels = TRIM(drive\_wheels)

WHERE

LENGTH(drive\_wheels) > 3;content\_copy

To verify whether the leading or trailing space was removed, make the following query again:

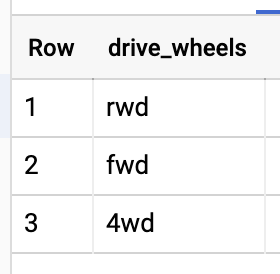
SELECT

DISTINCT drive\_wheels

FROM

cars.car\_info;content\_copy

You should get these results:



Now each unique value is listed only once and therefore, the drive\_wheels column is clean.

**Congratulations!**

You've successfully cleaned this dataset using SQL. Working with clean data will ensure that investors make good decisions about their stock. Mastering the data cleaning skills presented in this Qwiklab will help you as a data analyst.

You can now close this window, manually end the lab, and continue to the next item in the course.

**End your lab**

When you have completed your lab, click **End Lab**. Qwiklabs removes the resources you’ve used and cleans the account for you.

You will be given an opportunity to rate the lab experience. Select the applicable number of stars, type a comment, and then click **Submit**.

The number of stars indicates the following:

* 1 star = Very dissatisfied
* 2 stars = Dissatisfied
* 3 stars = Neutral
* 4 stars = Satisfied
* 5 stars = Very satisfied

You can close the dialog box if you don't want to provide feedback.

For feedback, suggestions, or corrections, ple