Visualizing BigQuery data in a Jupyter notebook

BigQuery is a petabyte-scale analytics data warehouse that you can use to run SQL queries over vast amounts of data in near realtime.

Data visualization tools can help you make sense of your BigQuery data and help you analyze the data interactively. You can use visualization tools to help you identify trends, respond to them, and make predictions using your data. In this tutorial, you use the BigQuery client library for Python and Pandas in a Jupyter notebook to visualize data in the BigQuery natality sample table.

Objectives

In this tutorial you:

- Set up an environment to run Jupyter notebooks
- Query and visualize BigQuery data using BigQuery Python client library and Pandas

Costs

BigQuery is a paid product and you incur BigQuery usage costs when accessing BigQuery. BigQuery query pricing provides the first 1 TB per month free of charge. For more information, see the BigQuery Pricing page.

Before you begin

Before you begin this tutorial, use the Google Cloud Console to create or select a project.

1. If you're new to Google Cloud, create an account to evaluate how our products perform in real-world scenarios. New customers also get $300 in free credits to run, test, and deploy workloads.
2. In the Google Cloud Console, on the project selector page, select or create a Google Cloud project.

   Note: If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

   GO TO PROJECT SELECTOR

3. BigQuery is automatically enabled in new projects. To activate BigQuery in a preexisting project, go to Enable the BigQuery API.

   ENABLE THE API

4. Optional: Enable billing for the project. If you don't want to enable billing or provide a credit card, the steps in this document still work. BigQuery provides a sandbox to perform the steps.

   To learn more about the BigQuery sandbox, including limitations, how to add a billing account and upgrade your project, and troubleshooting, see About BigQuery sandbox.

   Note: If your project has a billing account and you want to use the BigQuery sandbox, disable billing for your project.

Setting up a local Jupyter environment

In this tutorial, you use a locally hosted Jupyter notebook. Complete the following steps to install Jupyter and set up authentication.

1. Follow the installation instructions in the Jupyter documentation to install Jupyter.

2. Follow the instructions in the Getting started with authentication page to set up Application Default Credentials. You set up authentication by creating a service account and setting an environment variable.

Install the client libraries

Install the BigQuery Python client library version 1.9.0 or higher and the BigQuery Storage API Python client library.

PIPConda

Install the google-cloud-bigquery and google-cloud-bigquery-storage packages.

```
pip install --upgrade 'google-cloud-bigquery[bqstorage,pandas]'
```

Overview: Jupyter notebooks

A notebook provides an environment in which to author and execute code. A notebook is essentially a source artifact, saved as an .ipynb file. It can contain descriptive text content, executable code blocks, and associated results (rendered as interactive HTML). Structurally, a notebook is a sequence of cells.

A cell is a block of input text that is evaluated to produce results. Cells can be of two types:

- *Code cells*: contain code to evaluate. Any outputs or results from executing the code are rendered immediately below the input code.
- *Markdown cells*: contain markdown text that is converted to HTML to produce headers, lists, and formatted text.

The following screenshot shows a markdown cell followed by a Python code cell. Note that the output of the Python cell is shown immediately below the code.

Each opened notebook is associated with a running session. In Python, this is also referred to as a kernel. This session executes all the code entered within the notebook, and manages the state (variables, their values, functions and classes, and any existing Python modules you load).

Querying and visualizing BigQuery data

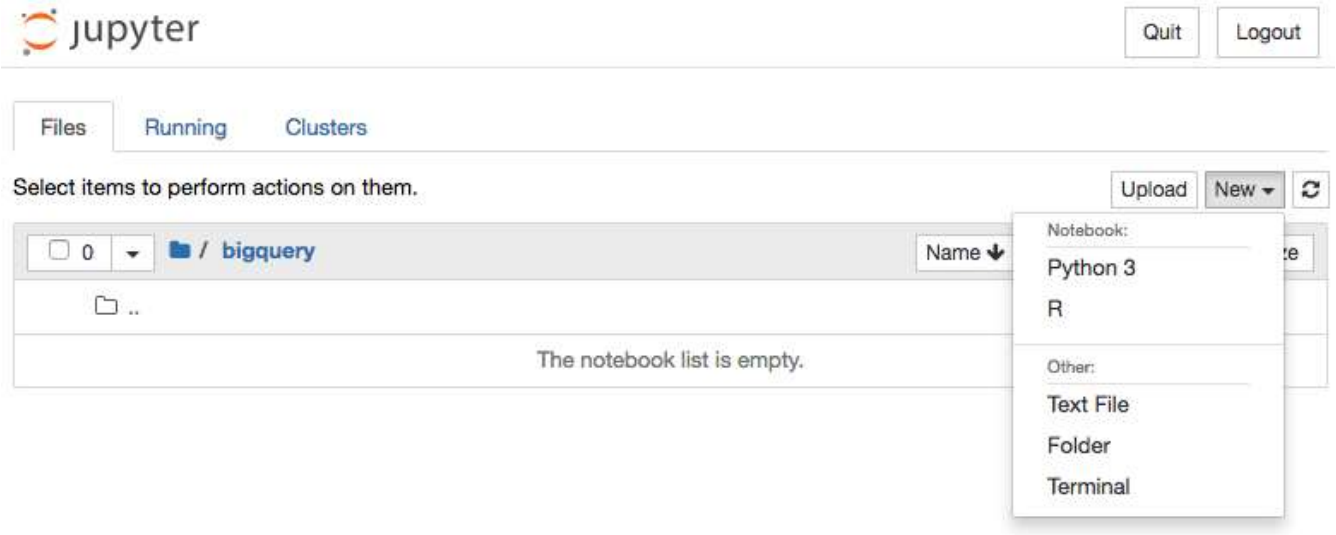
In this section of the tutorial, you create a Datalab notebook used to query and visualize data in BigQuery. You create visualizations using the data in the natality sample table. All queries in this tutorial are in standard SQL syntax.

To query and visualize BigQuery data using a Jupyter notebook:

1. If you haven't already started Jupyter, run the following command in your terminal:

```
jupyter notebook
```

2. Jupyter should now be running and open in a browser window. In the Jupyter window, click the **New** button and select **Python 3** to create a Python notebook.



3. At the top of the page, click **Untitled**.
4. In the **Rename notebook** dialog, type a new name such as BigQuery tutorial, and then click **Rename**.
5. The BigQuery client library for Python provides a magic command that lets you run queries with minimal code. To load the magic commands from the client library, paste the following code into the first cell of the notebook.

```
%load_ext google.cloud.bigquery
```

Note: `%load_ext` is one of the many Jupyter built-in magic commands. See the Jupyter documentation for more information about `%load_ext` and other built-in magic commands.

6. Run the command by clicking the **Run** button or with SHIFT + ENTER.
7. The BigQuery client library for Python provides a cell magic, %%bigquery, which runs a SQL query and returns the results as a Pandas DataFrame. Enter the following code in the next cell to return total births by year:

```
%%bigquery
SELECT
    source_year AS year,
    COUNT(is_male) AS birth_count
FROM `bigquery-public-data.samples.natality`
GROUP BY year
ORDER BY year DESC
LIMIT 15
```

8. Click **Run**.
9. The query results appear below the code cell.

Out[2]:

	year	birth_count
0	2008	4255156
1	2007	4324008
2	2006	4273225
3	2005	4145619
4	2004	4118907
5	2003	4096092
6	2002	4027376
7	2001	4031531
8	2000	4063823
9	1999	3963465
10	1998	3945192
11	1997	3884329
12	1996	3894874
13	1995	3903012
14	1994	3956925

10. In the next cell block, enter the following command to run the same query, but this time save the results to a new variable total_births, which is given as an argument to %%bigquery. The results can then be used for further analysis and visualization.

```
%%bigquery total_births
SELECT
    source_year AS year,
    COUNT(is_male) AS birth_count
FROM `bigquery-public-data.samples.natality`
GROUP BY year
ORDER BY year DESC
LIMIT 15
```

Note: See the client library magics documentation to read more about the possible arguments for **`%%bigquery`**.

11. Click **Run**.
12. Now you have a Pandas DataFrame saved to variable total_births, which is ready to plot. To prepare for plotting the query results, paste the following built-in magic command in the next cell to activate matplotlib. Matplotlib is the library used by Pandas for plotting.

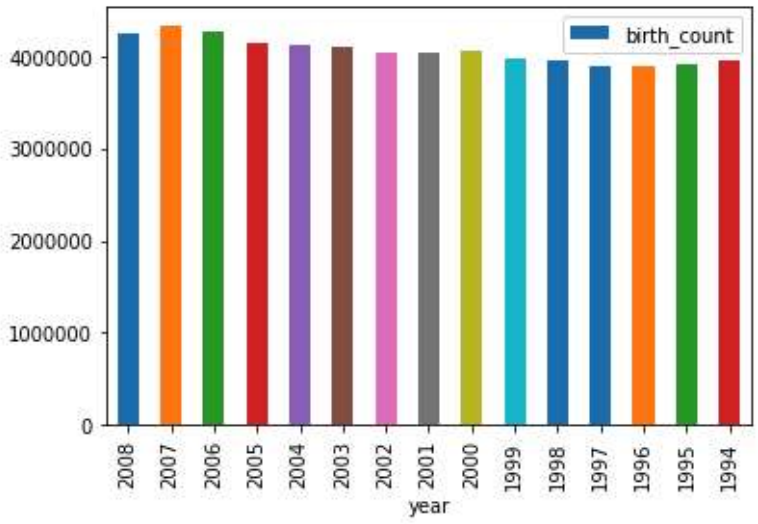
```
%matplotlib inline
```

13. Click **Run**.
14. In the next cell, enter the following code to use the Pandas DataFrame.plot() method to visualize the query results as a bar chart. See the Pandas documentation to learn more about data visualization with Pandas.

```
total_births.plot(kind="bar", x="year", y="birth_count")
```

Note: Semicolons are not normally used in Python, but in Jupyter notebooks, they can be used to suppress the output of the function. Try running the command with and without the semicolon to see the difference in output.

15. Click **Run**.
16. The chart appears below the code block.



17. Next, paste the following query into the next cell to retrieve the number of births by weekday.

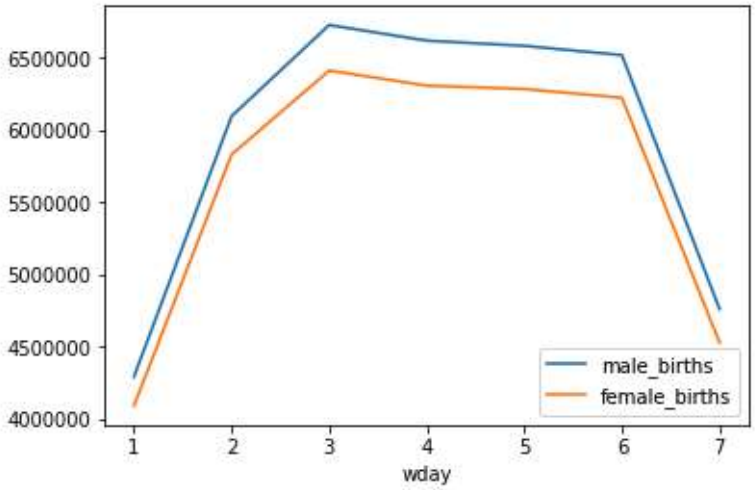
```
%%bigquery births_by_weekday
SELECT
    wday,
    SUM(CASE WHEN is_male THEN 1 ELSE 0 END) AS male_births,
    SUM(CASE WHEN is_male THEN 0 ELSE 1 END) AS female_births
FROM `bigquery-public-data.samples.natality`
WHERE wday IS NOT NULL
GROUP BY wday
ORDER BY wday ASC
```

Because the wday (weekday) field allows null values, the query excludes records where wday is null.

18. Click **Run**.
19. In the next cell, enter the following code to visualize the query results using a line chart.

```
births_by_weekday.plot(x="wday")
```

20. Click **Run**.
21. The chart appears below the code block. Notice the number of births dramatically decreases on Sunday (1) and Saturday (7).



22. Click **File > Save and Checkpoint** or click the save icon in the toolbar. Creating a checkpoint lets you roll the notebook back to a previous state.

Pandas DataFrames

Magic commands let you use minimal syntax to interact with BigQuery. Behind the scenes, %%bigquery uses the BigQuery client library for Python to run the given query, convert the results to a Pandas Dataframe, optionally save the results to a variable, and finally display the results. Using the BigQuery client library for Python directly, instead of through magic commands, gives you more control over your queries and lets you use more complex configurations. The library's integrations with Pandas enable you to combine the power of declarative SQL with imperative code (Python) to perform interesting data analysis, visualization, and transformation tasks.

Note: You can use a number of Python data analysis, data wrangling, and visualization libraries, such as **`numpy`**, **`pandas`**, **`matplotlib`**, and many others. Several of these libraries build on top of a DataFrame object.

Querying and visualizing BigQuery data using pandas DataFrames

In this section of the tutorial, you query and visualize data in BigQuery using pandas DataFrames. You use the BigQuery client library for Python to query BigQuery data. You use the Pandas library to analyze data using DataFrames.

1. Type the following Python code into the next cell to import the BigQuery client library for Python and initialize a client. The BigQuery client is used to send and receive messages from the BigQuery API.

```
from google.cloud import bigquery

client = bigquery.Client()
```

2. Click **Run**.
3. Use the Client.query() method to run a query. In the next cell, enter the following code to run a query to retrieve the annual count of plural births by plurality (2 for twins, 3 for triplets, and so on).

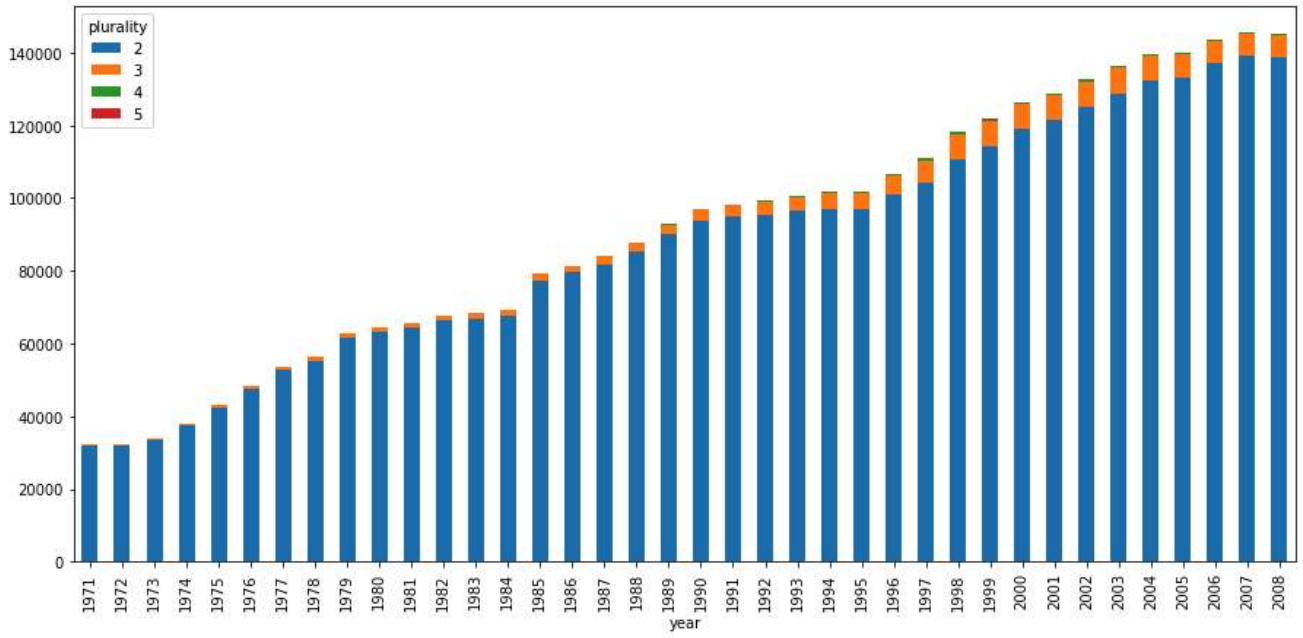
```
sql = """
SELECT
    plurality,
    COUNT(1) AS count,
    year
FROM
    `bigquery-public-data.samples.natality`
WHERE
```

```
    NOT IS_NAN(plurality) AND plurality > 1
GROUP BY
    plurality, year
ORDER BY
    count DESC
"""
df = client.query(sql).to_dataframe()
df.head()
```

4. Click **Run**.
5. To chart the query results in your DataFrame, insert the following code into the next cell to pivot the data and create a stacked bar chart of the count of plural births over time.

```
pivot_table = df.pivot(index="year", columns="plurality", values="count")
pivot_table.plot(kind="bar", stacked=True, figsize=(15, 7))
```

6. Click **Run**.
7. The chart appears below the code block.



8. In the next cell, enter the following query to retrieve the count of births by the number of gestation weeks.

```
sql = """
SELECT
    gestation_weeks,
    COUNT(1) AS count
FROM
    `bigquery-public-data.samples.natality`
WHERE
    NOT IS_NAN(gestation_weeks) AND gestation_weeks <> 99
GROUP BY
    gestation_weeks
ORDER BY
    gestation_weeks
"""
df = client.query(sql).to_dataframe()
```

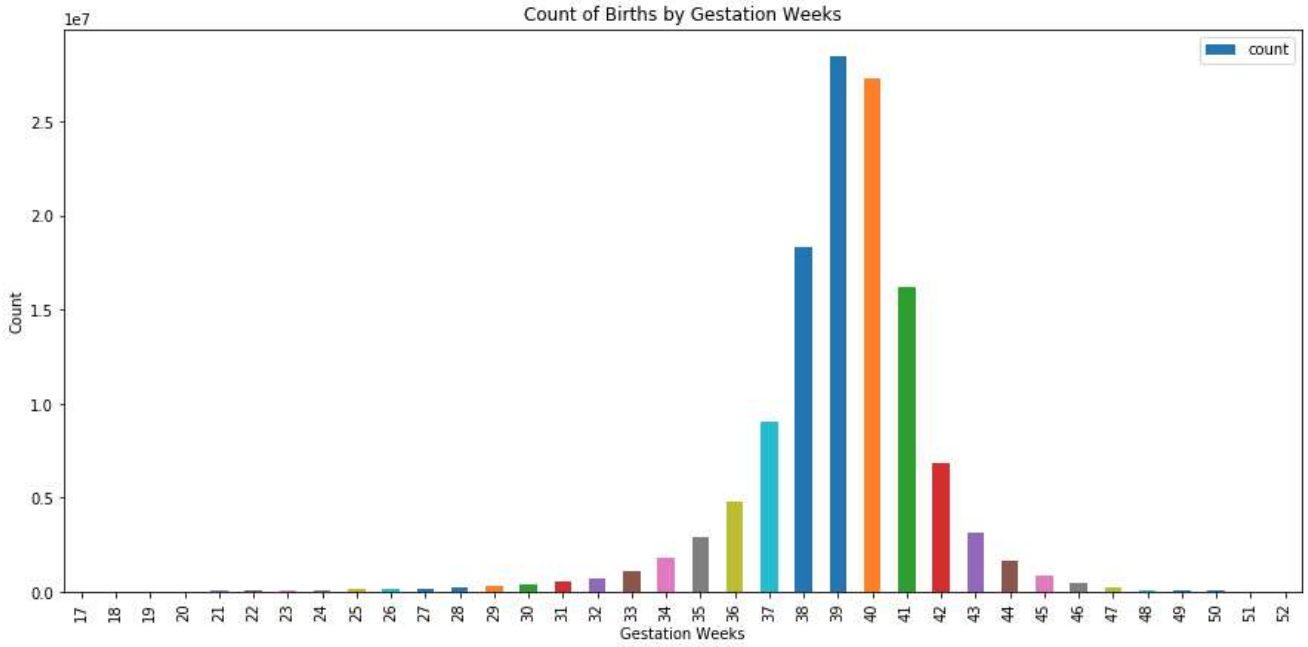
Note: Because the gestation_weeks field allows null values and stores unknown values as 99, this query excludes records where **`gestation_weeks`** is null or 99.

9. Click **Run**.
10. To chart the query results in your DataFrame, paste the following code in the next cell.

```
ax = df.plot(kind="bar", x="gestation_weeks", y="count", figsize=(15, 7))
ax.set_title("Count of Births by Gestation Weeks")
ax.set_xlabel("Gestation Weeks")
ax.set_ylabel("Count")
```

Note: This code plots the data as a bar chart and sets a chart title, x-axis label, and y-axis label. There are many other options you can specify when plotting data. For more information, see **`pandas.DataFrame.plot`** in the pandas API reference.

11. Click **Run**.
12. The bar chart appears below the code block.



What's next

- **Learn more about writing queries for BigQuery:** Querying data in the BigQuery documentation explains how to run queries create user-defined functions (UDFs), and more.
- **Explore BigQuery syntax:** the preferred dialect for SQL queries in BigQuery is standard SQL, which is described in the SQL reference. BigQuery's legacy SQL-like syntax is described in the Query reference (legacy SQL).