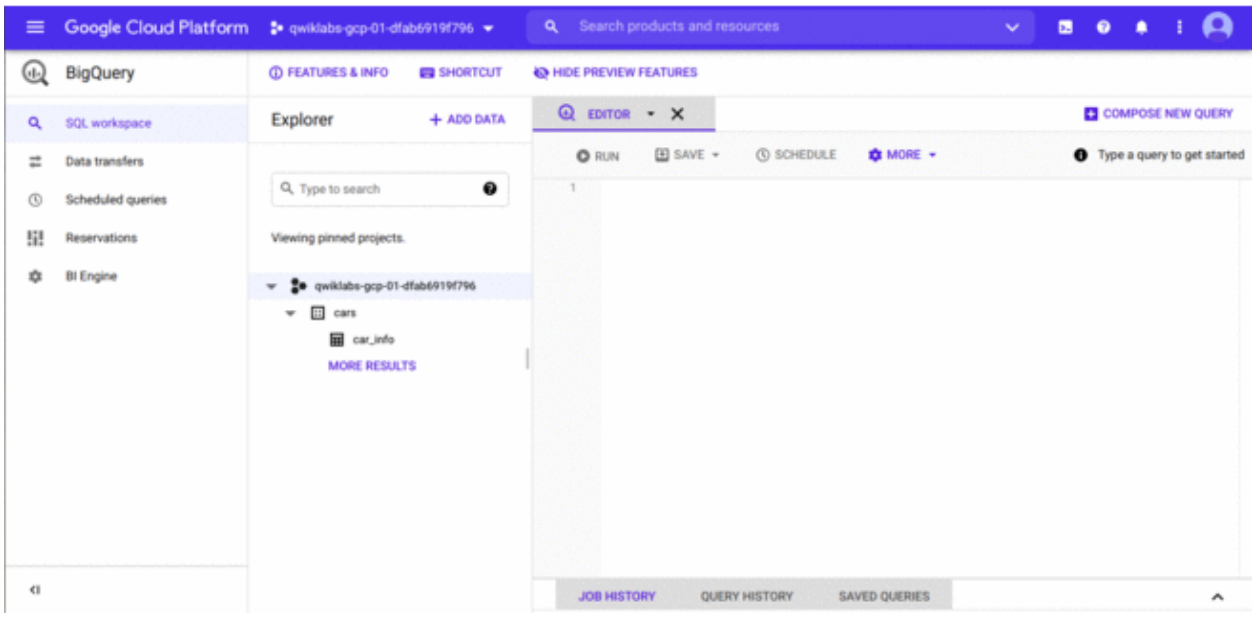
Data Cleaning Lab - Cars table

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;
```

content_copy

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

Row	make	fuel_type	num_of_doors	body_style	drive_wheels	engine_location	wheel_base	length	width	height	curb_weight	engine_type	num_of_cylinders	engine_size	fuel_system	compression_ratio	horsepower	city_mpg	highway_mpg	price
1	alfa-romero	gas	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	dohc	four	130	mpfi	9.0	111	21	27	13495
2	alfa-romero	gas	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	dohc	four	130	mpfi	9.0	111	21	27	16500
3	alfa-romero	gas	two	hatchback	rwd	front	94.5	171.2	65.5	52.4	2823	ohcv	six	152	mpfi	9.0	154	19	26	16500
4	audi	gas	four	sedan	fwd	front	99.8	176.6	66.2	54.3	2337	ohc	four	109	mpfi	70.0	102	24	30	13950
5	audi	gas	four	sedan	4wd	front	99.4	176.6	66.4	54.3	2824	ohc	five	136	mpfi	8.0	115	18	22	17450
6	audi	gas	two	sedan	fwd	front	99.8	177.3	66.3	53.1	2507	ohc	five	136	mpfi	8.5	110	19	25	15250
7	audi	gas	four	sedan	fwd	front	105.8	192.7	71.4	55.7	2844	ohc	five	136	mpfi	8.5	110	19	25	17710
8	audi	gas	four	wagon	fwd	front	105.8	192.7	71.4	55.7	2954	ohc	five	136	mpfi	8.5	110	19	25	18920
9	audi	gas	four	sedan	fwd	front	105.8	192.7	71.4	55.9	3086	ohc	five	131	mpfi	8.3	140	17	20	23875
10	audi	gas	two	hatchback	4wd	front	99.5	178.2	67.9	52.0	3053	ohc	five	131	mpfi	7.0	160	16	22	0
11	bmw	gas	two	sedan	rwd	front	101.2	176.8	64.8	54.3	2395	ohc	four	108	mpfi	8.8	101	23	29	16430
12	bmw	gas	four	sedan	rwd	front	101.2	176.8	64.8	54.3	2395	ohc	four	108	mpfi	8.8	101	23	29	16925
13	bmw	gas	two	sedan	rwd	front	101.2	176.8	64.8	54.3	2710	ohc	six	164	mpfi	9.0	121	21	28	20970
14	bmw	gas	four	sedan	rwd	front	101.2	176.8	64.8	54.3	2765	ohc	six	164	mpfi	9.0	121	21	28	21105
15	bmw	gas	four	sedan	rwd	front	103.5	189.0	66.9	55.7	3055	ohc	six	164	mpfi	9.0	121	20	25	24565
16	bmw	gas	four	sedan	rwd	front	103.5	189.0	66.9	55.7	3230	ohc	six	209	mpfi	8.0	182	16	22	30760
17	bmw	gas	two	sedan	rwd	front	103.5	193.8	67.9	53.7	3380	ohc	six	209	mpfi	8.0	182	16	22	41315
18	bmw	gas	four	sedan	rwd	front	110.0	197.0	70.9	56.3	3505	ohc	six	209	mpfi	8.0	182	15	20	36880
19	chevrolet	gas	two	hatchback	fwd	front	88.4	141.1	60.3	53.2	1488	l	three	61	2bbl	9.5	48	47	53	5151
20	chevrolet	gas	two	hatchback	fwd	front	94.5	155.9	63.6	52.0	1874	ohc	four	90	2bbl	9.6	70	38	43	6295
21	chevrolet	gas	four	sedan	fwd	front	94.5	158.8	63.6	52.0	1909	ohc	four	90	2bbl	9.6	70	38	43	6575
22	dodge	gas	two	hatchback	fwd	front	93.7	157.3	63.8	50.8	1876	ohc	four	90	2bbl	9.41	68	37	41	5572
23	dodge	gas	two	hatchback	fwd	front	93.7	157.3	63.8	50.8	1876	ohc	four	90	2bbl	9.4	68	31	38	6377

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;
```

content_copy

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

Row	f0_
1	203

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, `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;
```

content_copy

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

Row	fuel_type
1	gas
2	diesel

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;
```

content_copy

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, 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;
```

content_copy

You should get the following:

Row	min_length	max_length
1	141.1	208.1

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.

engine_location
null

engine_location

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;
```

content_copy

You should get the following:

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

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";
```

content_copy

The results should be like this:

Row	make	fuel_type	num_of_doors	body_style
1	dodge	gas	four	sedan
2	dodge	gas	four	sedan
3	dodge	gas	*null*	sedan

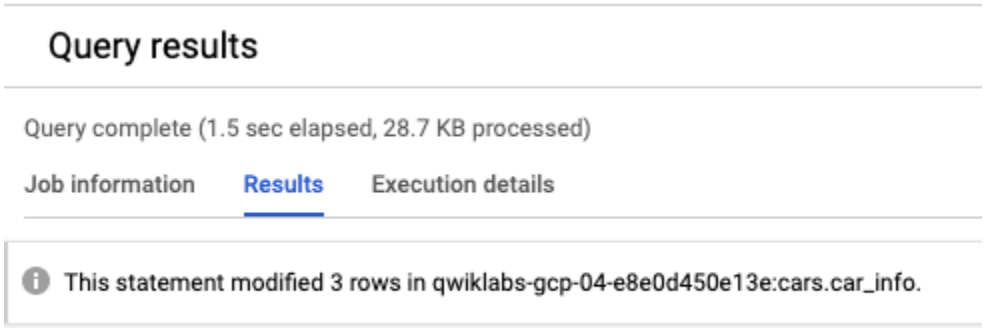
For all records that have the query attributes `fuel_type = gas` and `body_style = sedan`, the non-missing value is four. Therefore, you can safely conclude that the missing value here should also be four. You'll need to modify this record by replacing the *null* value in the `num_of_doors` column with 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";
```

content_copy

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:

Row	make	fuel_type	num_of_doors	body_style
1	dodge	gas	four	sedan
2	dodge	gas	four	sedan
3	dodge	gas	four	sedan

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.

Row	make	fuel_type	num_of_doors	body_style
1	mazda	gas	four	sedan
2	mazda	gas	four	sedan
3	mazda	gas	four	sedan
4	mazda	gas	four	sedan
5	mazda	gas	four	sedan

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;
```

content_copy

You should get the following results:

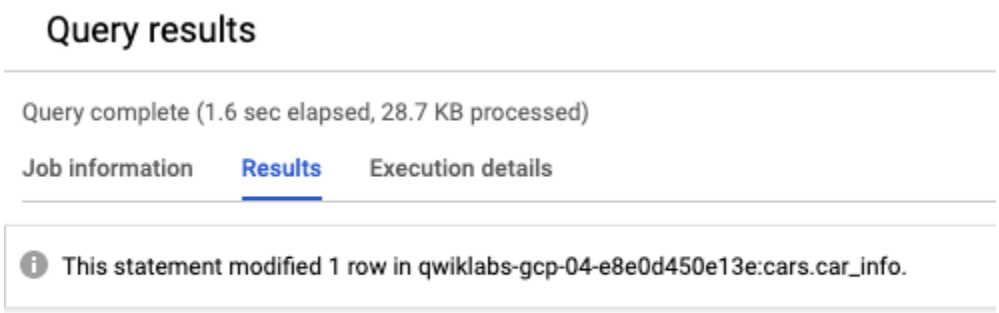
Row	num_of_cylinders
1	six
2	eight
3	three
4	two
5	tow
6	twelve
7	four
8	five

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";
```

content_copy

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:

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

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;
```

content_copy

The results should be like this:

Row	min_compression_ratio	max_compression_ratio
1	7.0	70.0

According to the data description, 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;
```

content_copy

The results should be like this:

There will be 203 results for this query.

Row	compression_ratio
1	70.0
2	23.0
3	23.0
4	23.0
5	23.0
6	23.0
7	22.7
8	22.5
9	22.5
10	22.5

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;
```

content_copy

The result should be like this:

Row	make	fuel_type	num_of_doors	body_style	drive_wheels	engine_location	wheel_base	length	width	height	curb_weight	engine_type	num_of_cylinders	engine_size	fuel_system	compression_ratio	horsepower	city_mpg	highway_mpg	price
1	audi	gas	four	sedan	fwd	front	99.8	176.6	66.2	54.3	2337	ohc	four	109	mpfi	70.0	102	24	30	13950

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;
```

content_copy

A message stating that 1 row was removed should appear.

Query results

Query complete (1.4 sec elapsed, 28.7 KB processed)

Job information | Results | Execution details

This statement removed 1 row from qwiklabs-gcp-04-e8e0d450e13e:cars.car_info.

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:

Row	min_compression_ratio	max_compression_ratio
1	7.0	23.0

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;
```

content_copy

Here are the results:

Row	min_price	max_price
1	0	45400

According to the data description, 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;
```

content_copy

The results should be this:

Row	price
1	0
2	0
3	0
4	0
5	5118
6	5151
7	5195
8	5348
9	5389
10	5399

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;
```

content_copy

You should get these results:

Row	average_price
1	12977.688118811886

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;
```

content_copy

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:

Row	min_price	max_price
1	5118	45400

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;
```

content_copy

You should get these results:

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

These values appear to agree with the `drive_wheels` values in the data description. 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:

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

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:

Row	drive_wheels
1	rwd
2	fwd
3	4wd

Now each unique value is listed only once and therefore, the `drive_wheels` column is clean.