BigQuery GIS

Summary	In this lab, you'll learn how to use Google Cloud Platform and BigQuery GIS to ingest and analyze large Geospatial datasets from common GIS formats, and inspect the results of your analyses.
URL	cloud-bqgis
Category	Cloud, Geo
Status	Final
Environment	web, kiosk, cloud, g4g2019

Overview

In a data warehouse like BigQuery, location information is very common. Many critical decisions revolve around location data. BigQuery GIS allows you to analyze geographic data in BigQuery. Geographic data is also known as geospatial data.

BigQuery GIS adds support for a `GEOGRAPHY` data type to standard SQL. The `GEOGRAPHY` data type represents a pointset on the Earth's surface. A pointset is a set of points, lines and polygons on the WGS84 reference spheroid, with geodesic edges.

You use the `GEOGRAPHY` data type by calling one of the standard SQL geography functions. The output of the geography functions is rendered as WKT (well-known text). WKT uses a longitude first, latitude second format.

This Qwiklab, and accompanying lecture, will introduce you to BigQuery itself, and then let you dive into running very large spatial queries quickly with BigQuery GIS.

What you'll do

- ✓ Run example Spatial Queries and learn about how they work
- ✓ Visualize the results of BigQuery GIS queries online or offline
- ✓ Create a virtual machine with the Google Cloud Platform Console
- ✓ Learn how to transform and load spatial data from common formats to BigQuery GIS
- ✓ Explore some bad data, and learn how to correct it
- ✓ Explore the BigQuery Public Datasets

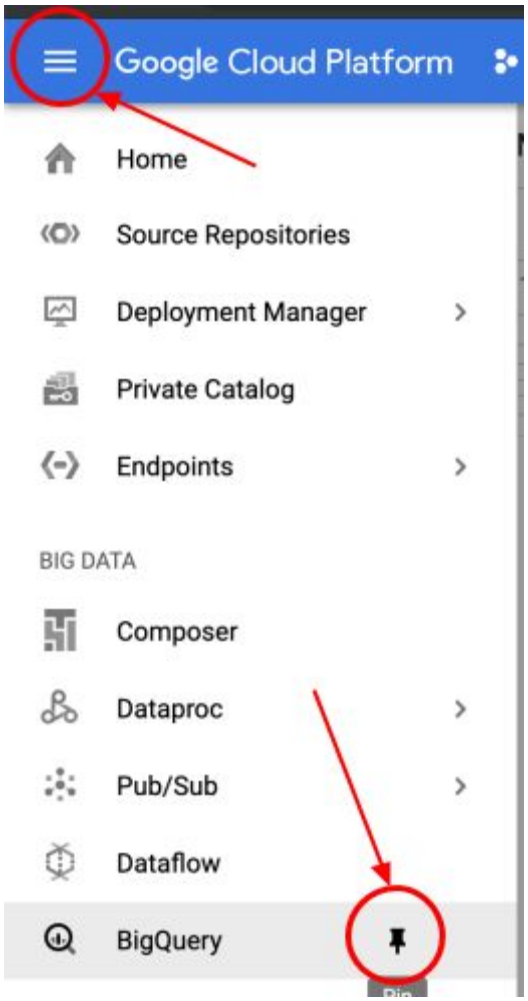
Prerequisites

- Familiarity with Google Cloud Platform, the linux terminal, SQL, GDAL, and desktop GIS applications will be helpful, but not required.

Section 1 - Intro to BigQuery GIS for Spatial Queries

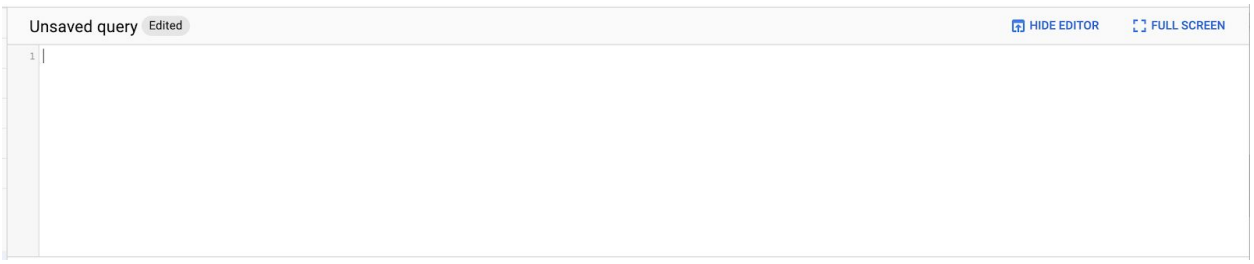
In this section, you'll have a chance to run some of the queries from the lecture to see the performance for yourself, as well as learn best practices for cleaning and preparing data for import into BigQuery GIS tables and datasets.

In the Google Cloud Platform console, click the **Menu** icon in the top left of the screen:



Under **Big Data**, hover over BigQuery and click the **Pushpin** to pin BigQuery to the top of the Developer's console. Then, click on **BigQuery.**

You will now see the BigQuery User Interface.



In the query dialogue, enter the following query:

```
SELECT Count(distinct a.objectid) FROM `vt-solar.vtdata.vtParcels`
a , `vt-solar.vtdata.vtImpervious` b  where
st_intersects(a.the_geom, b.the_geom)
```

Click the "**Run**" button

Note: The dataset "vt-solar.vtdata" comes from another GCP project, but it has been shared with all authenticated Google Users as *bigquery.DataViewers*

[RETURN TO LECTURE]

Section 2 - More advanced Spatial Queries

Now what about a question that is an order of magnitude more difficult?

In this case, we want to figure out *"For each parcel that has an intersection with the impervious surface layer, what **percent of each parcel** is impervious?"*

In the query dialogue, clear the previous query and enter the following query:

```
WITH parcel_impervious_area AS (
  SELECT p.objectid, Sum(ST_Area(ST_Intersection(im.the_geom,
```

```
p.the_geom))) AS impervious_area
  FROM `vt-solar.vtdata.vtImpervious` im
  JOIN  `vt-solar.vtdata.vtParcels` p
  ON ST_Intersects(im.the_geom, p.the_geom)
  GROUP BY p.objectid
)
SELECT p.objectid,p.town,
  CASE
    WHEN pia.impervious_area IS NULL
    THEN 0.0
    ELSE pia.impervious_area
  END AS impervious_area,
  CASE
    WHEN pia.impervious_area IS NULL
    THEN 0
    ELSE round(100 * pia.impervious_area / ST_Area(p.the_geom))
  END AS impervious_pct,p.the_geom
FROM  `vt-solar.vtdata.vtParcels` p
LEFT JOIN parcel_impervious_area pia
ON p.objectid = pia.objectid
ORDER BY impervious_pct DESC
```

Impressively, this should take just around 90 seconds to complete!

Examine the results. We've ordered them by impervious_pct, descending.

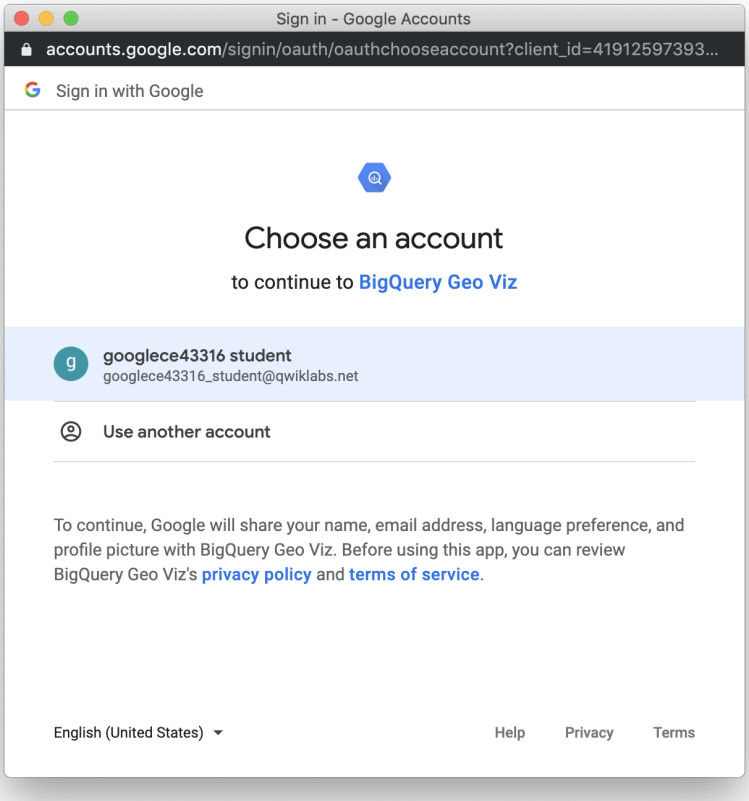
Row	objectid	town	impervious_area	impervious_pct	the_geom
1	1542061	ESSEX	956.2404156458163	200.0	-73.0470992722545 44.5074815384019, -73.0470913724974 44.5074844128596, -73.0470834784686 44.5074880342411, -73.0470753770755 44.5074918051057, -73.0470683236337 44.5074960191213, -73.0470610642249 44.5075005320037, -73.0470546413687 44.5075053405595, -73.0470484285128 44.5075101472091, -73.0470426388887 44.507515698783, -73.0470374741229 44.5075210979735, -73.0470327283973 44.5075267939374, -73.0470284014244 44.5075324869956, -73.0470249069529 44.5075381769663, -73.0470216238877 44.5075440153147, -73.0470189670703 44.5075498497636, -73.0470171440125 44.5075556811192, -73.0470153139686 44.5075607655565, -73.0470141145124 44.507566444535,

We shouldn't expect values for impervious_pct to be larger than **100**, so what is going on?

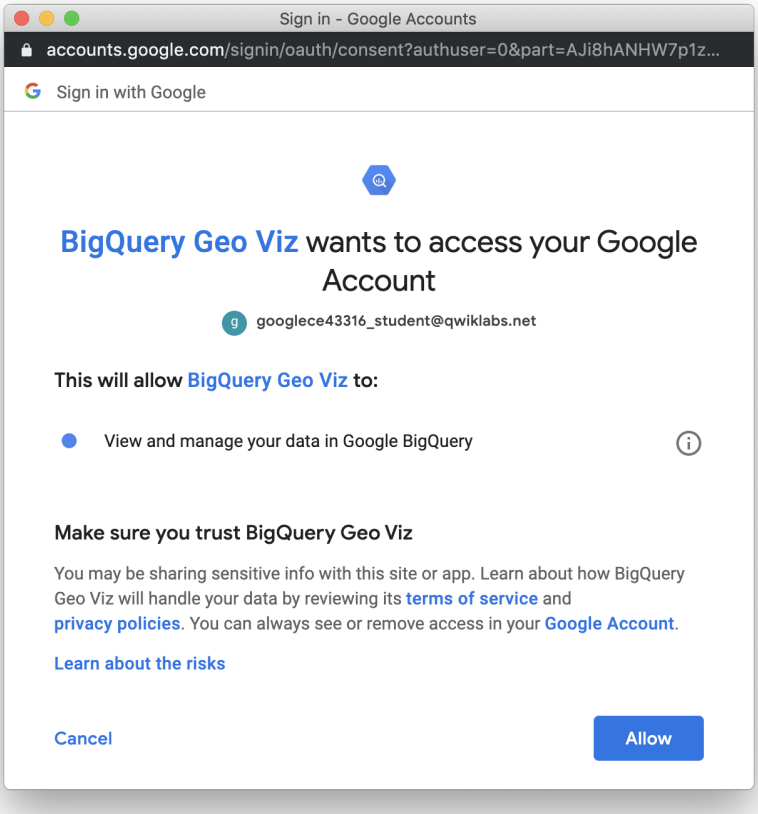
We will use the BigQueryGeoViz tool to investigate. This viewer allows you to query BigQuery GIS tables and plot and style them on top of the Google Maps API. It also utilizes the open source deck.gl library from Uber to allow the rendering of very high number of features client side in your browser, however, it does have practical limitations. Also, you can not share a visualization created in BigQueryGeoViz.

Open up BigQueryGeoViz if you have not already. Click the "**Authorize"** button.

Choose the qwiklabs account on the OAuth 2.0 Authorization Screen

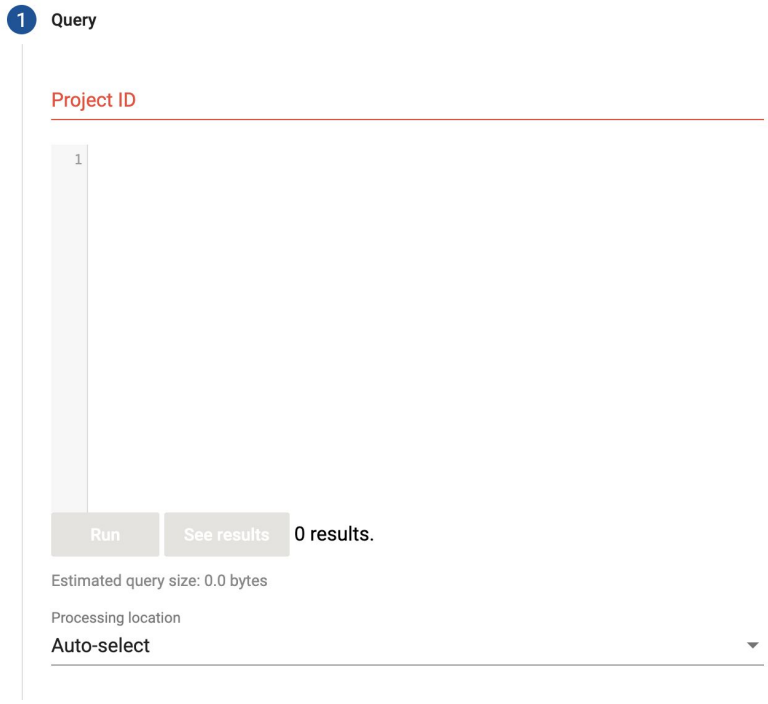


Whenever you're granting access to an application to act on your behalf via OAuth, it is important to review the permissions you are granting. Review the permissions and click "**Allow**".

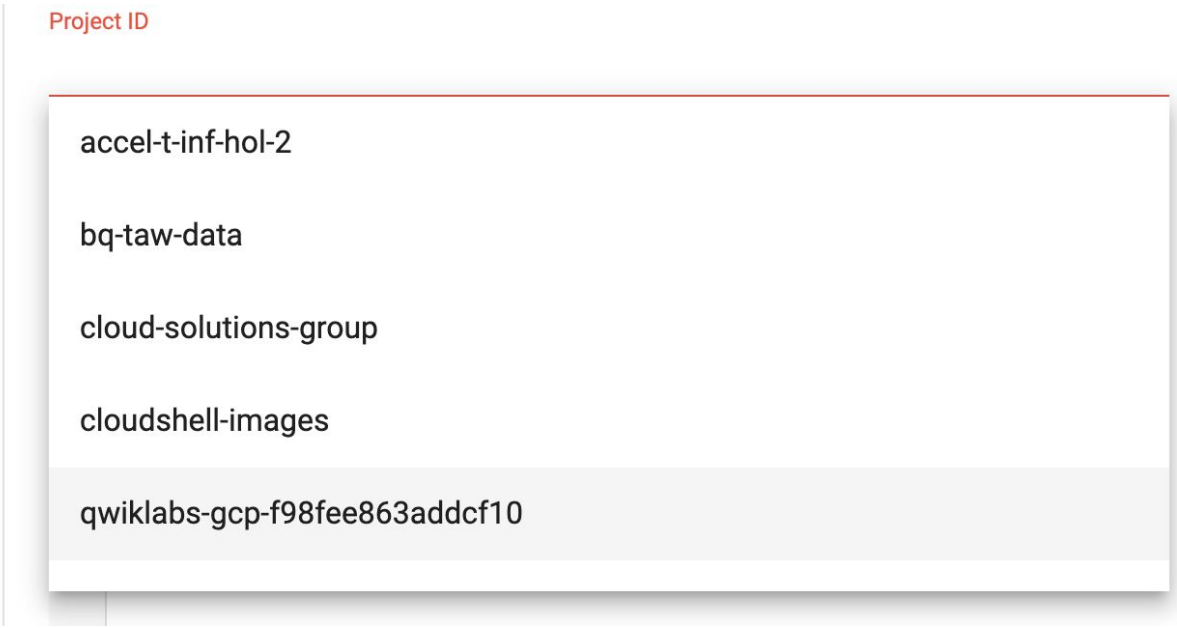


Now BQGeoViz is authorized to view and manage your data in BigQuery.

Click the "**Project ID"** to choose in which GCP project queries will run.



You may see more than one project, but pick the project that Qwiklabs generated for you when you started this lab.



Now we will use the BigQueryGeoViz tool to investigate some of of our unexpected results.

For instance, let's try to figure out how this parcel, at the top of our results in the town of Essex, has 200% impervious surface coverage?

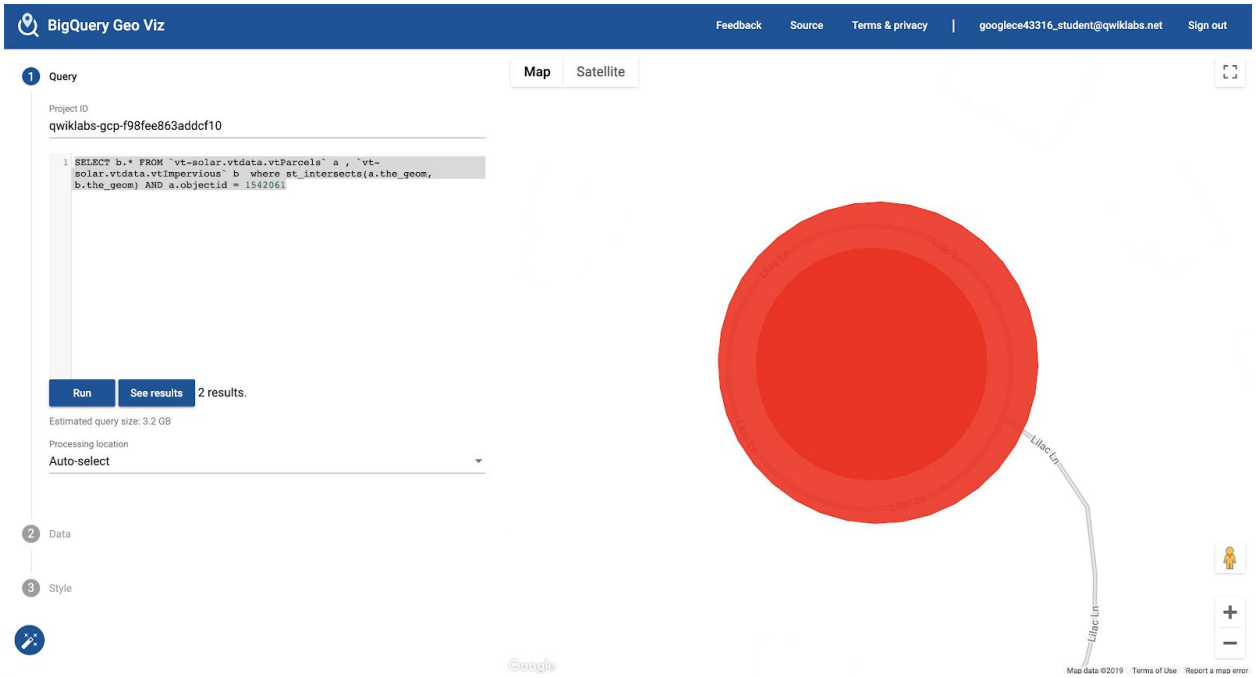
Row	objectid	town	impervious_area	impervious_pct
1	1542061	ESSEX	956.2404156458163	200.0

Copy the following query into the BigQueryGeoViz's query dialogue:

```
SELECT b.* FROM `vt-solar.vtdata.vtParcels` a ,
`vt-solar.vtdata.vtImpervious` b  where st_intersects(a.the_geom,
b.the_geom) AND a.objectid = 1542061
```

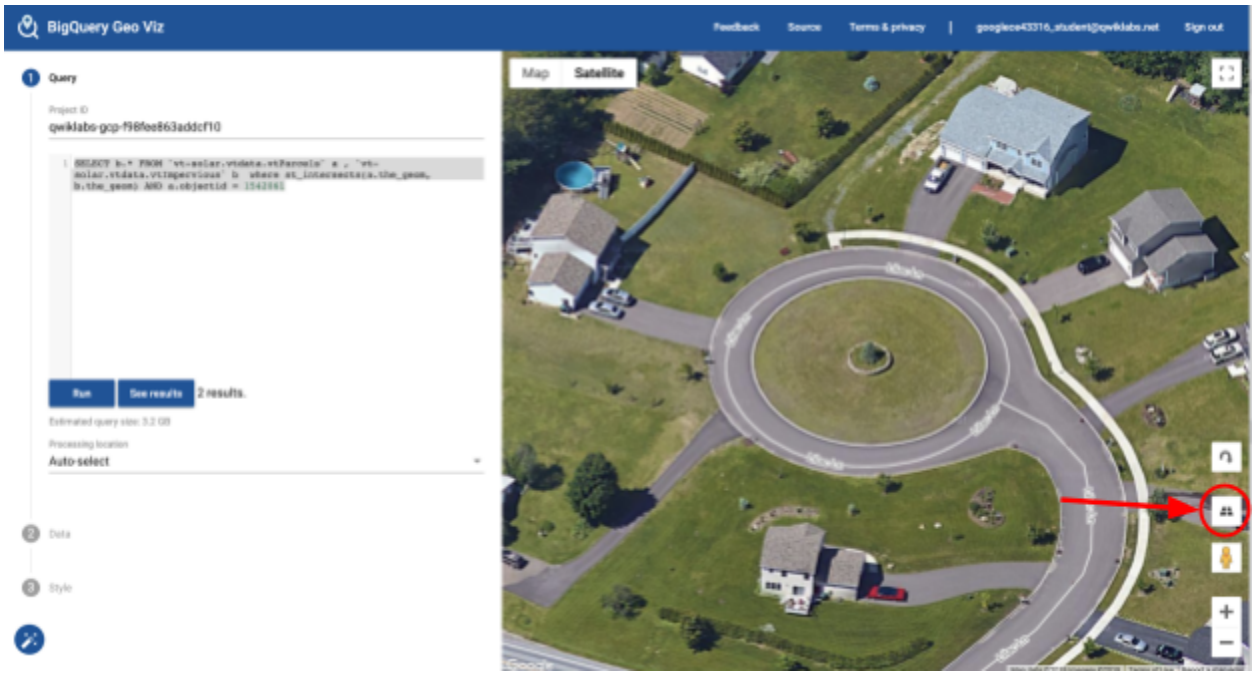
Then, click **"Run"**

By default, GeoViz will zoom you to the extent of the result, but this doesn't tell us too much:

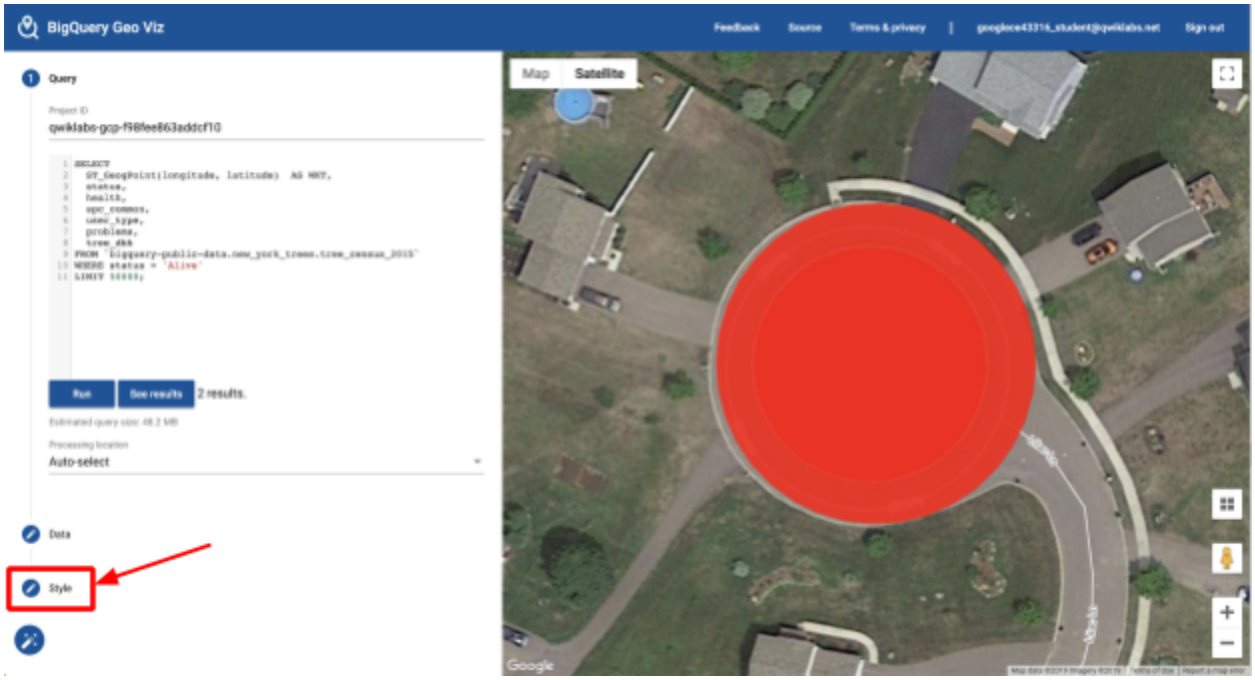


First, on the top of the Google Map, Toggle over to **"Satellite"** mode.

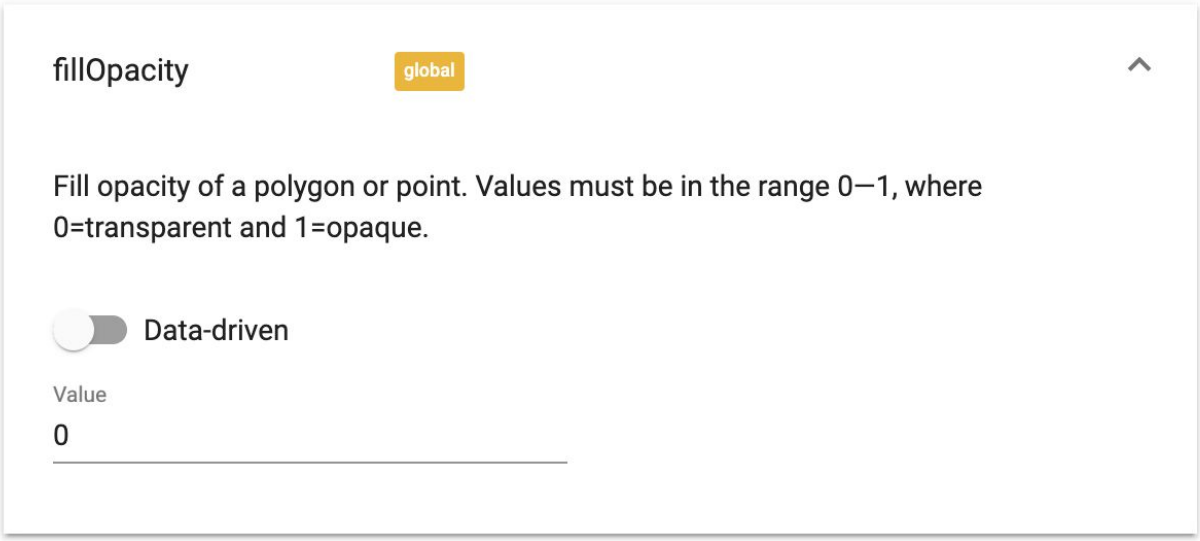
Interesting, this appears to be some sort of fancy Vermontian cul-de-sac.. However, clicking over to Satellite actually put us in 3D View, so we want to click the toggle to return to an orthogonal view.



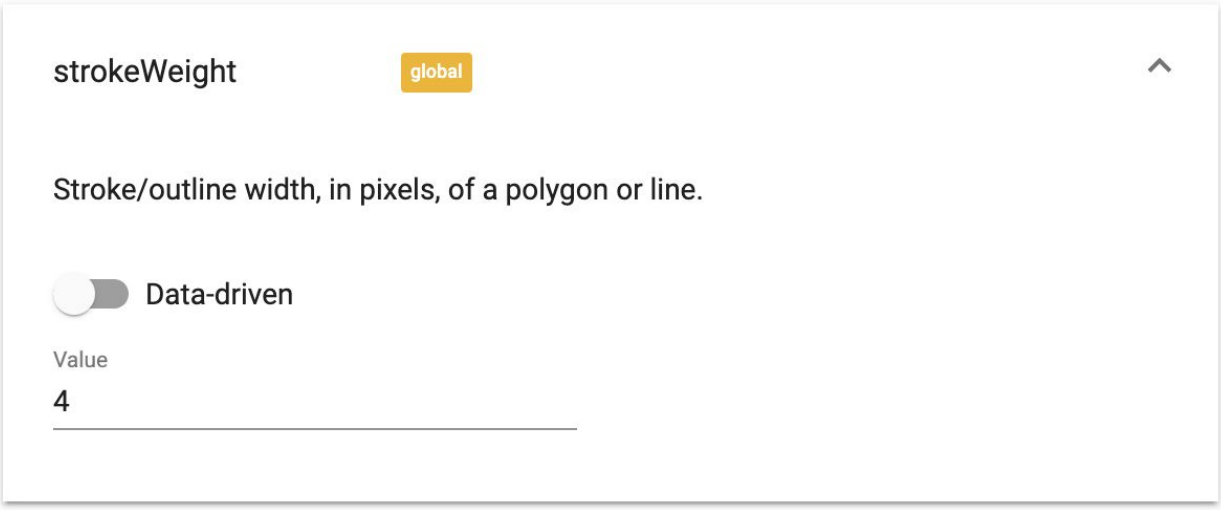
Better, but it is still tough to see what is going on. Let's try styling the polygon.



Change the **fillOpacity** to **0**



And the **strokeWeight** to **4.**

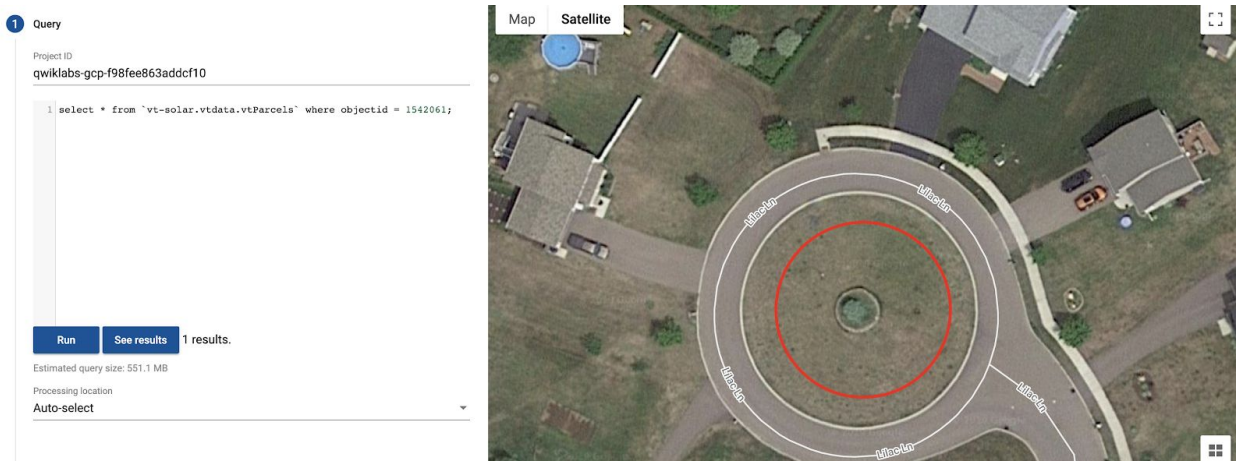


Now we can more clearly see that there are actually **2** impervious surface parcels that were returned for our query, meaning 2 of these polygons overlay our parcel 100%. Thus, this is a data issues in the impervious surface layer and our coverage is returned as 200%.

You can also see that there are 2 polygons if you click on the **"Data"** section.

If you want to see the parcel itself, change the query to:

```
select * from `vt-solar.vtdata.vtParcels` where objectid = 1542061;
```



Using BigQuery GIS to *quickly* identify data issues is a common use case.

Now that we know we can quickly run very complex and large spatial queries with BigQuery GIS, what if you wanted to test a hypothesis about this data? For instance, you could hypothesize the following:

"*I expect there to be a higher average percentage of impervious landcover per parcel in urban areas of Vermont vs rural areas.*"

We can use a GROUP function with our previous large query to test that.

Back in the BigQuery UI, run the following query:

```
WITH parcel_impervious_area AS (
  SELECT p.objectid, Sum(ST_Area(ST_Intersection(im.the_geom,
p.the_geom))) AS impervious_area
  FROM `vt-solar.vtdata.vtImpervious` im
  JOIN  `vt-solar.vtdata.vtParcels` p
  ON ST_Intersects(im.the_geom, p.the_geom)
  GROUP BY p.objectid
)
Select town, Avg(impervious_pct) as AveragePct FROM(
SELECT p.objectid,p.the_geom,p.town,
  IF(pia.impervious_area IS NULL, 0.0,
round(100 * pia.impervious_area / ST_Area(p.the_geom))) AS
impervious_pct
FROM  `vt-solar.vtdata.vtParcels` p
```

```
LEFT JOIN parcel_impervious_area pia
ON p.objectid = pia.objectid
)  Group by town order by AveragePCT Desc
```

```
WITH parcel_impervious_area AS (
  SELECT p.objectid, Sum(ST_Area(ST_Intersection(im.the_geom, p.the_geom))) AS impervious_area
  FROM `vt-solar.vtdata.vtImpervious` im
  JOIN `vt-solar.vtdata.vtParcels` p
  ON ST_Intersects(im.the_geom, p.the_geom)
  GROUP BY p.objectid
)
Select town, Avg(impervious_pct) as AveragePct FROM(
SELECT p.objectid,p.the_geom,p.town,
  CASE
    WHEN pia.impervious_area IS NULL
    THEN 0.0
    ELSE pia.impervious_area
  END AS impervious_area,
  CASE
    WHEN pia.impervious_area IS NULL
    THEN 0
    ELSE round(100 * pia.impervious_area / ST_Area(p.the_geom))
  END AS impervious_pct
FROM  `vt-solar.vtdata.vtParcels` p
LEFT JOIN parcel_impervious_area pia
ON p.objectid = pia.objectid
)  Group by town order by AveragePCT Desc
```

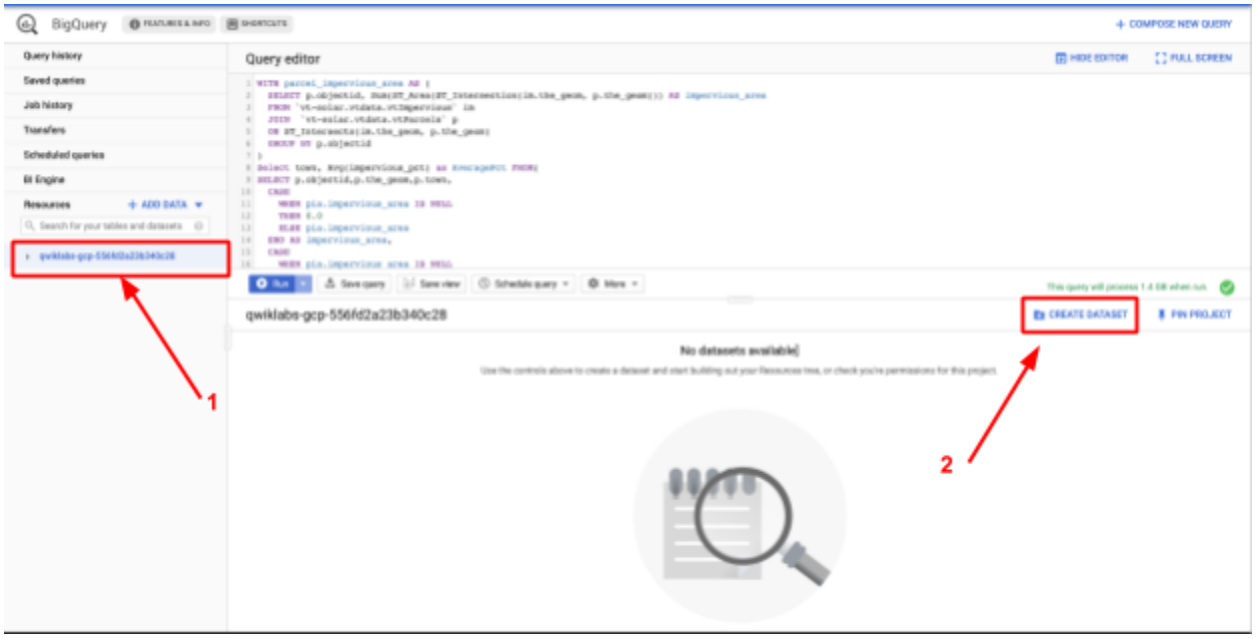
The results prove our hypothesis in around a minute! There is a higher average percentage of impervious landcover per parcel in urban areas of Vermont vs rural areas.

Row	town	AveragePct
1	WINOOSKI	42.94353472614343
2	BURLINGTON	42.25106240214717
3	ST. ALBANS CITY	41.569741026805985
4	BARRE CITY	40.42732993706921
5	RUTLAND CITY	38.887080191912304
6	STRATTON	37.831355402873236
7	MONTPELIER	36.660134542263826
8	BOLTON	32.63314285714288
9	VERGENNES	28.762806236080174
10	ST. JOHNSBURY	26.73846153846153
11	BENNINGTON	25.650932684509325

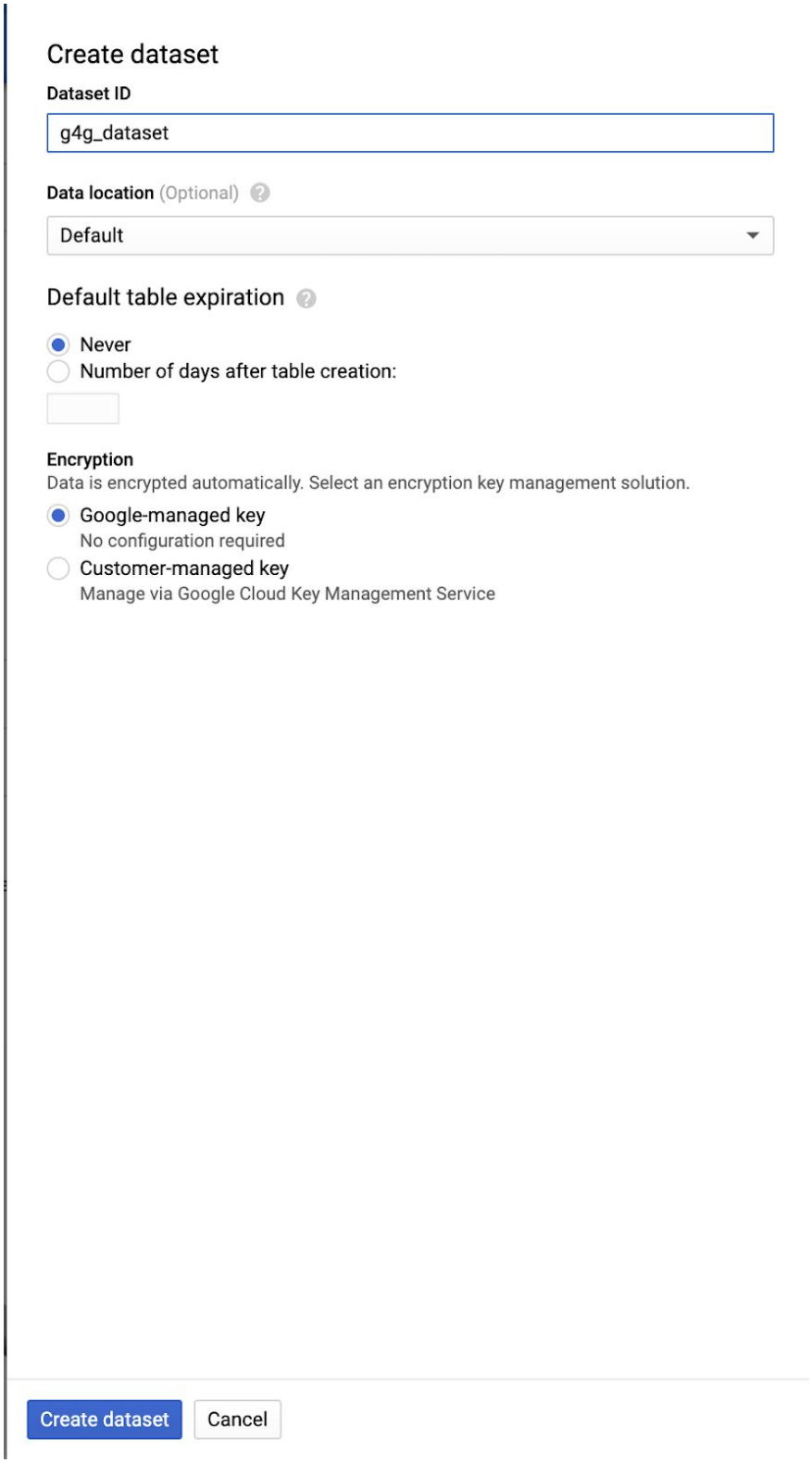
What if we wanted to map this to see it spatially and confirm? We don't currently have a BigQuery GIS table for the town geometries, so in Section 3 we'll cover loading data to BigQuery GIS.

First, however, let's quickly create a new BigQuery Dataset in our project. Dataset names are unique *per project*, so we can all name our dataset the same thing.

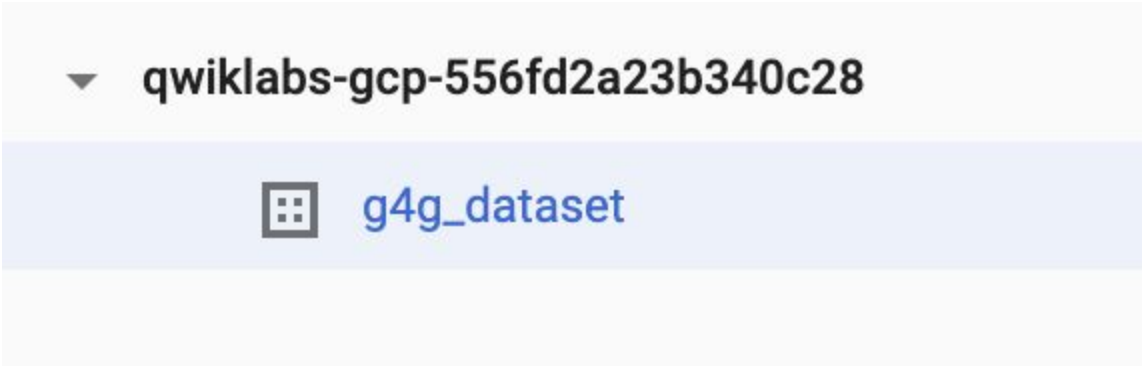
In the BigQuery User Interface, click on your project name in the left panel. Then, click **“Create Dataset”**.



Name your dataset “**g4g_dataset**”. Then click “**Create Dataset”.**



You should now be able to expand your project name on the left side of the User Interface and see your new, empty dataset.



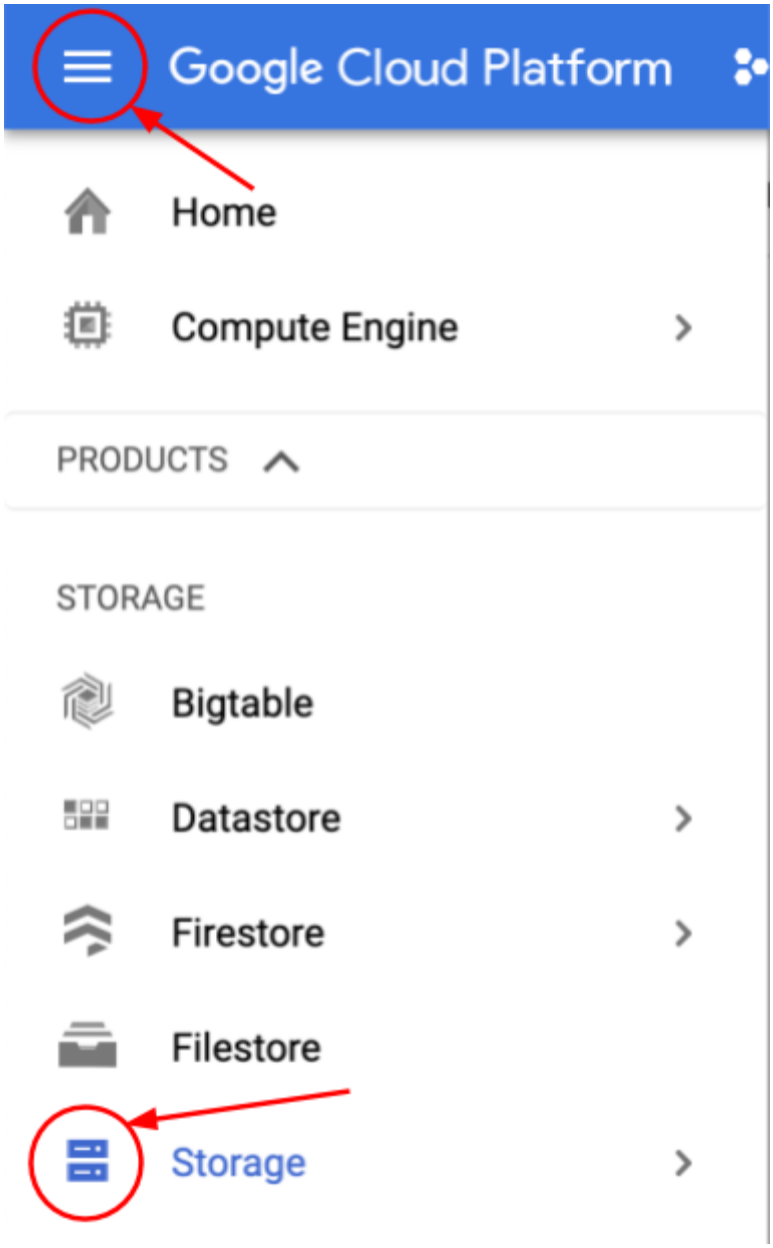
Section 3 - Working with YOUR Data When You Leave Here.

These Qwiklabs are nice and all, but they're designed so that everything works with simple copy and paste. So far, all the queries we've demonstrated use data that Google has cleaned and ingested. So, we want to make sure you're prepared to use BigQuery GIS with YOUR data when you leave here.

We'll need to use 2 additional components of Google Cloud Platform for this section. Compute Engine, and Google Cloud Storage.

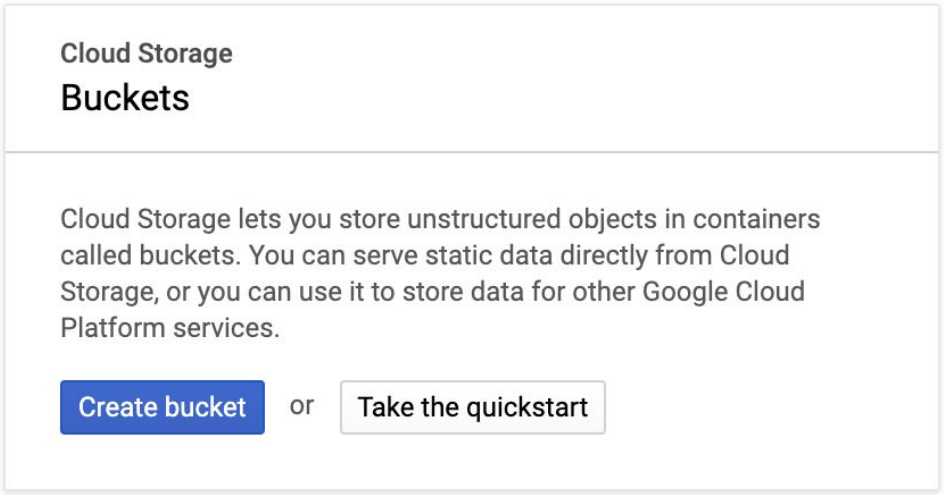
Google Cloud Storage

Google Cloud Storage is Google Cloud Platform's Object storage. We will be using it as a staging location for data that we generate in a Virtual Machine that we want to load into BigQuery GIS.
In the Google Cloud Platform Console, click the **Menu** icon on the top left of the screen:

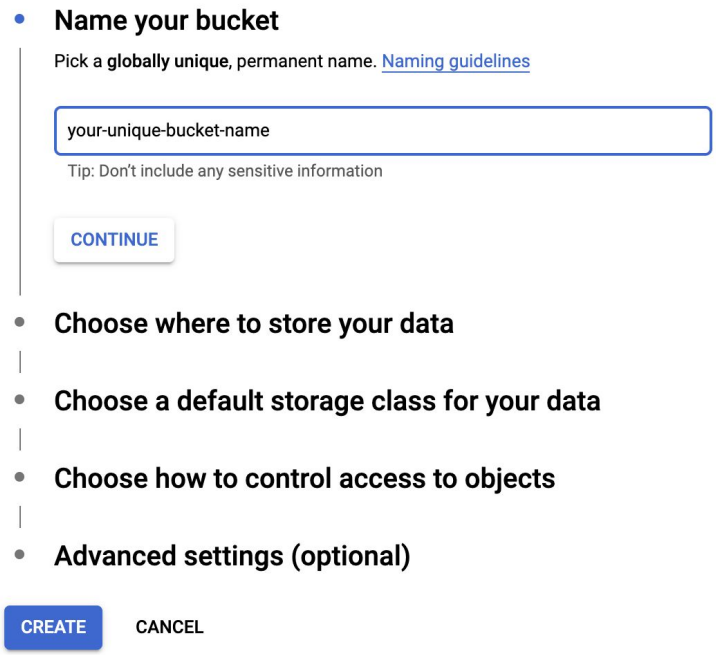


Then navigate to **Storage** > **Storage.**

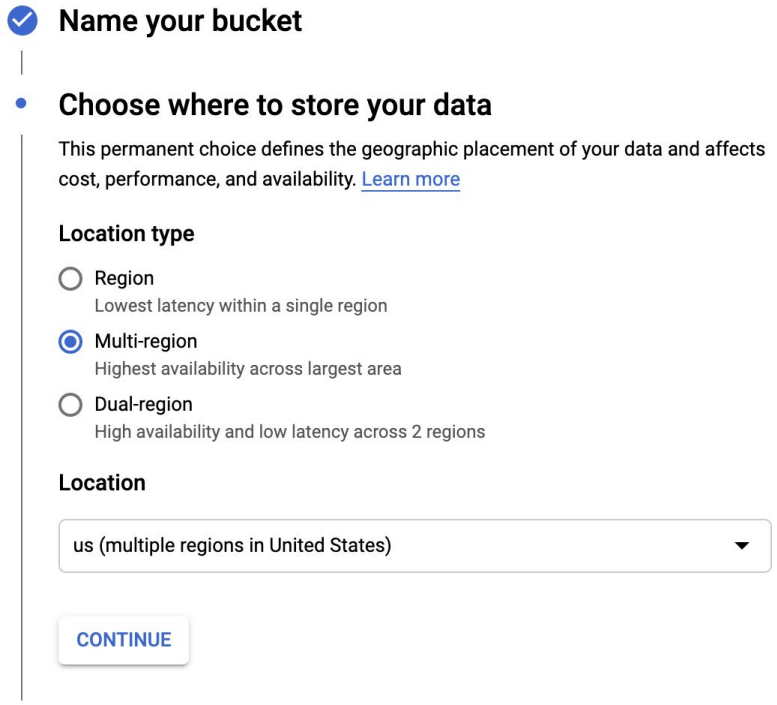
This will take you to Cloud Storage where you will create a bucket that you will use throughout this lab. Note - buckets have to have UNIVERSALLY unique names, so, pick something unique that you'll remember...but maybe not the name of a major project you're working on in case you want to name a bucket that name later! Click **"Create bucket"**.



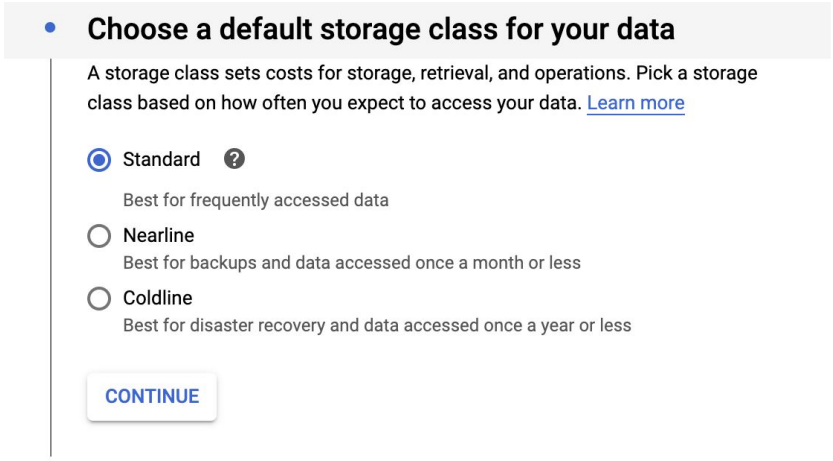
Next, enter your unique name for your bucket (and remember it). Then click **"Continue".**



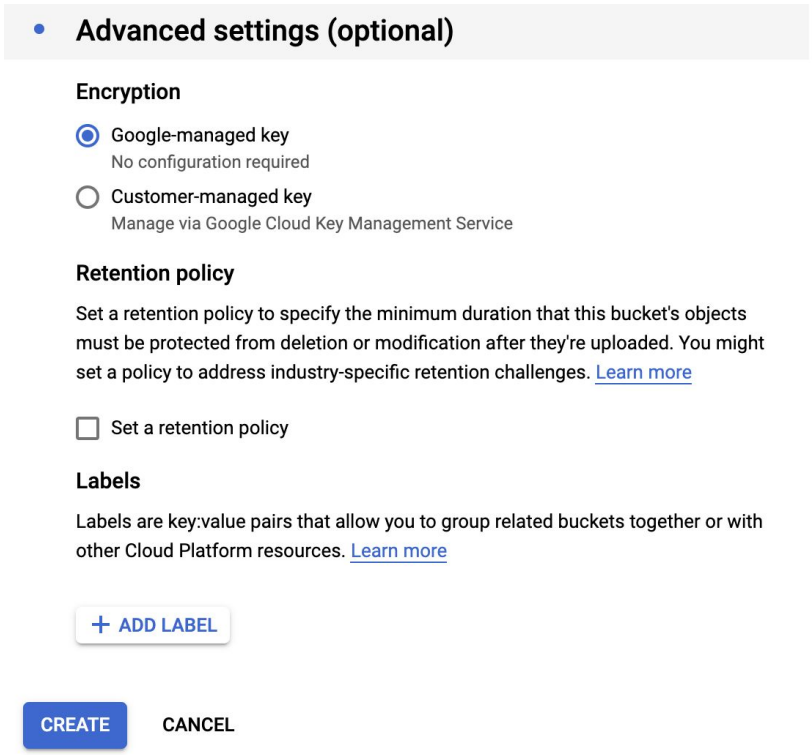
Next, choose where to store your data. We'll use the default, which is Multi-region. Then click **"Continue".**



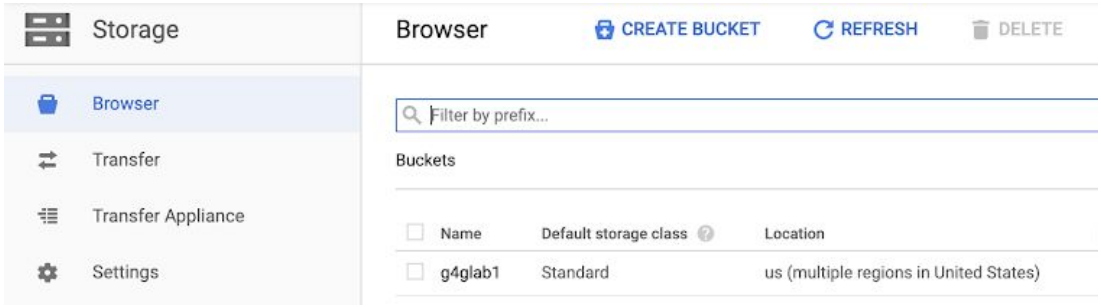
Next, choose a default storage *class* for your data. Chose the default, **"Standard"**, then click **"Continue".**



Finally, under **"Advanced Settings (optional)"** leave the defaults in place and Then click **"Create".**



In the Storage Browser you should now see your bucket. Ours is called **"g4glab1"**.

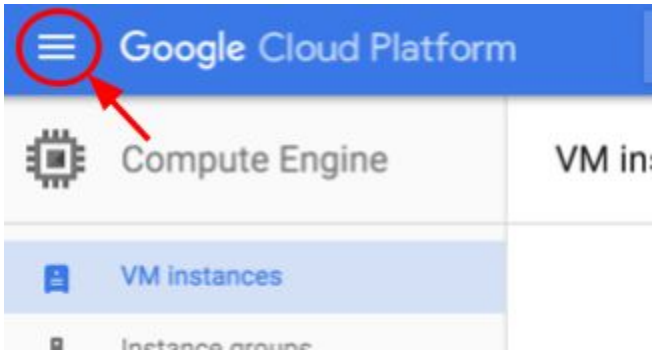


Note: *For the remained of this lab, substitute "your-unique-bucket-name" with the name of your actual bucket.*

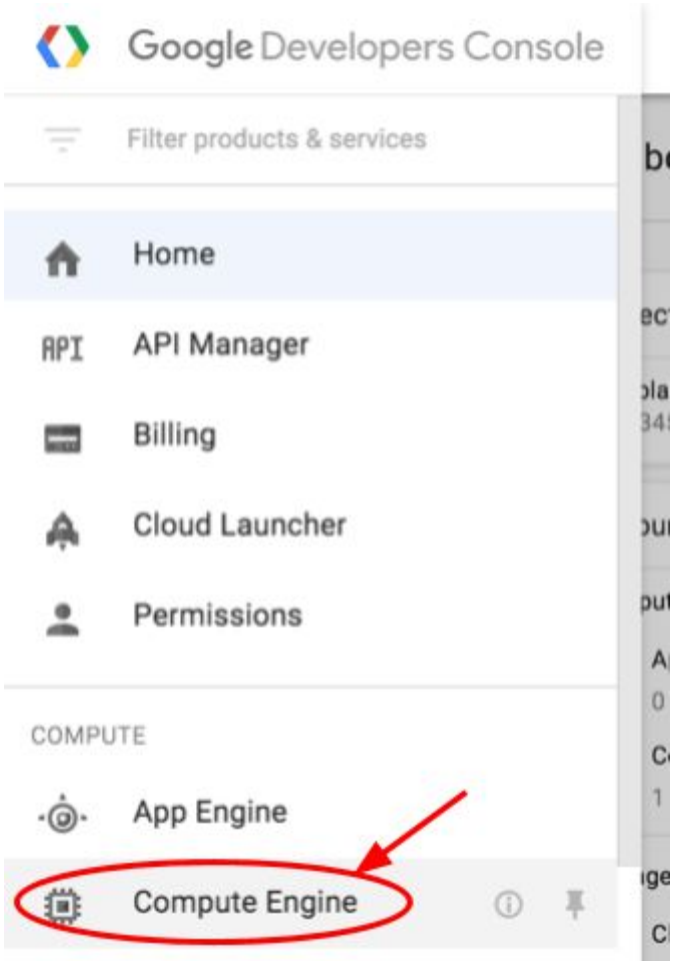
Google Compute Engine

Google Compute Engine is Google Cloud Platform's Infrastructure as a Service offering which allows you to create virtual machines. Generally, Google supplies and updates the operating system images used to create these virtual machines. We're going to create a virtual machine that we will use to download example geospatial data, transform it

In the Google Cloud Platform Console, click the **Menu** icon on the top left of the screen:

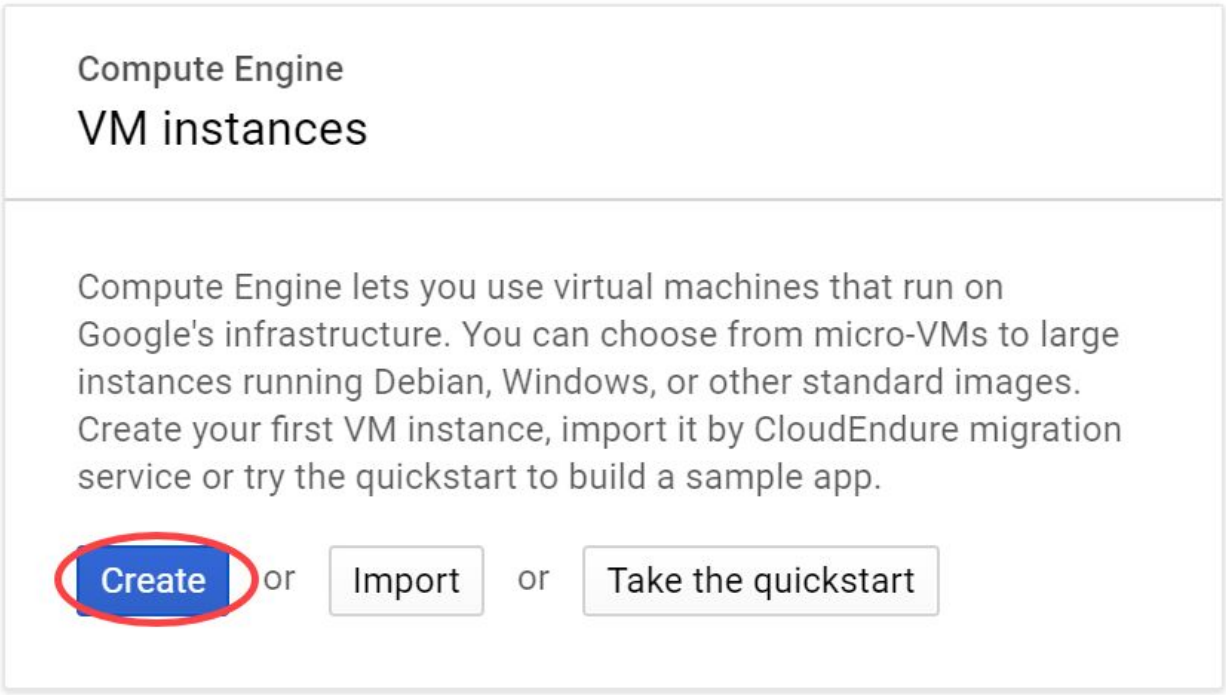


Then navigate to **Compute Engine** > **VM Instances.**



This may take a minute to initialize for the first time.

To create a new instance, click **Create**.



There are many parameters you can configure when creating a new instance. Use the following for this lab:

Name	`instance-1`	**Additional Information**
Zone	`us-central1-a`	Learn more about zones in Regions & Zones documentation.
Machine Type	`1 vCPU` This is a (n1-standard-1), 1-CPU, 3.75GB RAM instance. There are a number of machine types, ranging from micro instance types to 32-core/208GB RAM instance types. Learn more in the Machine Types documentation.	**Note**: A new project has a default resource quota, which may limit the number of CPU cores. You can request more when you work on projects outside of this lab.

Boot Disk	`New 10 GB standard persistent disk`
	`OS Image: Debian GNU/Linux 9 (Stretch)`
	`Boot disk type:Standard`
	`Size (GB): 10`
		There are a number of images to choose from, including: Debian, Ubuntu, CoreOS as well as premium images such as Red Hat Enterprise Linux and Windows Server. See

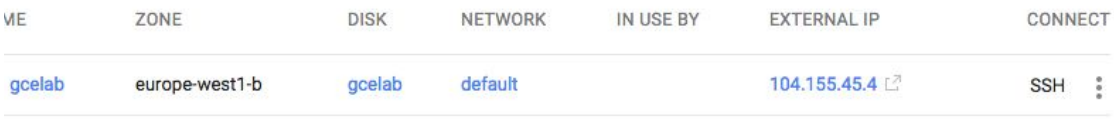
		Operating System documentation for more detail.
Identity and API access	Check `Access scopes` Check the `Allow full access to all Cloud APIs`	**Note:** This is just for simplicity, not necessarily what you'd want to do in production.

Note: The instance creation process is asynchronous. You can check on the status of the task using the top right hand-side **Activities** icon. Wait for it to finish - it shouldn't take more than a minute.

Hint: If you receive an error when creating a VM, click into **Details**. Most likely you need to try again with a different zone.

Once finished, you should see the new virtual machine in the **VM Instances** page.

To SSH into the virtual machine, click on **SSH** on the right hand side. This launches a SSH client directly from your browser.



Note: You can also SSH into the virtual machine. See the [Connect to an instance using ssh documentation].

As the `root` user, update your OS:

```
sudo apt-get update
```

(Output)

```
Get:1 http://security.debian.org jessie/updates InRelease [63.1 kB]
Ign http://httpredir.debian.org jessie InRelease
Get:2 http://security.debian.org jessie/updates/main amd64 Packages [247
kB]
...
```

Install GDAL and UNZIP:

```
sudo apt-get install gdal-bin unzip -y
```

(Output)

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
...
```

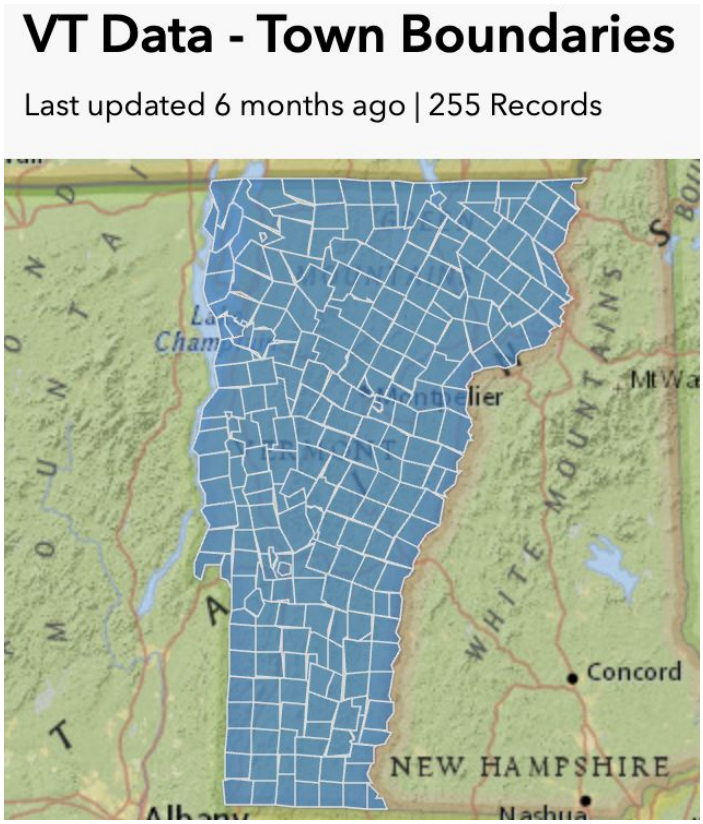
Check that ogr2ogr is installed:

```
which ogr2ogr
```

(Output)

```
/usr/bin/ogr2ogr
```

Now, we will download a zip file that contains a shapefile for all of the towns of Vermont.



This data comes from the [Vermont Open Data Portal] and our friends at VT CGI, but we have mirrored it to Google Cloud Storage already. We move data to and from Google Cloud Storage and our Virtual Machine using the gsutil command, which is part of the Google Cloud SDK and is prebuilt and authenticated for your project already.

```
gsutil cp gs://vtprepdata/VT_Data__Town_Boundaries.zip .
```

Note: *Make sure you have the trialing ".", that tells gsutil to copy the data to the current directory.*

Now we unzip this file to find the shapefile and all of its sidecar files.

```
unzip VT_Data__Town_Boundaries.zip
```

(Output)

```
Archive:  VT_Data__Town_Boundaries.zip
  inflating: VT_Data__Town_Boundaries.xml
 extracting: VT_Data__Town_Boundaries.cpg
  inflating: VT_Data__Town_Boundaries.prj
```

```
  inflating: VT_Data__Town_Boundaries.shp
  inflating: VT_Data__Town_Boundaries.dbf
  inflating: VT_Data__Town_Boundaries.shx
```

Next we want to understand what exactly we're looking at here with this particular shapefile. We can do that using the GDAL tool, **ogrinfo** to inspect this vector file.

```
ogrinfo -al -so VT_Data__Town_Boundaries.shp
```

(Output)

```
INFO: Open of `VT_Data__Town_Boundaries.shp'
      using driver `ESRI Shapefile' successful.

Layer name: VT_Data__Town_Boundaries
Geometry: Polygon
Feature Count: 255
Extent: (424788.840000, 25211.836900) - (581554.370100, 279798.473600)
Layer SRS WKT:
PROJCS["NAD83_Vermont",
    GEOGCS["GCS_North_American_1983",
        DATUM["North_American_Datum_1983",
            SPHEROID["GRS_1980",6378137,298.257222101]],
        PRIMEM["Greenwich",0],
        UNIT["Degree",0.0174532925199433295]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",42.5],
    PARAMETER["central_meridian",-72.5],
    PARAMETER["scale_factor",0.999964286],
    PARAMETER["false_easting",500000],
    PARAMETER["false_northing",0],
    UNIT["Meter",1]]
OBJECTID: Integer64 (10.0)
FIPS6: Integer64 (10.0)
TOWNNAME: String (80.0)
TOWNNAMEMC: String (80.0)
CNTY: Integer64 (10.0)
ShapeSTAre: Real (24.15)
ShapeSTLen: Real (24.15)
TOWNGEOID: String (80.0)
```

ogrinfo tells us a lot. First, we see the layer name inside the shapefile is "VT_Data__Town_Boundaries"

Second, there are 255 features in the shapefile.
Third, this shapefile is NOT in EPSG:4326, which BigQuery GIS requires, so, we'll have to handle that transformation before loading it.
Fourth, we see the field names, and their types. This will be critical information for constructing our BigQuery Table Schema for the table.

Now, we're going to use ogr2ogr to convert this shapefile into a CSV file with a GeoJSON-encoded "the_geom" field which will be our GEOGRAPHY field in BigQuery GIS.

```
ogr2ogr -f csv -dialect sqlite -sql "select
AsGeoJSON(Transform(geometry,4326)) the_geom, * from
VT_Data__Town_Boundaries" VT_Town_Boundaries.csv
VT_Data__Town_Boundaries.shp
```

A lot is going on in that command, so let's break it down.

"**ogr2ogr** -f csv" says we're going to be outputting a csv file.

"-dialect sqlite -sql" says that we're going to be doing a query of the data using the sqlite dialect which we'll need for special transformation and conversion functions.

"select AsGeoJSON(Transform(geometry,4326)) the_geom, * from VT_Data__Town_Boundaries" is our SQL statement that says:

"First, transform the 'geometry' field of the shapefile into EPSG:4326. Then, return that as a GeoJSON formatted field called "the_geom". Then, return all the other fields as is.
"VT_Data__Town_Boundaries" is the **Layer name** we were given by **ogrinfo**.

"VT_Town_Boundaries.csv" is the csv we want to export.

"VT_Data__Town_Boundaries.shp" is our input shapefile.

Next, we want to see the field order of the exported csv file.

```
head -1 VT_Town_Boundaries.csv
```

(Output)

```
the_geom,OBJECTID,FIPS6,TOWNNAME,TOWNNAMEMC,CNTY,ShapeSTAre,ShapeSTLen,TOWN
GEOID
```

(Optional) If you're interested in what the CSV looks like, you can read a second line, but you'll see the entire GeoJSON formatted geometry for the first town so it will take up a lot of screen space and we won't show that output here.

```
head -2 VT_Town_Boundaries.csv
```

Ok, now we have out csv file and we know the column order. It's a good time to construct the BigQuery Schema we're going to need to load this table. Remember, **ogrinfo** provided us with the field names *and types* which we'll need.

We know our new field, "the_geom" is going to be type GEOGRAPHY, the rest again are:

OBJECTID: Integer64
FIPS6: Integer64
TOWNNAME: String
TOWNNAMEMC: String
CNTY: Integer64
ShapeSTAre: Real
ShapeSTLen: Real
TOWNGEOID: String

Thus, our schema will be:

*the_geom:GEOGRAPHY,OBJECITD:INTEGER,FIPS6:INTEGER,TOWNNAME:STRING,TOWN
NAMEMC:STRING,CNTY:INTEGER,ShapeSTAre:FLOAT,ShapeSTLen:FLOAT,TOWNGEOID:
STRING*

Next, we need to move our csv file off of the local hard drive of our Virtual Machine and to our Google Cloud Storage bucket. We do this with gsutil. (Remember, you'll want to copy and paste the command below but replace your actual bucket name.)

```
gsutil cp VT_Town_Boundaries.csv gs://your-unique-bucket-name/
```

Now that the csv file is in GCS, we can use the BigQuery command line tool to direct BigQuery to ingest this csv file into a new table in our dataset. We do this using the **bq** tool which is already installed. (Remember, you'll want to copy and paste the command below but replace your actual bucket name.)

```
bq load --source_format=CSV --skip_leading_rows=1 --replace
--schema=the_geom:GEOGRAPHY,OBJECITD:INTEGER,FIPS6:INTEGER,TOWNNAME
:STRING,TOWNNAMEMC:STRING,CNTY:INTEGER,ShapeSTAre:FLOAT,ShapeSTLen:
FLOAT,TOWNGEOID:STRING g4g_dataset.VT_Town_Boundaries
gs://your-unique-bucket-name/VT_Town_Boundaries.csv
```

Again, that was a big command, so we'll break it down.

"bq load --source_format=CSV --skip_leading_rows=1 --replace" - This says to use the **bq** tool, and we're going to load data to a table. We're using a CSV source file, and because of that, we want to skip the header row (1). We also use --replace in case this table currently exists, we'll overwrite it.

Next, we provide our preconstructed schema to the --schema flag.

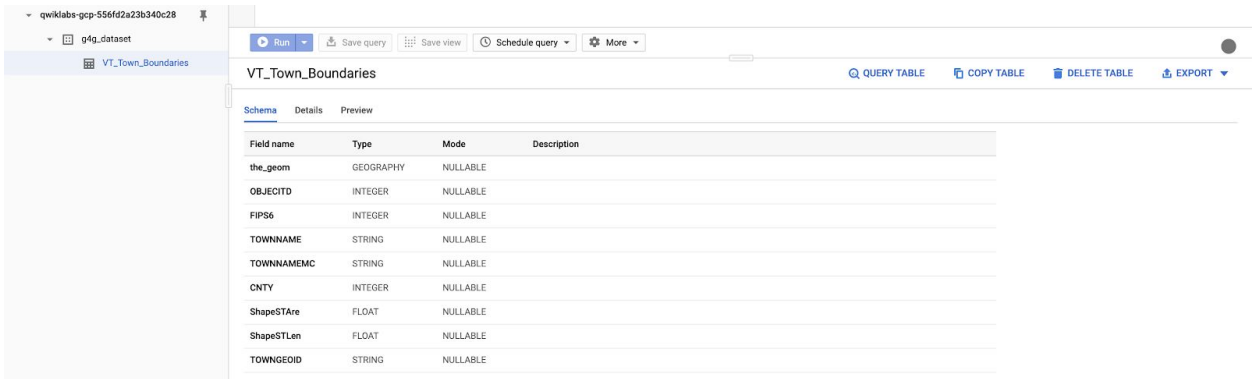
g4g_dataset.VT_Town_Boundaries - This says create or replace a table called VT_Town_Boundaries in your g4g_dataset Dataset.

gs://your-unique-bucket-name/VT_Town_Boundaries.csv - This is our source file for the load job.

(Output) You'll see something like:

```
Waiting on bqjob_r6fc610b83aeb59f9_0000016d21d8a907_1 ... (1s) Current
status: DONE
```

Once the job is done, In the BigQuery User Interface you will see the new table in your dataset.



Now, we can construct a query that will join this new table from our dataset with the publicly shared VT data that we've been analyzing previously.

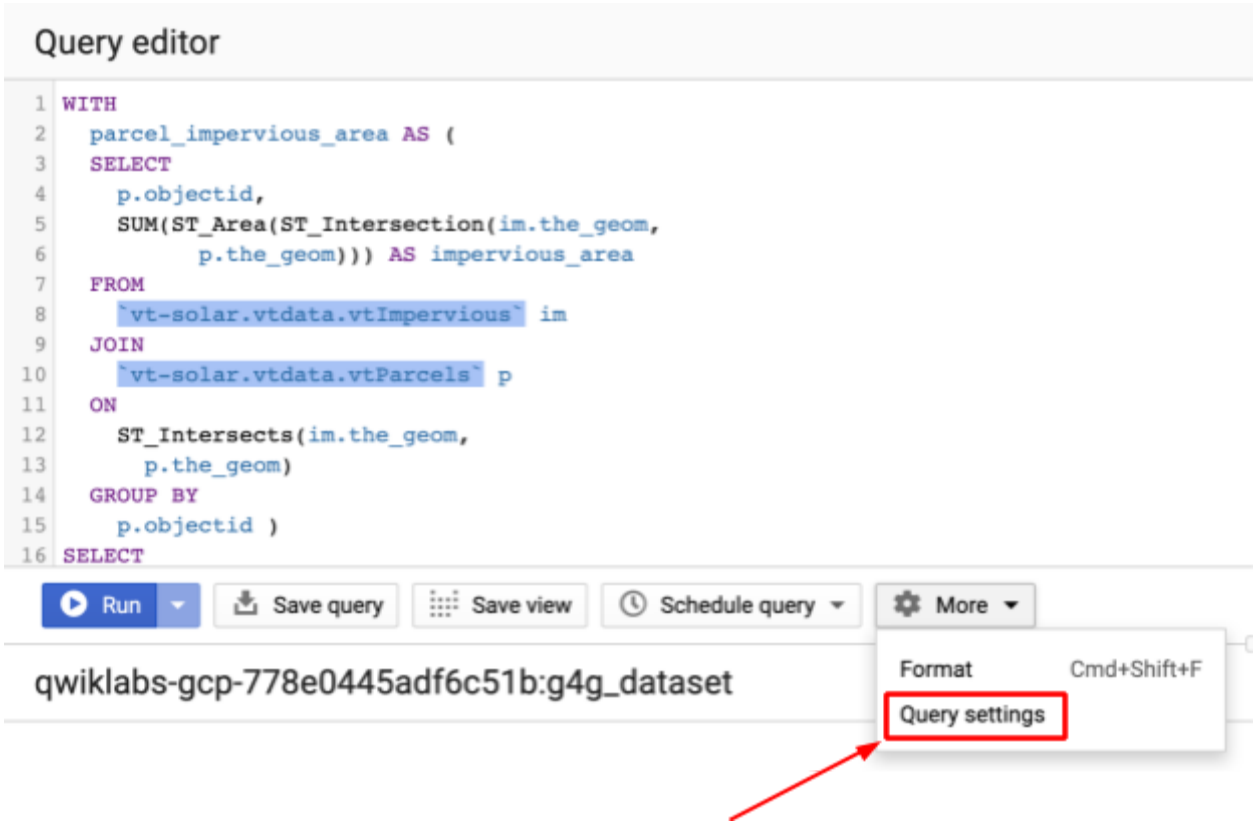
In the Big Query Query Editor, enter the following query:

```
WITH
  parcel_impervious_area AS (
  SELECT
    p.objectid,
    SUM(ST_Area(ST_Intersection(im.the_geom,
          p.the_geom))) AS impervious_area
  FROM
    `vt-solar.vtdata.vtImpervious` im
  JOIN
    `vt-solar.vtdata.vtParcels` p
  ON
    ST_Intersects(im.the_geom,
      p.the_geom)
  GROUP BY
    p.objectid )
SELECT
  towns.the_geom,
  towns.TOWNNAME,
  percentages.AveragePct
FROM
  `g4g_dataset.VT_Town_Boundaries` towns
LEFT JOIN (
  SELECT
    town,
    AVG(impervious_pct) AS AveragePct
  FROM (
    SELECT
      p.objectid,
      p.the_geom,
      p.town,
    IF
      (pia.impervious_area IS NULL,
        0.0,
        ROUND(100 * pia.impervious_area / ST_Area(p.the_geom))) AS
impervious_pct
    FROM
      `vt-solar.vtdata.vtParcels` p
    LEFT JOIN
      parcel_impervious_area pia
    ON
```

```
      p.objectid = pia.objectid )
  GROUP BY
    town ) percentages
ON
  towns.TOWNNAME = percentages.town
  -- WHERE
  --   percentages.AveragePct IS NOT NULL
ORDER BY
  AveragePCT DESC
```

However, rather than just running the query for a temporary table, this time we'll save the results to your dataset so it is persistent for future operations.

In the BigQuery UI, click the **"More"** button and then choose Query settings.

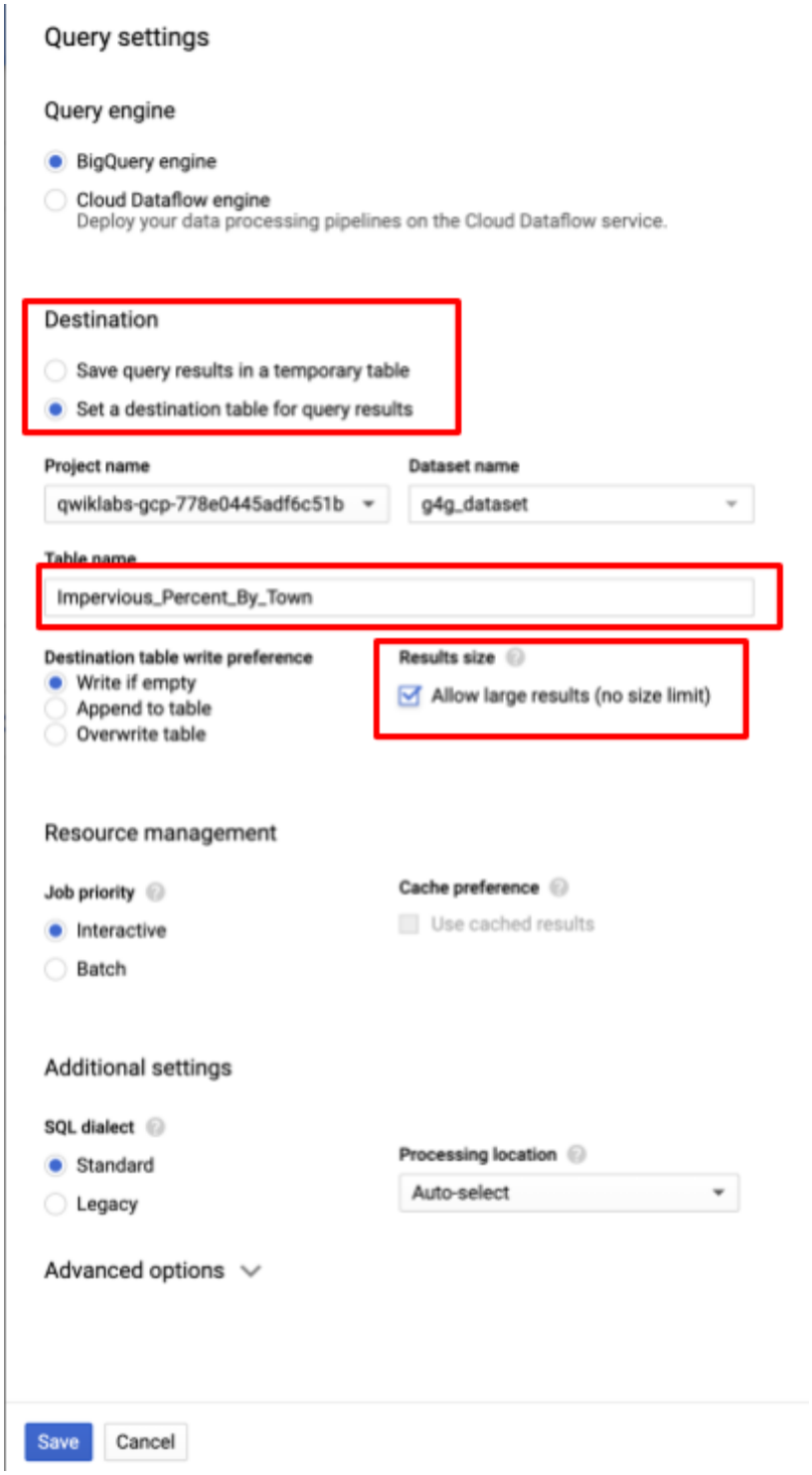


Next, under **Destination**, choose "Set a destination table for query results". Make sure your current project is selected and that the g4g_dataset is as well.

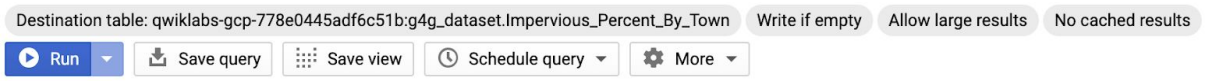
For the **Table name** enter:

```
Impervious_Percent_By_Town
```

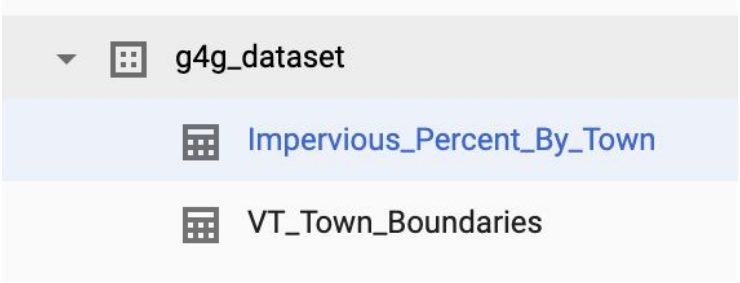
Finally, under **Results size**, check the box to “Allow large results (no size limit)”. This isn’t a large table by any means, but, this is a good practice in general to ensure large queries aren’t cut off before they finish. Then, click save.



Now you should see similar query settings above the **Run** button in the UI.



Click **Run** and the query should create the new table in your g4g_dataset.

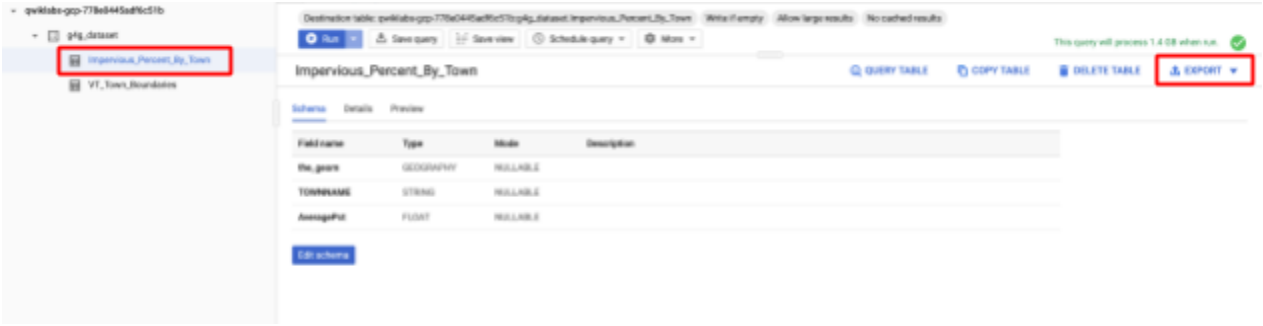


Visualizing Results

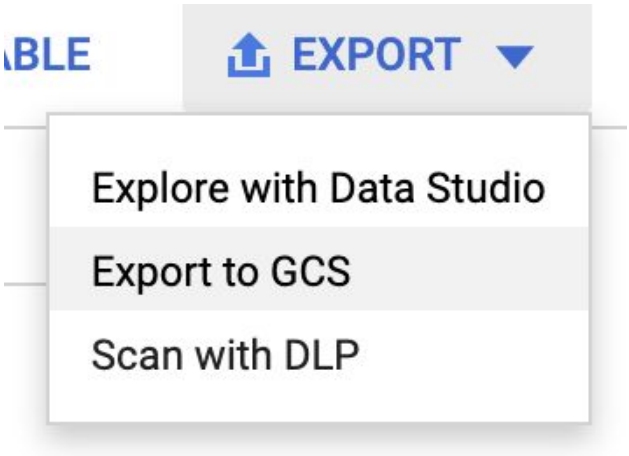
There are 2 major ways to visualize the data; online and offline. For the next segment, choose either one (or both!)

Offline (Using QGIS)

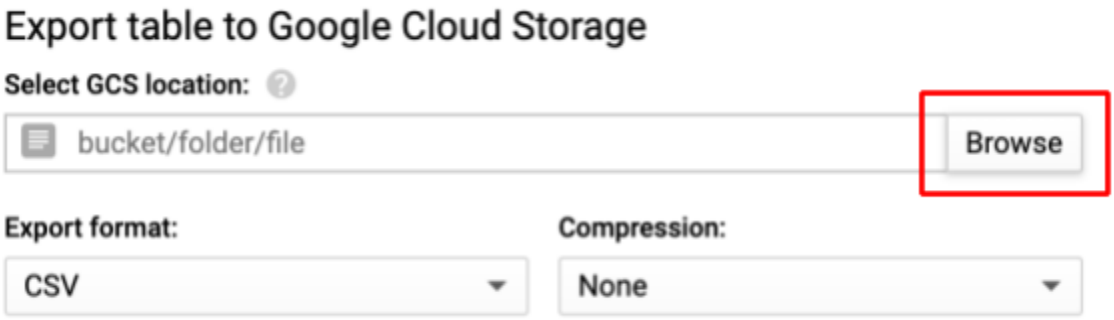
To visualize the new table offline in QGIS, you'll need to export the table. Click on the new table and then click on **Export.**



Chose **"Export to GCS"**

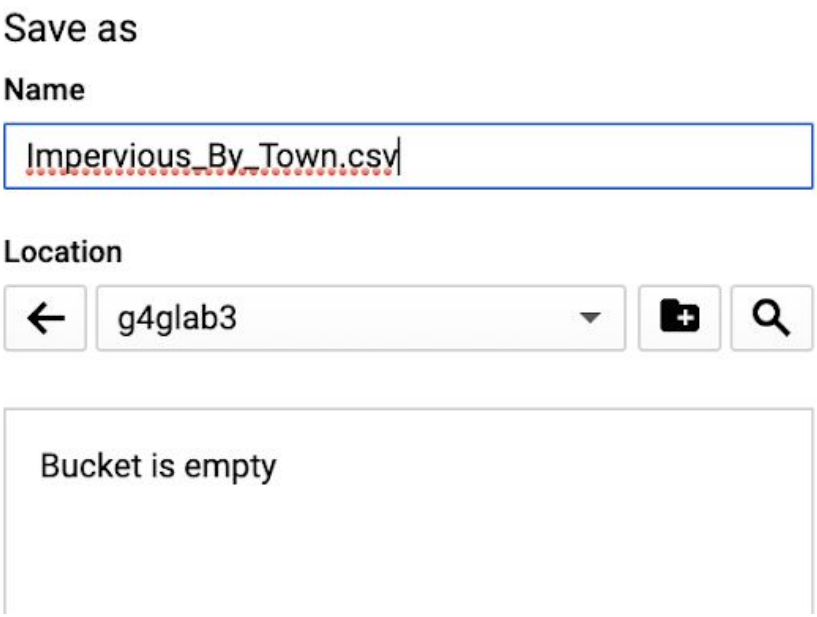


Then, click **"Browse"**

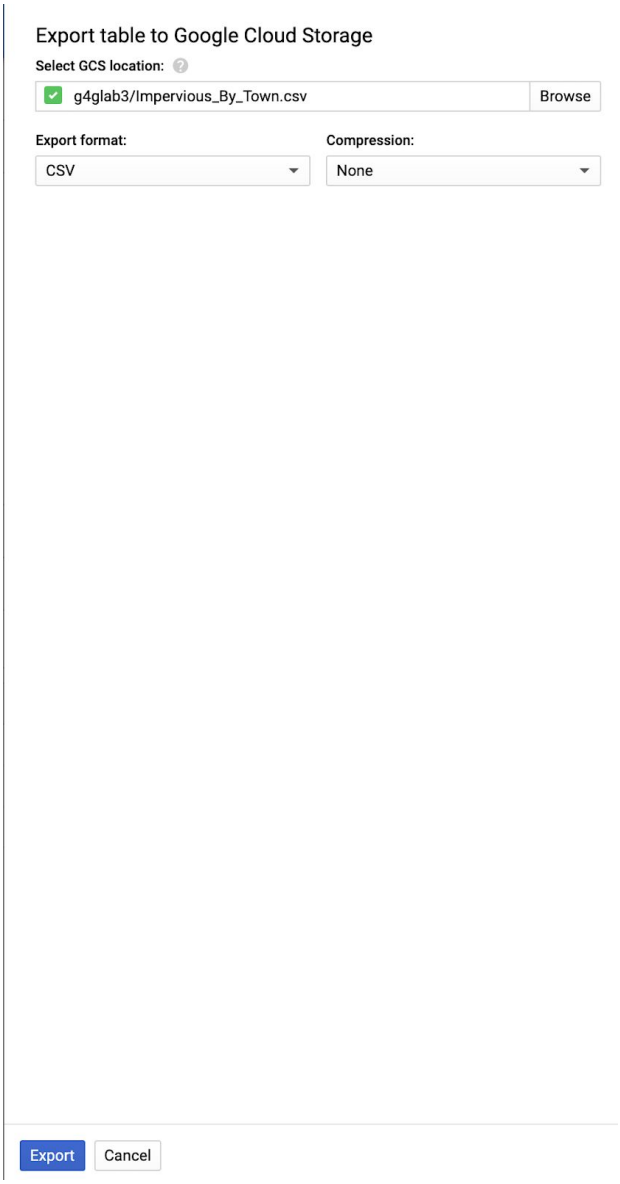


Choose your GCS bucket, and then name the export:

```
Impervious_By_Town.csv
```

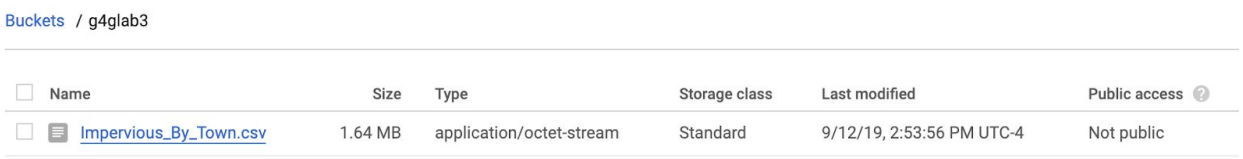


Hit **“Select”** and then hit “**Export”**

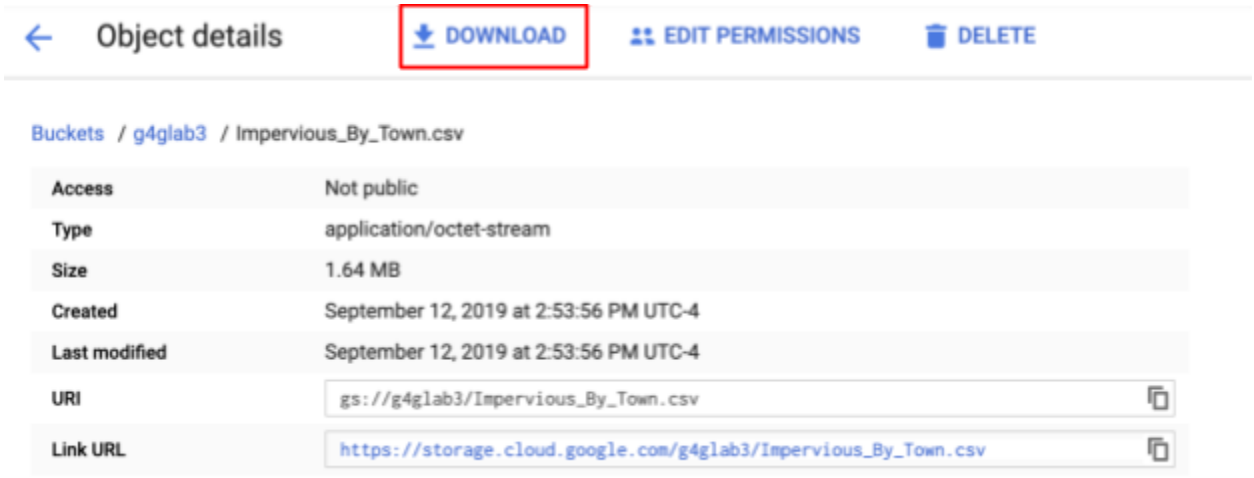


When the job finishes, you'll be shown a notification. Return to Google Cloud Storage, and refresh your Bucket if needed.

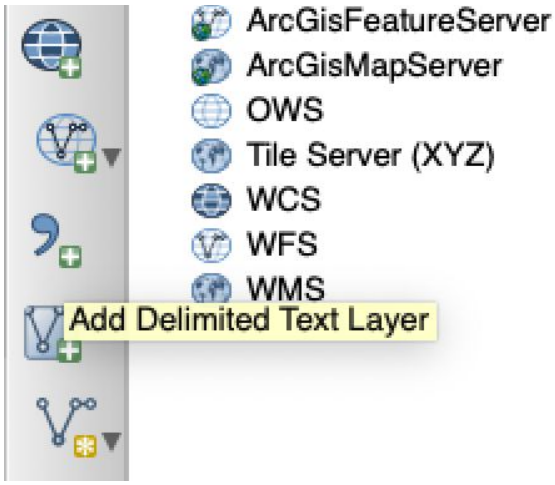
You should now see the CSV file in the bucket.



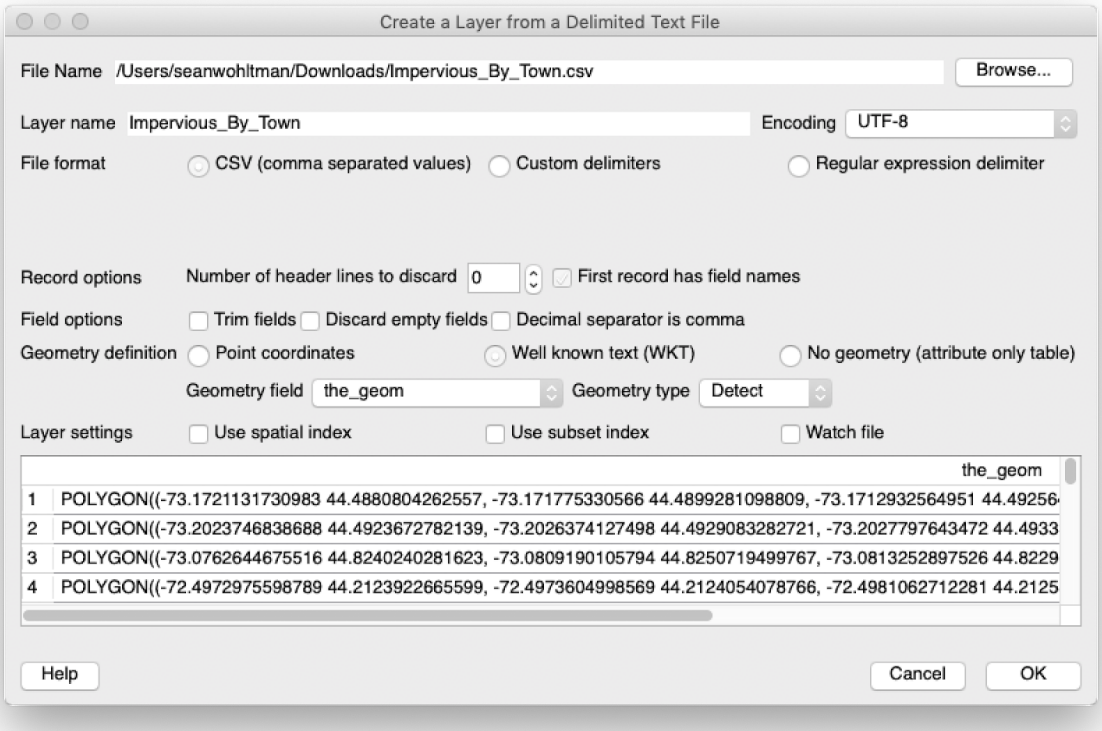
Click on its name to go to the object's detail view. Click "**Download".**



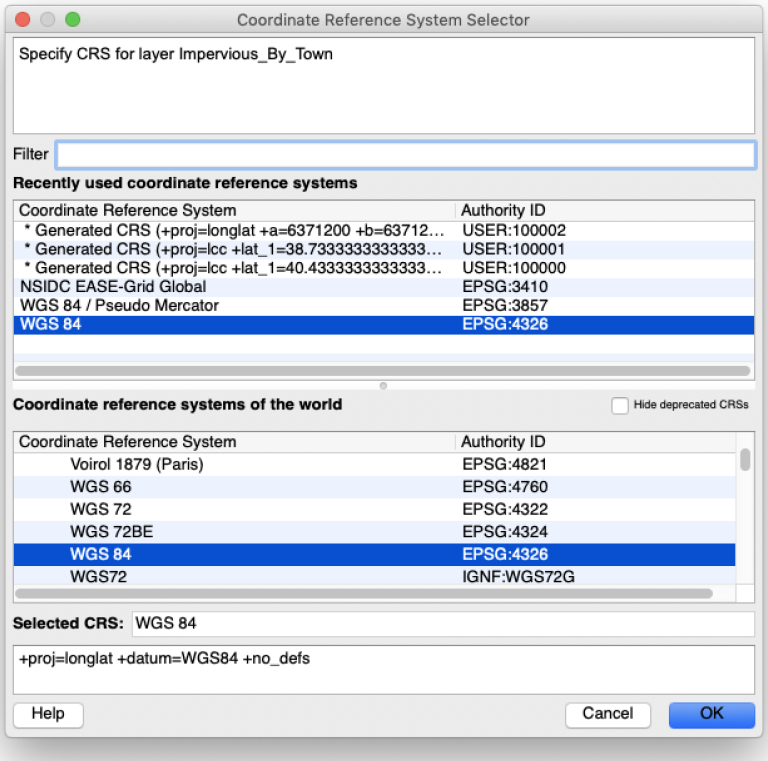
In QGIS 2 or 3, add a new “Text-delimited” file source.



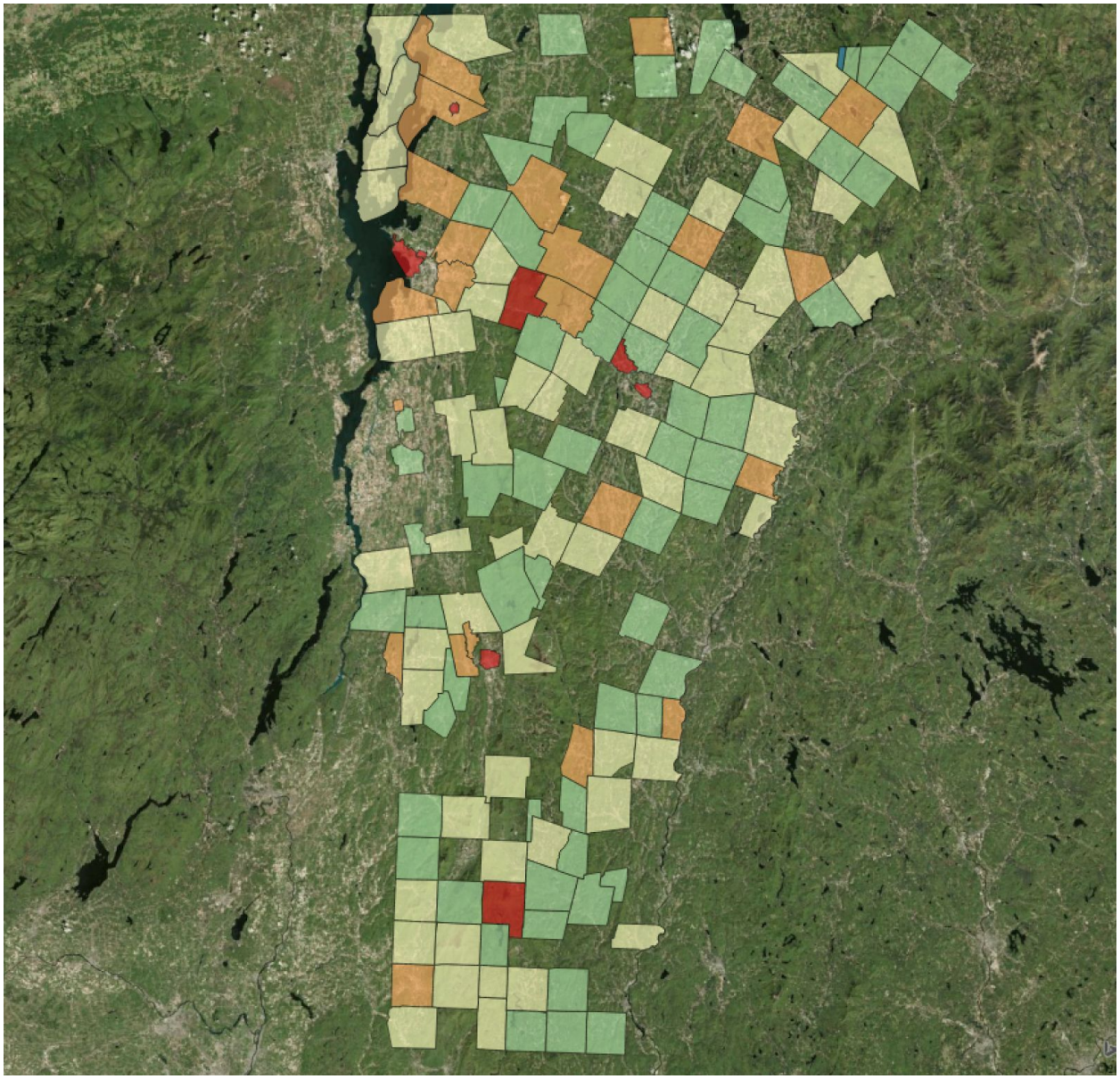
Browse to load the CSV file. QGIS should automatically detect the “the_geom” field and recognize that it is WKT.



However, you will have to specify the SRS since the CSV file doesn't have any metadata. Remember, BQGIS uses EPSG:4326.



When the layer loads, you can style it using QGIS' styling wizard and the "AveragePct" field. Here's an example of using Natural Breaks and a green to blue color ramp.



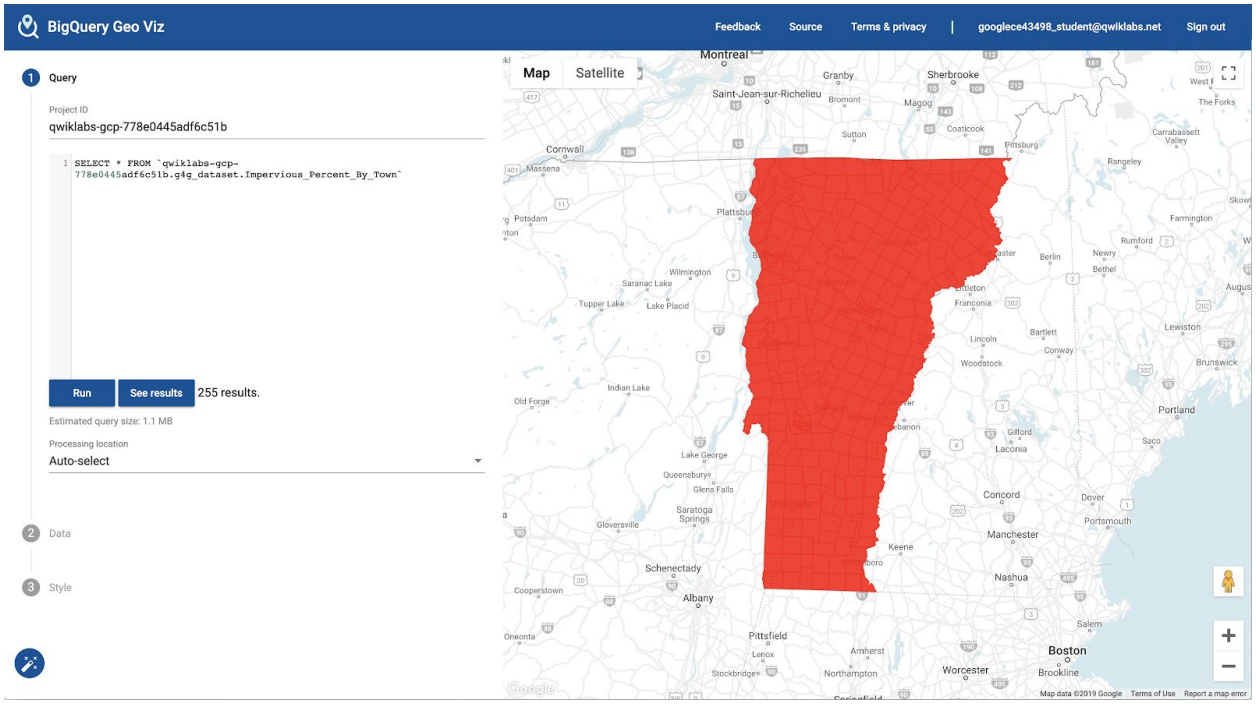
Note: *Missing towns are due to the fact that there are missing parcels in the VT statewide parcel layer* for those towns.

Online (Using BQGeoViz)

In the BigQuery GeoViz tool, clear any existing query and enter the following:

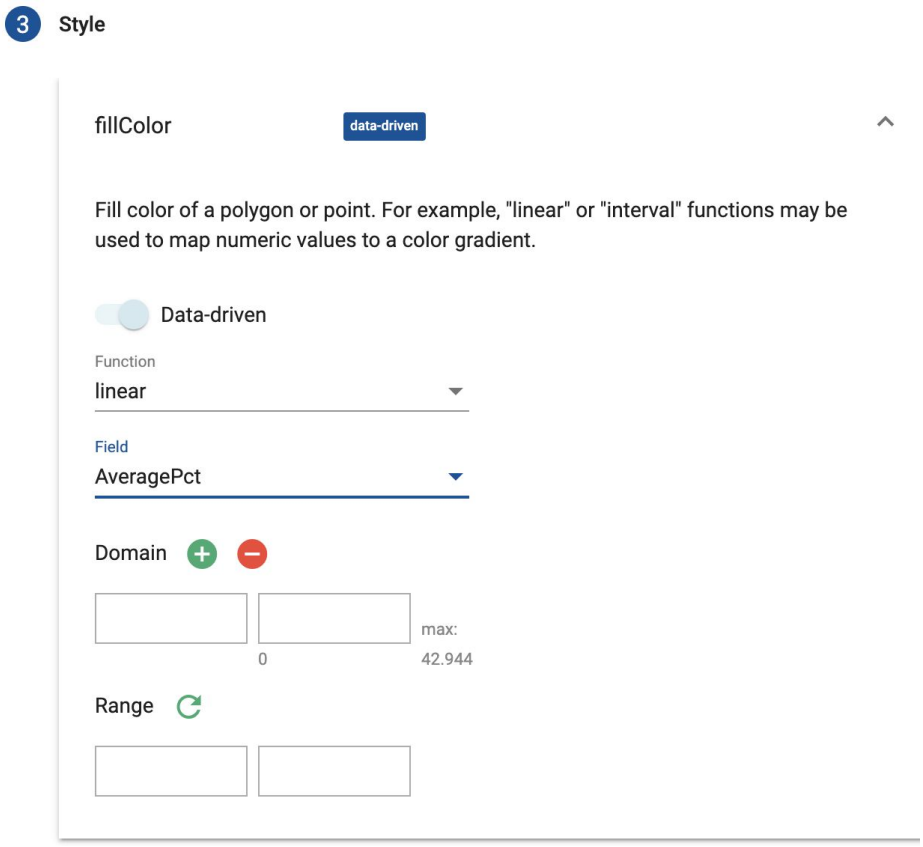
```
SELECT * FROM `g4g_dataset.Impervious_Percent_By_Town`
```

The query should be rather quick since you're just accessing the saved results of the larger query we ran previously. When the query completes, you should see a map of Vermont where all the towns are solid red.



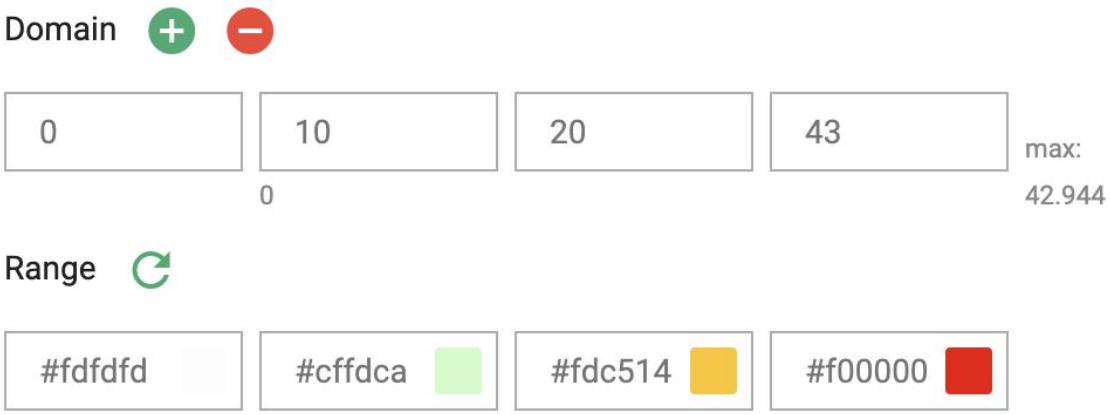
Click the **"Style"** button to start designing a Choropleth Map.

Under **fillColor**, toggle on **Data-driven.**
Under **Function** choose **linear**.
Under **Field** choose **AveragePct**



You'll see the **Domain** min and max values are automatically set to 0 and 42.944.

Click the **Green Plus** button to add 2 more breakpoints, and enter the numbers shown below. Pick any colors you like in the corresponding **Range** boxes.



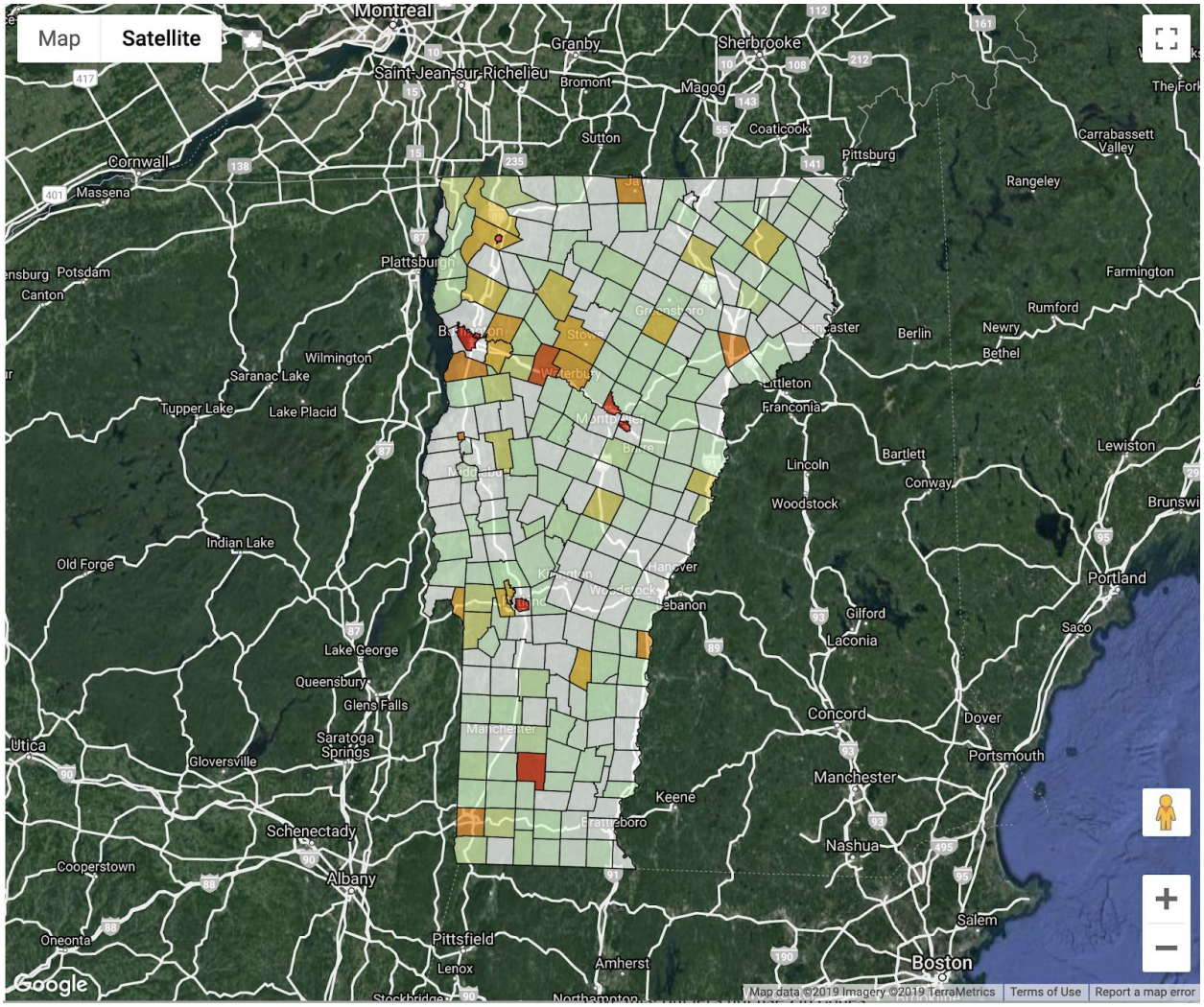
Under **fillOpacity** set it to 80%. Enter:

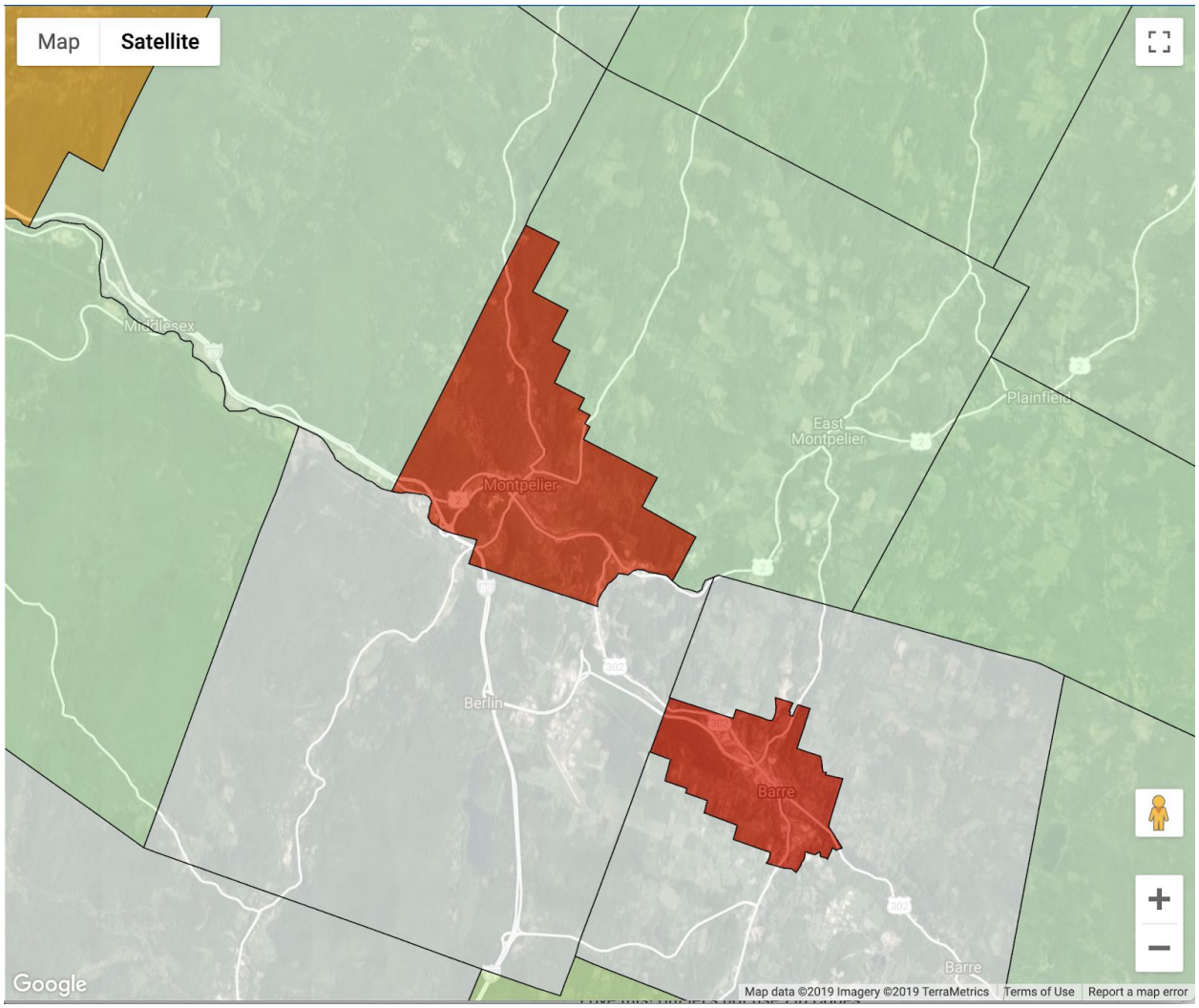
```
.8
```

Under **strokeColor** set it to Black. Enter:

#000000

Switch the Google Maps API map to Satellite view and you can easily view the more urban areas with higher average impervious surface percentages.





[RETURN TO LECTURE]

Section 4 - Finding and Handling Bad Spatial Data

Finding Bad Geometries

BigQuery GIS is very strict about geometry validity. A Shapefile that might appear to open up in QGIS or ArcGIS without incident could actually have one or several invalid geometries. Here are some common reasons why a geometry could be invalid:

No.	Reason why geometry is non-simple	Description
1	Ring orientation is incorrect (applicable only for polygons)	Polygon which does not self-intersect, but its rings are not oriented correctly.
2	Segment orientation is incorrect	Individual segments are not consistently oriented. The 'to' point of segment "i" should be incident on the 'from' point of segment "i+1".
3	Contains short segments	Some segments are shorter than allowed by the spatial reference system units associated with the geometry.
4	Contains self-intersecting planar parts/rings(for polygons/polylines)	The interior of each part (rings, planar parts) intersects with itself or other parts.
5	Contains unclosed rings (applicable only for polygons)	The last segment in a ring does not have its 'to' point incident on the 'from' point of the first segment.
6	Contains empty parts	Geometry contains empty parts.
7	Contains mismatched attributes	Geometry has mismatched attributes.
8	Contains discontinuous parts	Geometry contains discontinuous parts.
9	Empty Z values (applicable to Z-enabled feature classes)	Geometry is Z-aware but contains NaN Zs.
10	Contains duplicate vertices	Geometry has duplicate vertices.

Desktop GIS applications might be able to display these geometries, but when you try to do a spatial query operation with them, you'll either get an error or you'll have to exclude them or fix them. Both QGIS and ArcGIS have tools with varying degrees of success at fixing bad geometries.

To see an example of what happens when you try to import a layer with bad geometry, try the following **bq** command from your virtual machine.

```
bq load --source_format=CSV --skip_leading_rows=1 --replace
--schema=the_geom:GEOGRAPHY,Class_name:STRING
g4g_dataset.impervious_raw gs://vtprepdata/vt_impervious_raw.csv
```

You should quickly see the following output:

```
BigQuery error in load operation: Error processing job
'your-project:bqjob_r6acbed77d5d5f8d3_0000016d2b072b43_1': Error
while reading data, error message: CSV table encountered too many errors,
giving up. Rows: 59; errors: 1. Please
look into the errors[] collection for more details.
```

```
Failure details:
- gs://vtprepdata/vt_impervious_raw.csv: Error while reading data,
error message: Could not parse '{"type":"Polygon","coordinates":[[[
-73.18498419182351,44.58137866268577],[-73...' as geography for
field the_geom (position 0) starting at location 1936405946 Invalid
polygon loop: Edge 6 has duplicate vertex with edge 9
- gs://vtprepdata/vt_impervious_raw.csv: Error while reading data,
error message: Could not parse '{"type":"Polygon","coordinates":[[[
-73.040167812682,43.71586668211814],[-73.0...' as geography for
field the_geom (position 0) starting at location 484127147 Invalid
polygon loop: Edge 308 has duplicate vertex with edge 340
```

As you can see, the BQ load job failed because there were too many errors. In the details, you can see that there were many the_geom fields that had geometry errors.

The bq load tool does have the ability to raise the threshold for the number of permitted errors, but, this is not recommended.

You could add "*--max_bad_records=X*" to the **bq load** command, and the job will finish, but it will simply drop the erroneous fields.

[RETURN TO LECTURE]

To understand the extent of the problem, you can try to load the csv but change the schema so that you're loading the_geom field as a type STRING instead of a type GEOGRAPHY.

E.g. --schema=the_geom:**STRING**,Class_name:STRING

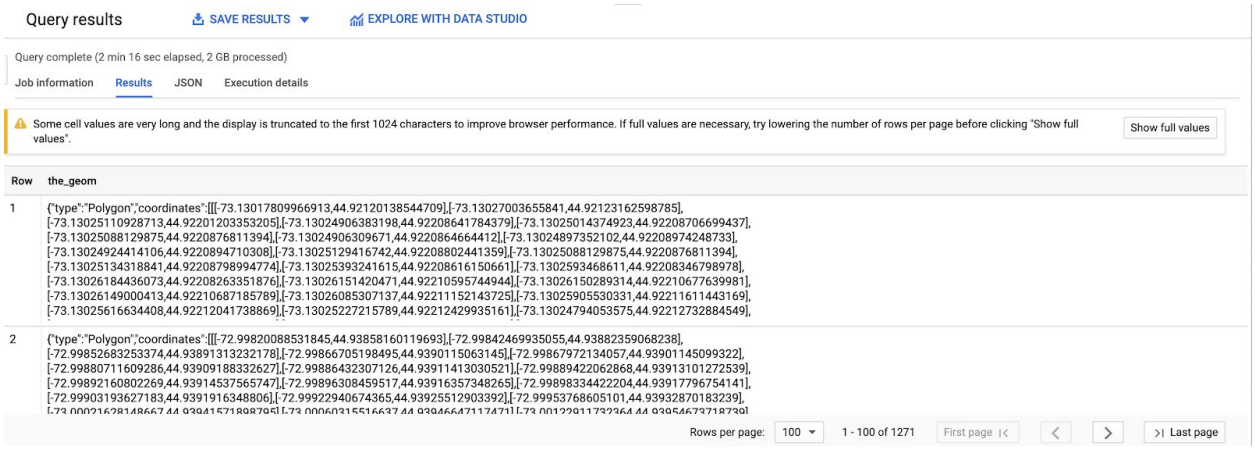
Then, we can examine the table by using the SAFE function prefix to output the problematic data. Using SAFE when running one of the BQ GIS spatial functions will create a NULL result for any fields that are invalid.

To save time, since this is a large file, we'll use an existing version of the imported table. Since we ingested the table with the the_geom field set as a string, that field is just a GeoJSON string. However, it can be converted to a GEOGRAPHY type on the fly using the **ST_GeogFromGeoJson** function.

In the BigQuery UI, run he following query which will output a list of all the rows that have invalid geometries.

```
SELECT
  *
FROM
  `vt-solar.vtdata.impervious_text`
WHERE
  the_geom IS NOT NULL
  AND SAFE.ST_GeogFromGeoJson(the_geom) IS NULL
```

This should take around 2 minutes to run, and you should see ~1,200 rows with errors!



Fixing Bad Geometries

As previously mentioned, there are some utilities in Desktop GIS applications to find and fix invalid geometries, but they can also be addressed in other spatial databases such as spatialite and PostGIS.

The *most straightforward* way to fix geometries is to leverage the spatialite sql dialect when you're initially creating the csv file from your shapefile or other vector datasource.

For example, rather than simply:

```
"ogr2ogr -f csv -dialect sqlite -sql "select
AsGeoJSON(Transform(geometry,4326)) the_geom, * from
VT_Data__Town_Boundaries" impervious.csv VT_Impervious.shp"
```

We could try:

```
"ogr2ogr -f csv -dialect sqlite -sql "select
AsGeoJSON(Transform(ST_MakeValid(geometry),4326)) the_geom, * from
VT_Data__Town_Boundaries" impervious.csv VT_Impervious.shp"
```

This would first correct the geometry, then transform it to EPSG:4326, and then finally convert the geometry to GeoJSON format.

However, as of the time of writing this lab, the gdal binaries for Debian and Ubuntu images on Google Cloud Platform are not compiled with the rttopo library support in the spatialite library used to build ogr2ogr.

Another way to fix geometries is within PostGIS, a very common spatial database. Geometry columns in tables can either be updated in place using the ST_MakeValid function, or, when you're exporting a table from PostGIS to csv using the \copy command, you could run ST_MakeValid in that query.

Importing first to PostGIS and then exporting to CSV can be a lot of overhead for one-off imports, but if your workflow already involves writing data to PostGIS and you're hoping to use BQ GIS for these very large spatial queries, cleaning data in PostGIS first and then loading to BQ GIS can just be part of your standard and automated workflow.

[RETURN TO LECTURE]

Section 5 - Utilizing Public Datasets and the BQ GIS Sandbox

Finally, while we used custom spatial data in this lab, and a Qwiklab GCP Project environment, we wanted to point out both the BigQuery Public Datasets and the BQ GIS Sandbox.

BigQuery Public Datasets

Google maintains a large amount of public datasets on Google Cloud platform that are of high interest to many and frequent use. Here are some useful links to review.

Directions for use in BigQuery
BigQuery Public Dataset Landing Page

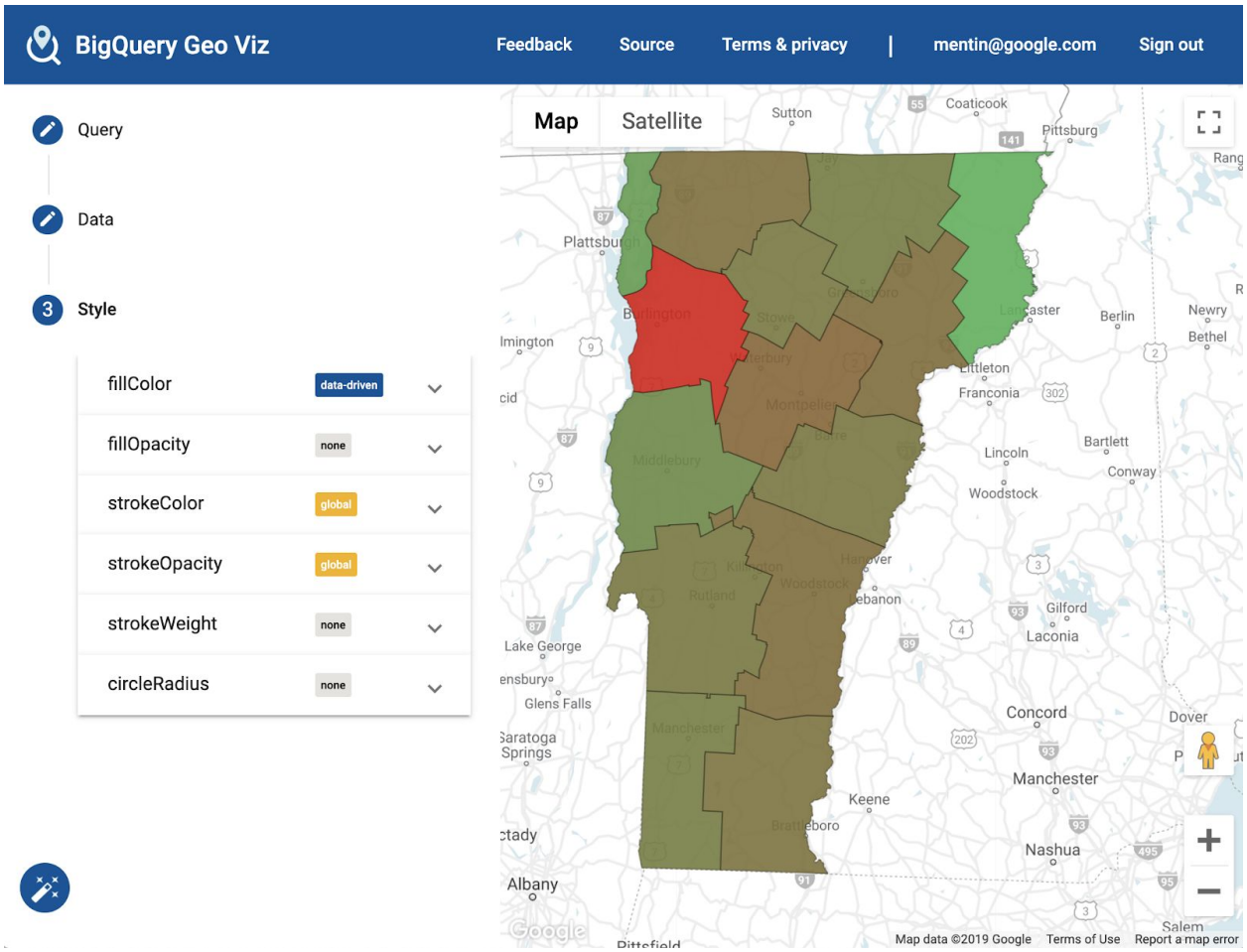
In this example, we'll use the Counties table from the US Census BQ GIS Dataset

In BigQuery GeoViz's query editor, enter the following query:

```
WITH county_impervious_area AS (
  SELECT c.geo_id,
```

```
          ANY_VALUE(c.county_geom) AS geom,
          Sum(ST_Area(ST_Intersection(im.the_geom, c.county_geom)))
AS impervious_area
  FROM `vt-solar.vtdata.vtImpervious` im
  JOIN `bigquery-public-data.geo_us_boundaries.us_counties` c
  ON ST_Intersects(im.the_geom, c.county_geom)
  WHERE c.state_name = 'Vermont'
  GROUP BY c.geo_id
)
SELECT *, impervious_area / ST_Area(geom) as impervious_pct
FROM county_impervious_area
```

This will calculate the impervious area per county in Vermont using the custom VT impervious surface data per county in the public US Census Counties table. Then, style the map.



[RETURN TO LECTURE]

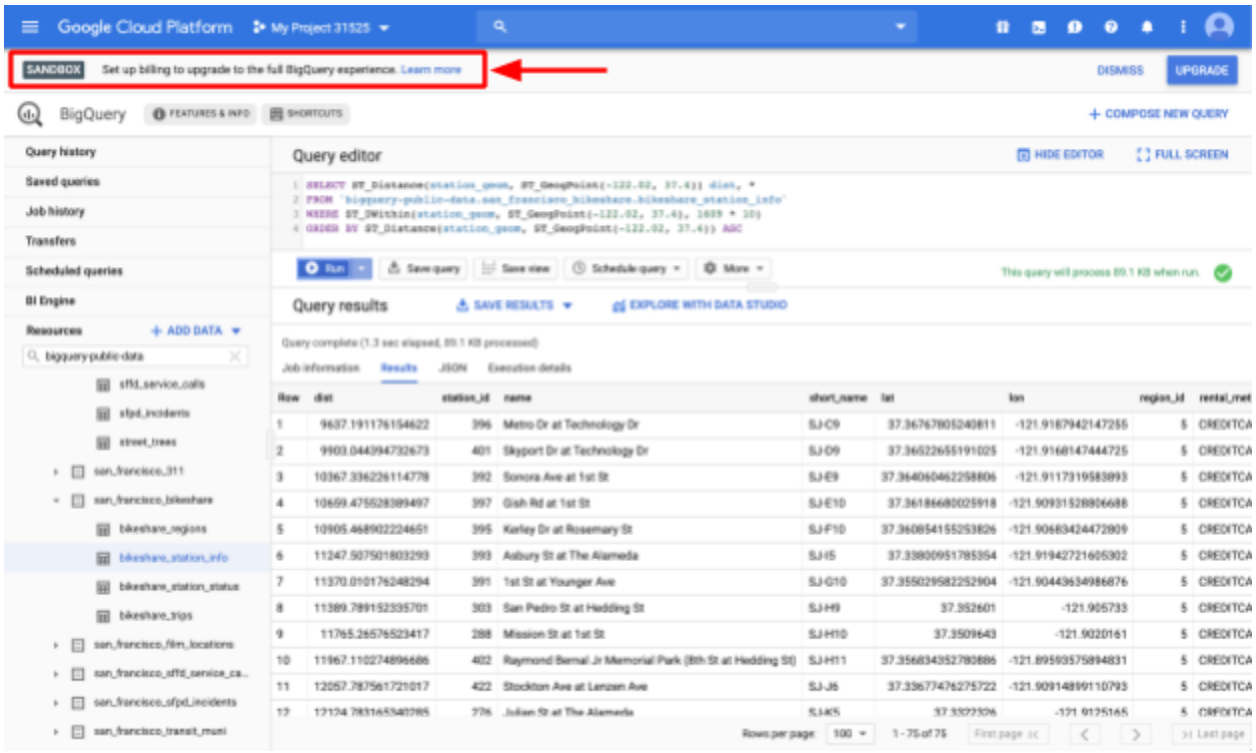
BigQuery Sandbox

Finally, after you leave her today, if you want to continue to work with BigQuery GIS, but you don't have a GCP billing account or can't add a credit card in your organization, you can still use the BigQuery Sandbox.

It is a Free Tier of BigQuery, available to everyone. You can store up to 10GB of data, and query up to 1TB of data (either user or shared data) free. It offers the full BigQuery performance as well.

To get started, visit the Google Cloud Console with your normal Gmail or G Suite account, and create a new project. If you do not have a Billing Account, you can still access BigQuery.

In the BigQuery UI the only difference you will notice is that there is a header bar that indicates you're using the BigQuery Sandbox.



Congratulations!

You have completed this BigQuery GIS Qwicklab and should now feel comfortable leveraging Google Compute Engine, Google Cloud Storage, and BigQuery GIS to quickly conduct large geospatial analyses!

Manual Last Updated September 13, 2019

Lab Last Tested September 13, 2019