BigQuery for Data Analysis

Liz Izhikevich

What is Google BigQuery?

“BigQuery is a fully-managed, serverless data warehouse that enables scalable analysis over petabytes of data. “

What is Google BigQuery?

“BigQuery is a fully-managed, serverless data warehouse that enables scalable analysis over petabytes of data. “

Liz’s definition: SQL* in the cloud

What is Google BigQuery?

“BigQuery is a fully-managed, serverless data warehouse that enables scalable analysis over **petabytes** of data. “

Liz’s definition: SQL* in the cloud

- Censys IPv4 Daily scan -> **728GB**
- Censys IPv4 Banner scan -> **1 TB**
- 1% IPv4 LZR scan -> **1 TB**
- 1% IPv4 LZR scan filtered only real hosts -> **30GB**

I highly recommend using BigQuery!

- Pros:
  - Data processing is so fast
    - Easy to explore the data across different axes

What are the top 100 most used ports ?

What are the top 100 ports used in networks that are owned by the government?

I highly recommend using BigQuery!

- Pros:
  - Data processing is so fast
    - Easy to explore the data across different axes
  - Data is organized in one place

I highly recommend using BigQuery!

- Pros:
    - Data processing is so fast
        - Easy to explore the data across different axes
    - Data is organized in one place
- Cons:
    - Doesn’t support statistics beyond the very basics*

I highly recommend using BigQuery!

- Pros:
  - Data processing is so fast
    - Easy to explore the data across different axes
  - Data is organized in one place
- Cons:
  - Doesn’t support statistics beyond the very basics*

Javascript User Defined Functions (“UDFs”) _can_ help

I highly recommend using BigQuery!

- Pros:
  - Data processing is so fast
    - Easy to explore the data across different axes
  - Data is organized in one place
- Cons:
  - Doesn’t support statistics beyond the very basics*
  - Interface for saving queries is poor...its just a list…. Although can use collab notebooks
  - Can’t *easily* plot/visualize the data in BigQuery interface
  - Learning curve

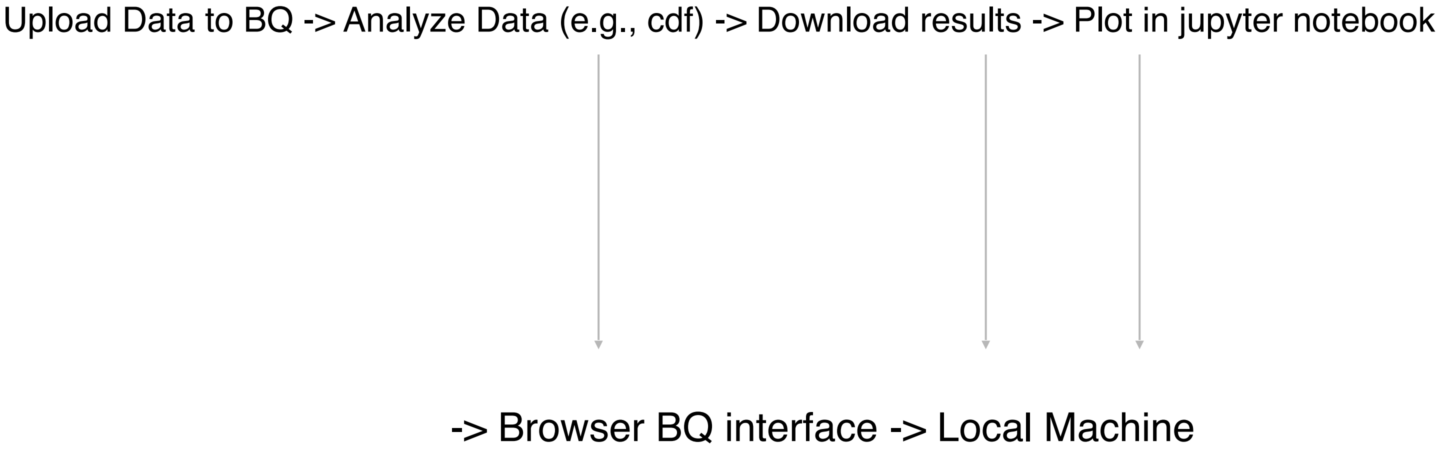
I highly recommend using BigQuery!

- Pros:
  - Data processing is so fast
    - Easy to explore the data across different axes
  - Data is organized in one place
- Cons:
  - Doesn’t support statistics beyond the very basics*
  - Interface for saving queries is poor
  - Can’t *easily* plot/visualize the data in BigQuery interface
  - Learning curve

(+) Can see patterns better

(-) Less inclined to do math-y things

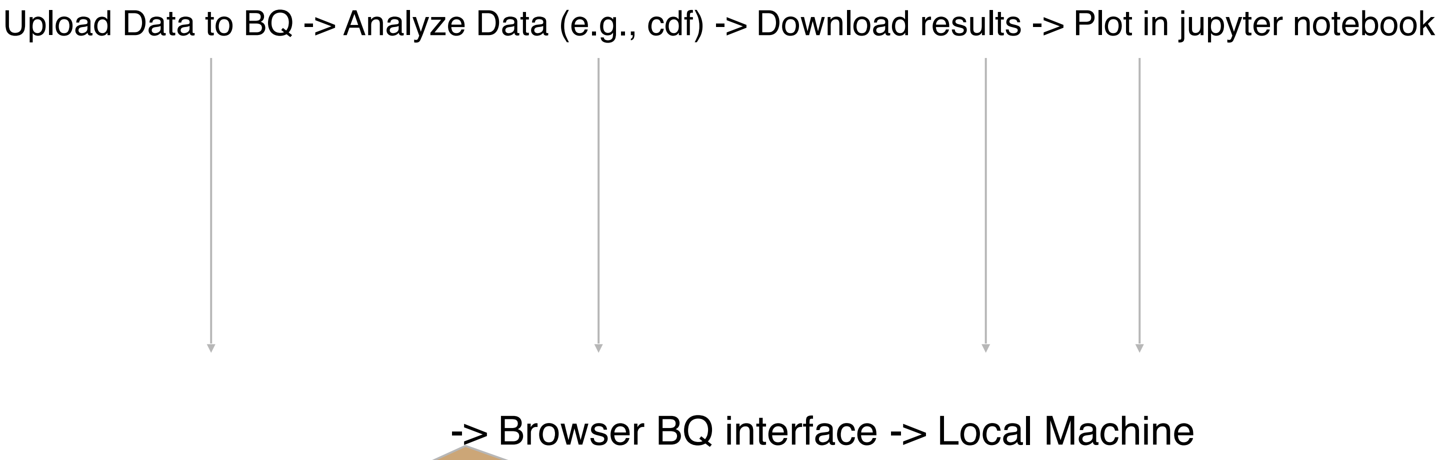
Liz’s BigQuery data analysis pipeline



Upload Data to BQ -> Analyze Data (e.g., cdf) -> Download results -> Plot in jupyter notebook

-> Browser BQ interface -> Local Machine

Liz’s BigQuery data analysis pipeline



Upload Data to BQ -> Analyze Data (e.g., cdf) -> Download results -> Plot in jupyter notebook

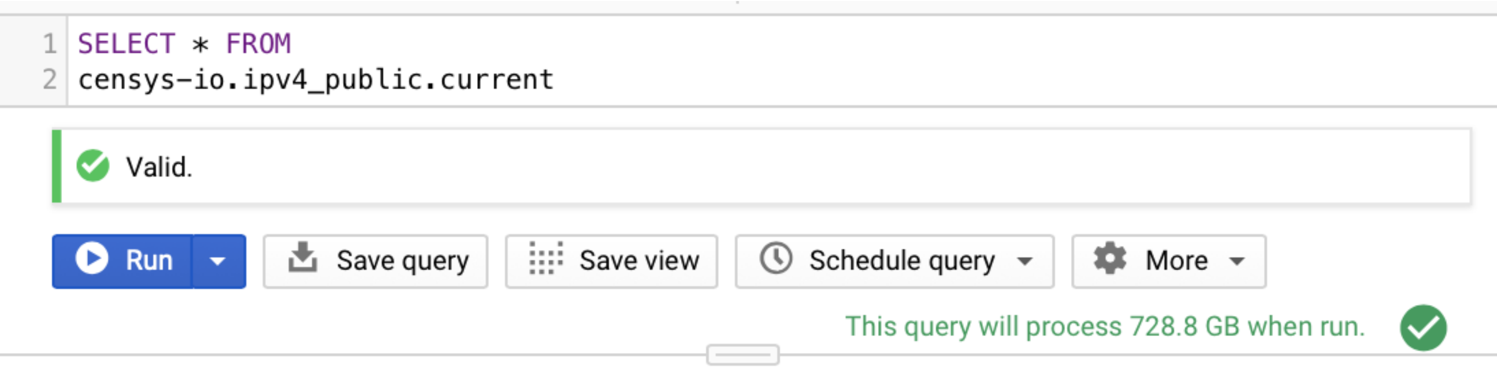
-> Browser BQ interface -> Local Machine

All steps can be done through jupyter notebook...but i prefer working with the BQ interface

- Offers interactive debugging
- Preview of tables
- Preview of cost

BigQuery can be expensive...

- $6 / Terabyte processed



```
SELECT * FROM
censys-io.ipv4_public.current
```

Valid.

This query will process 728.8 GB when run.

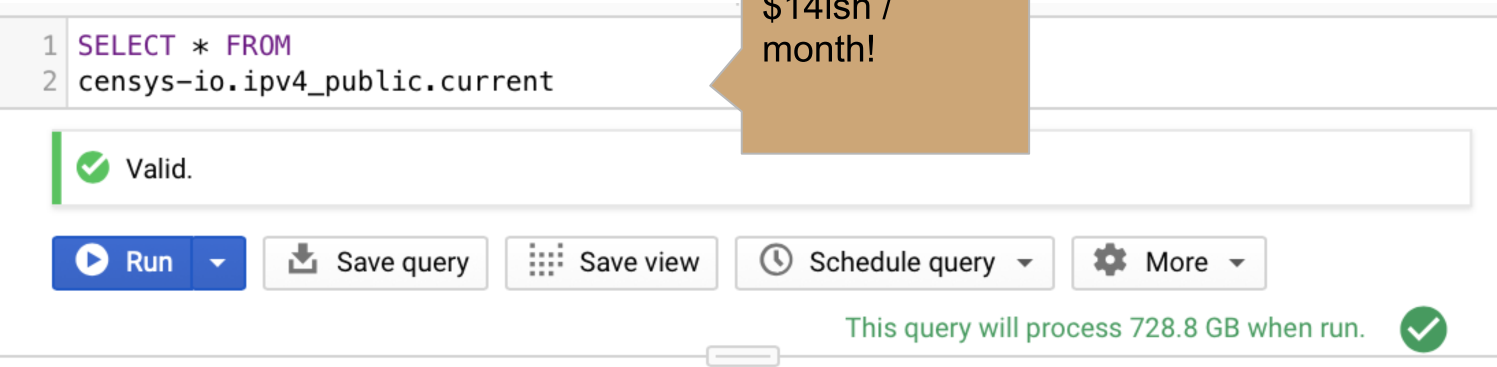
Best practices for processing (to avoid $$$$$)

- Query only specific columns (avoid SELECT *)
- Filter by only relevant dates
  - If querying over a range of dates, first test that your script works for a small range
- Check the amount of data BQ says it will process; does that seem reasonable?

- If you only need a sub-sample of data to test with, save the sub-sample as a new table and query that

BigQuery can be expensive...

- $0.02 / per GB / month storage



```
SELECT * FROM
censys-io.ipv4_public.current
```

Valid.

This query will process 728.8 GB when run.

$14ish / month!

Best practices for storage (to avoid $$$$$)

- Delete tables when you no longer need them!

Pipeline Step 1: Upload data to BQ

Uploading data to BigQuery

```
bq load --autodetect --max_bad_records 100 --source_format
NEWLINE_DELIMITED_JSON ${datasetName}.${tableName} $filename
```

Uploading data to BigQuery

autodetect
the schema

```
bq load --autodetect --max_bad_records 100 --source_format
NEWLINE_DELIMITED_JSON ${datasetName}.${tableName} $filename
```

Uploading data to BigQuery

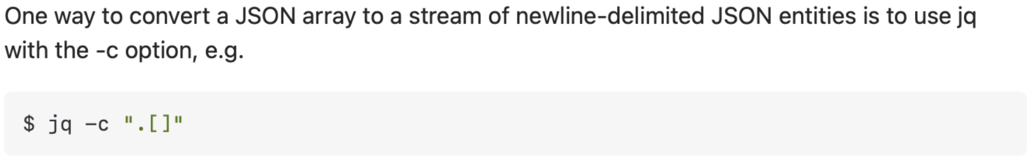
In case some records are malformed

```
bq load --autodetect --max_bad_records 100 --source_format
NEWLINE_DELIMITED_JSON ${datasetName}.${tableName} $filename
```

Uploading data to BigQuery

```
bq load --autodetect --max_bad_records 100 --source_format NEWLINE_DELIMITED_JSON ${datasetName}.${tableName} $filename
```

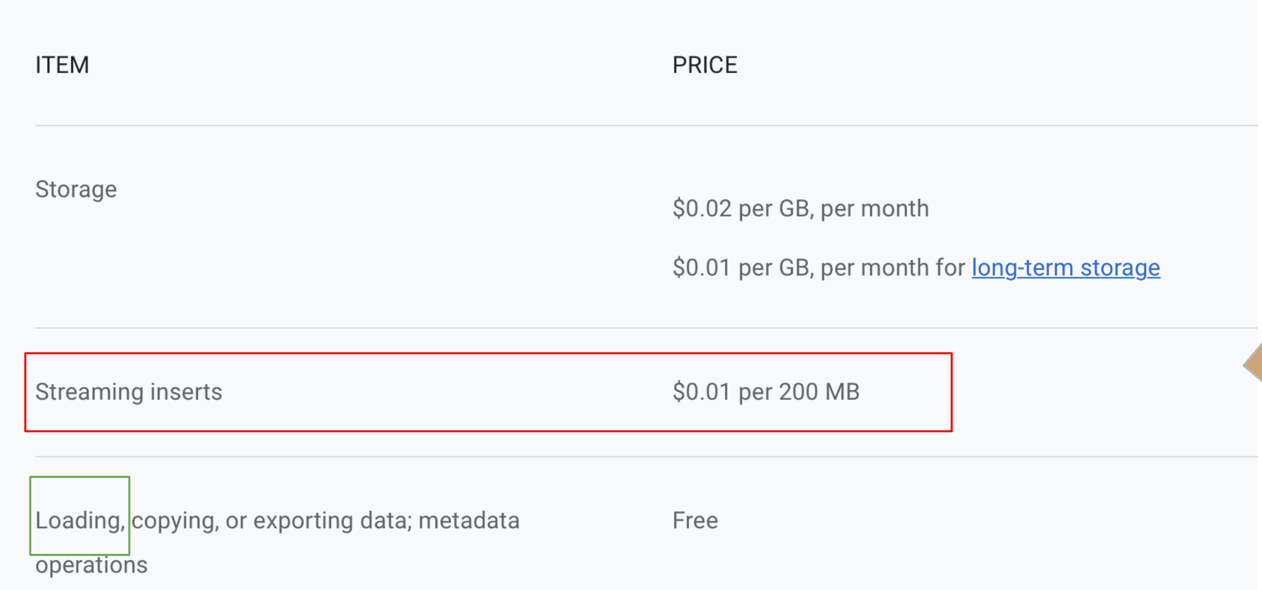
CSV or JSON



One way to convert a JSON array to a stream of newline-delimited JSON entities is to use jq with the -c option, e.g.

```
$ jq -c ".[]"
```

BigQuery can be expensive...



ITEM	PRICE
Storage	$0.02 per GB, per month $0.01 per GB, per month for long-term storage
Streaming inserts	$0.01 per 200 MB
Loading, copying, or exporting data; metadata operations	Free

Don’t use streaming inserts to upload data to big uery!

Pipeline Step 2: Analyze data in BQ

BigQuery case study

Dataset: 20min sample of darknet traffic provided by Greynoise

DO KNOW EVIL

GreyNoise tells security analysts what not to worry about.

Greynoise set up “honeypots/sensors” in different cloud providers to classify who (e.g., Stanford, China, etc) is scanning what

Greynoise dataset features we will use

`metadata.org` (scanner) - Censys, etc

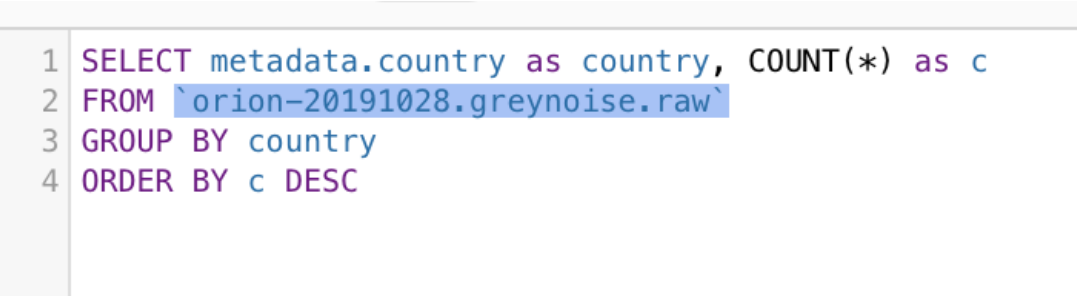
`metadata.country` (scanner)

`destination_port`

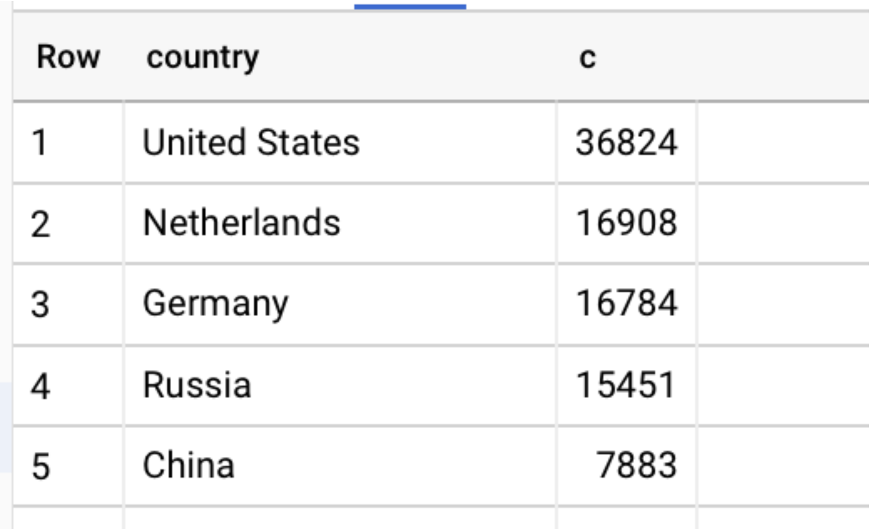
`provider` (sensor) - AWS, Digital Ocean, etc

`sensor_metadata.country` (sensor)

Which countries scan the most?

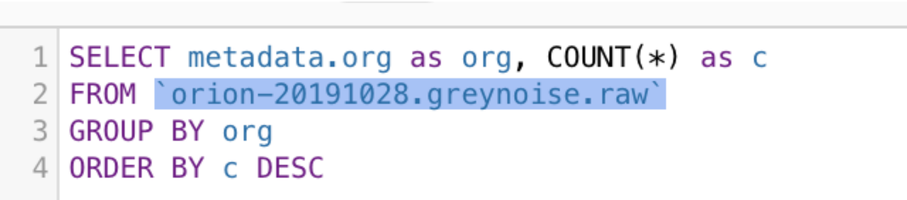


```sql
SELECT metadata.country as country, COUNT(*) as c
FROM `orion-20191028.greynoise.raw`
GROUP BY country
ORDER BY c DESC
```

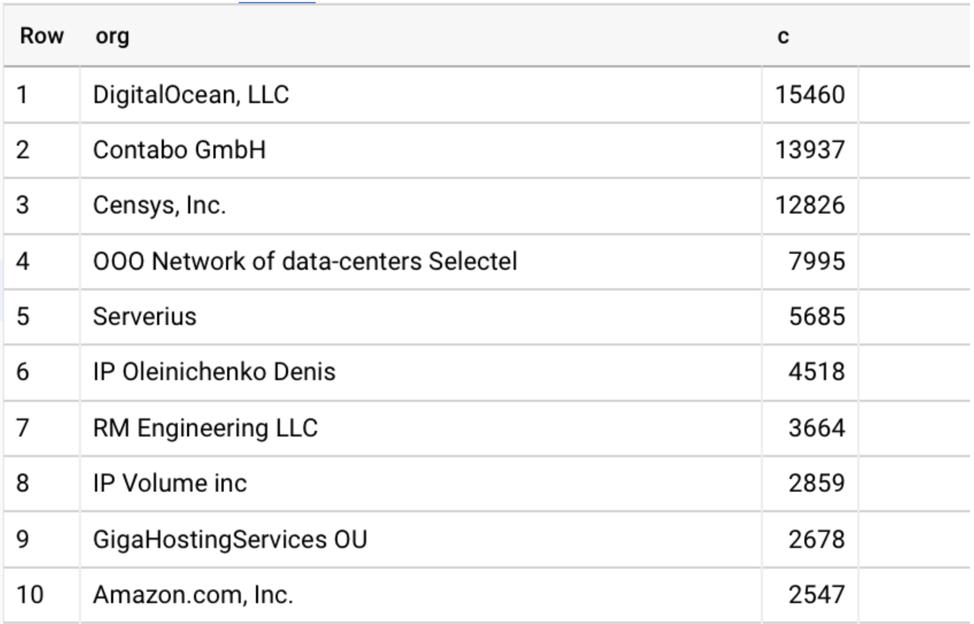


Row	country	c
1	United States	36824
2	Netherlands	16908
3	Germany	16784
4	Russia	15451
5	China	7883

Which organizations scan the most?

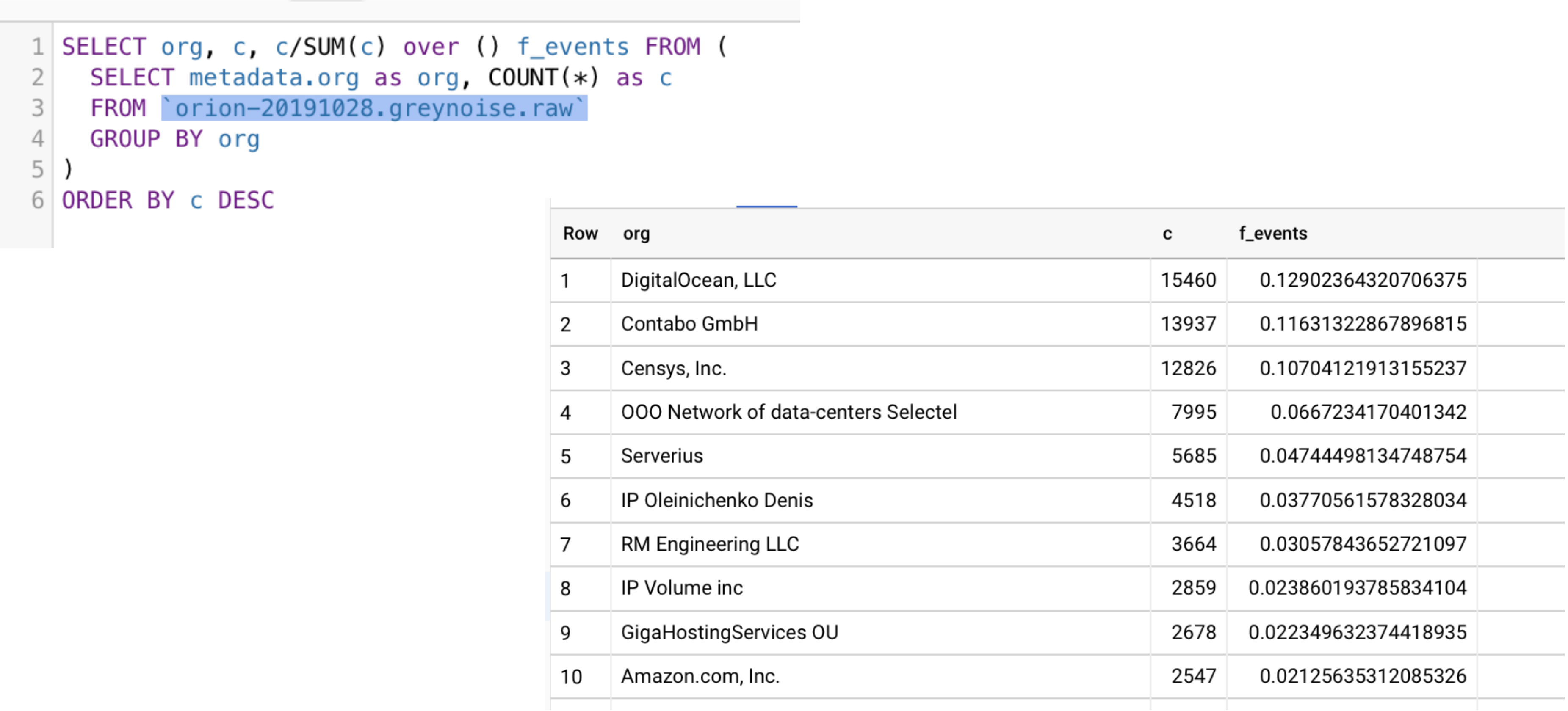


```sql
SELECT metadata.org as org, COUNT(*) as c
FROM `orion-20191028.greynoise.raw`
GROUP BY org
ORDER BY c DESC
```



Row	org	c
1	DigitalOcean, LLC	15460
2	Contabo GmbH	13937
3	Censys, Inc.	12826
4	OOO Network of data-centers Selectel	7995
5	Serverius	5685
6	IP Oleinichenko Denis	4518
7	RM Engineering LLC	3664
8	IP Volume inc	2859
9	GigaHostingServices OU	2678
10	Amazon.com, Inc.	2547

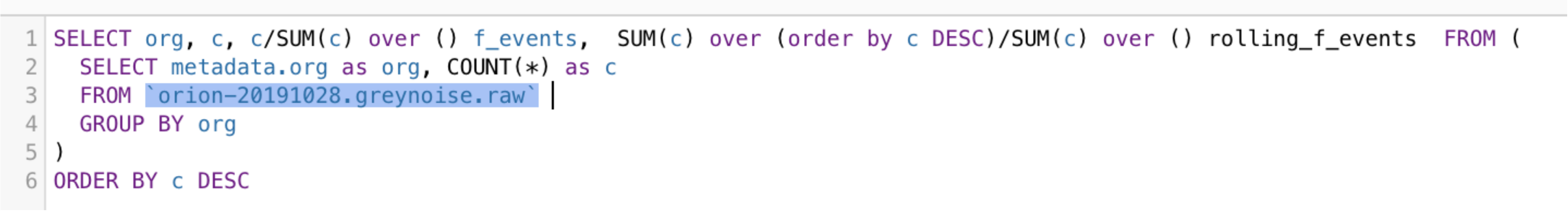
What fraction of all received traffic do individual organizations send?



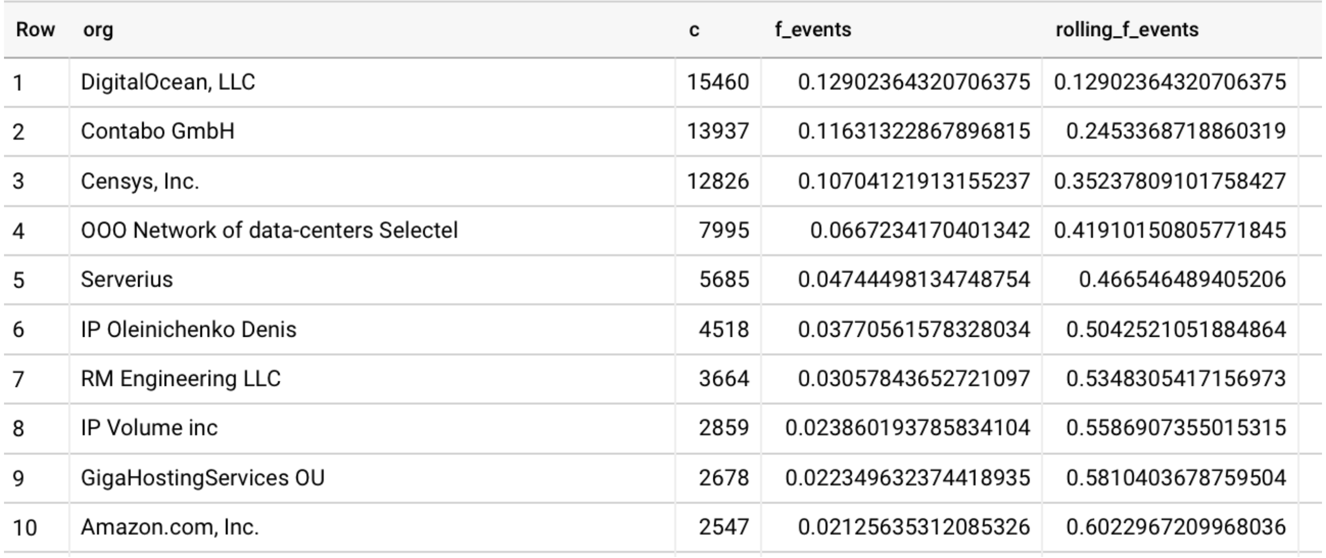
```sql
SELECT org, c, c/SUM(c) over () f_events FROM (
  SELECT metadata.org as org, COUNT(*) as c
  FROM `orion-20191028.greynoise.raw`
  GROUP BY org
)
ORDER BY c DESC
```

Row	org	c	f_events
1	DigitalOcean, LLC	15460	0.12902364320706375
2	Contabo GmbH	13937	0.11631322867896815
3	Censys, Inc.	12826	0.10704121913155237
4	OOO Network of data-centers Selectel	7995	0.0667234170401342
5	Serverius	5685	0.04744498134748754
6	IP Oleinichenko Denis	4518	0.03770561578328034
7	RM Engineering LLC	3664	0.03057843652721097
8	IP Volume inc	2859	0.023860193785834104
9	GigaHostingServices OU	2678	0.022349632374418935
10	Amazon.com, Inc.	2547	0.02125635312085326

The top 10 scanning organizations send what % of all received traffic?

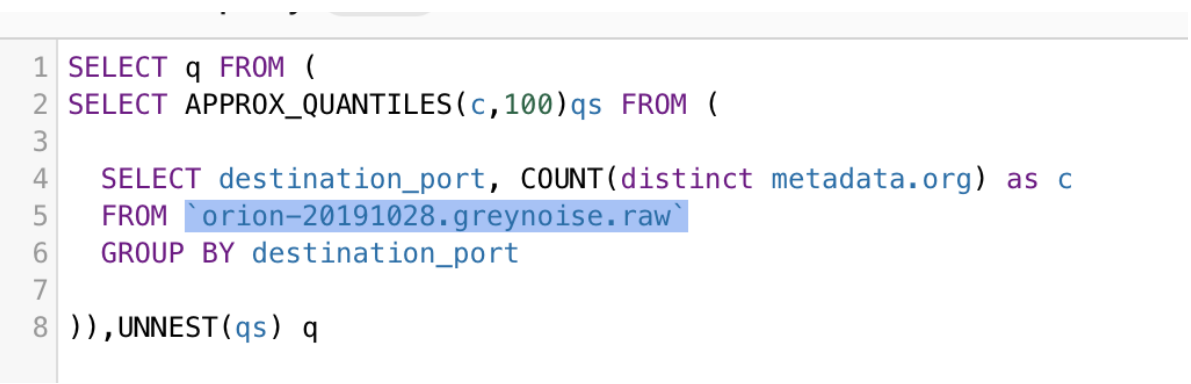


```
SELECT org, c, c/SUM(c) over () f_events,  SUM(c) over (order by c DESC)/SUM(c) over () rolling_f_events  FROM (
  SELECT metadata.org as org, COUNT(*) as c
  FROM `orion-20191028.greynoise.raw`
  GROUP BY org
)
ORDER BY c DESC
```

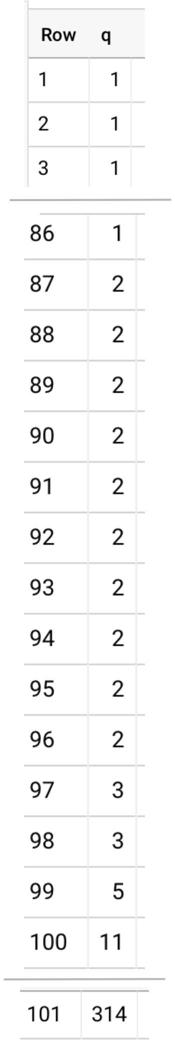


Row	org	c	f_events	rolling_f_events
1	DigitalOcean, LLC	15460	0.12902364320706375	0.12902364320706375
2	Contabo GmbH	13937	0.11631322867896815	0.2453368718860319
3	Censys, Inc.	12826	0.10704121913155237	0.35237809101758427
4	OOO Network of data-centers Selectel	7995	0.0667234170401342	0.41910150805771845
5	Serverius	5685	0.04744498134748754	0.466546489405206
6	IP Oleinichenko Denis	4518	0.03770561578328034	0.5042521051884864
7	RM Engineering LLC	3664	0.03057843652721097	0.5348305417156973
8	IP Volume inc	2859	0.023860193785834104	0.5586907355015315
9	GigaHostingServices OU	2678	0.022349632374418935	0.5810403678759504
10	Amazon.com, Inc.	2547	0.02125635312085326	0.6022967209968036

How many unique organizations scan each port?

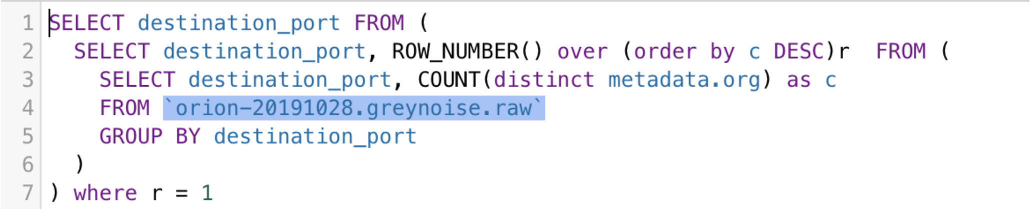


```
SELECT q FROM (
SELECT APPROX_QUANTILES(c,100)qs FROM (

  SELECT destination_port, COUNT(distinct metadata.org) as c
  FROM `orion-20191028.greynoise.raw`
  GROUP BY destination_port

)),UNNEST(qs) q
```

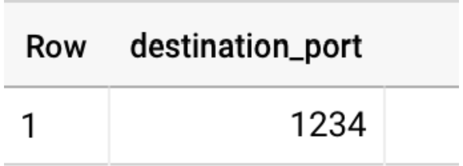


Row	q
1	1
2	1
3	1
86	1
87	2
88	2
89	2
90	2
91	2
92	2
93	2
94	2
95	2
96	2
97	3
98	3
99	5
100	11
101	314

What port is scanned by the most number of unique organizations?



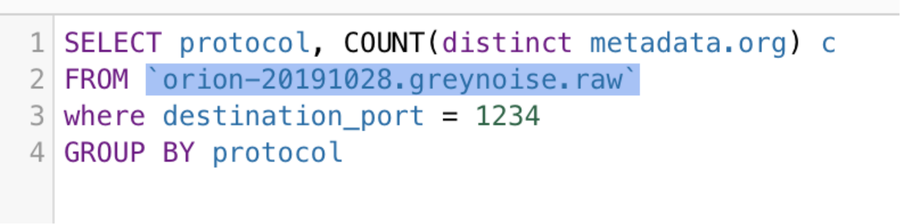
```
SELECT destination_port FROM (
  SELECT destination_port, ROW_NUMBER() over (order by c DESC)r  FROM (
    SELECT destination_port, COUNT(distinct metadata.org) as c
    FROM `orion-20191028.greynoise.raw`
    GROUP BY destination_port
  )
) where r = 1
```



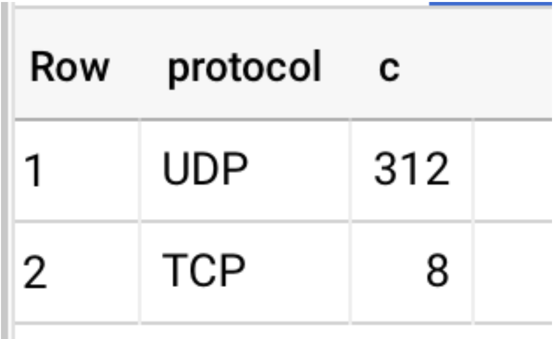
Row	destination_port
1	1234

lol!

What are protocols are being scanned for on port 1234?

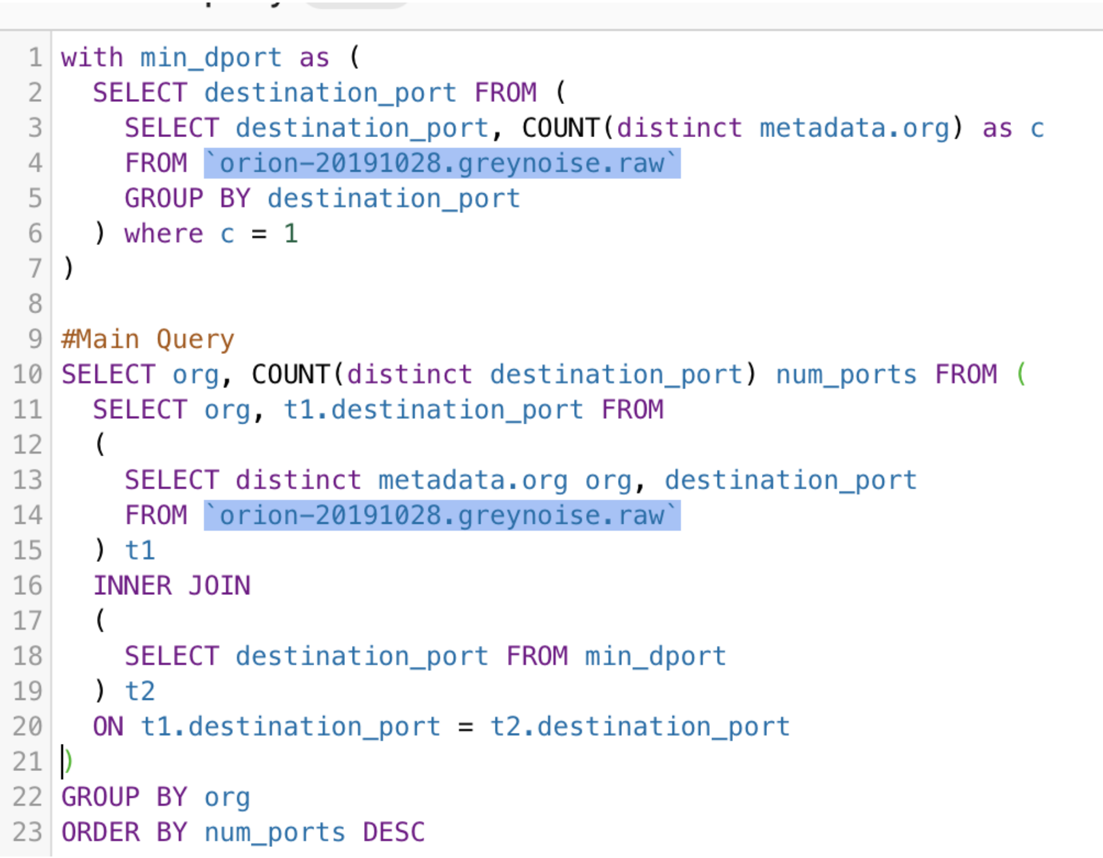


```
SELECT protocol, COUNT(distinct metadata.org) c
FROM `orion-20191028.greynoise.raw`
where destination_port = 1234
GROUP BY protocol
```

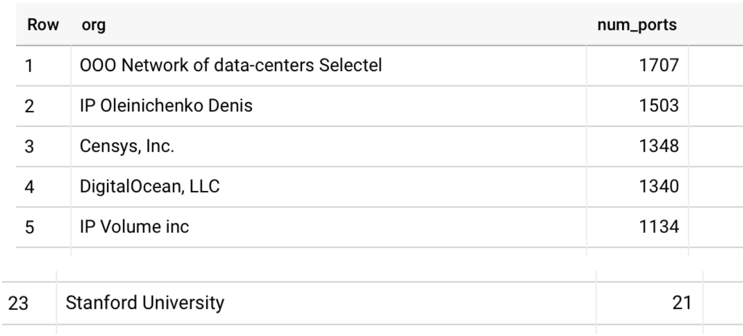


Row	protocol	c
1	UDP	312
2	TCP	8

Which organization(s) are responsible for scanning ports that are scanned by only one organization?

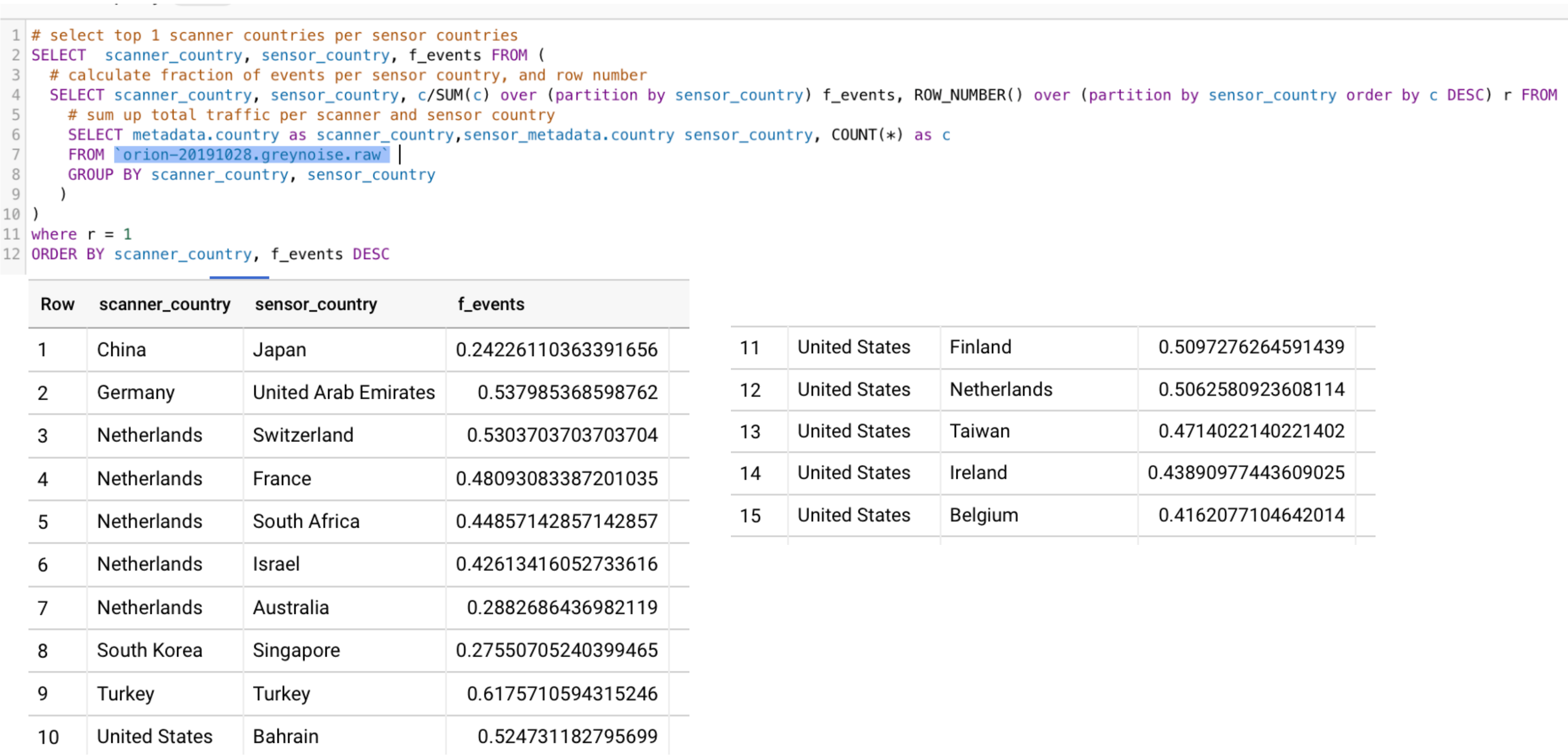


```
with min_dport as (
  SELECT destination_port FROM (
    SELECT destination_port, COUNT(distinct metadata.org) as c
    FROM `orion-20191028.greynoise.raw`
    GROUP BY destination_port
  ) where c = 1
)

#Main Query
SELECT org, COUNT(distinct destination_port) num_ports FROM (
  SELECT org, t1.destination_port FROM
  (
    SELECT distinct metadata.org org, destination_port
    FROM `orion-20191028.greynoise.raw`
  ) t1
  INNER JOIN
  (
    SELECT destination_port FROM min_dport
  ) t2
  ON t1.destination_port = t2.destination_port
)
GROUP BY org
ORDER BY num_ports DESC
```



Row	org	num_ports
1	OOO Network of data-centers Selectel	1707
2	IP Oleinichenko Denis	1503
3	Censys, Inc.	1348
4	DigitalOcean, LLC	1340
5	IP Volume inc	1134
23	Stanford University	21

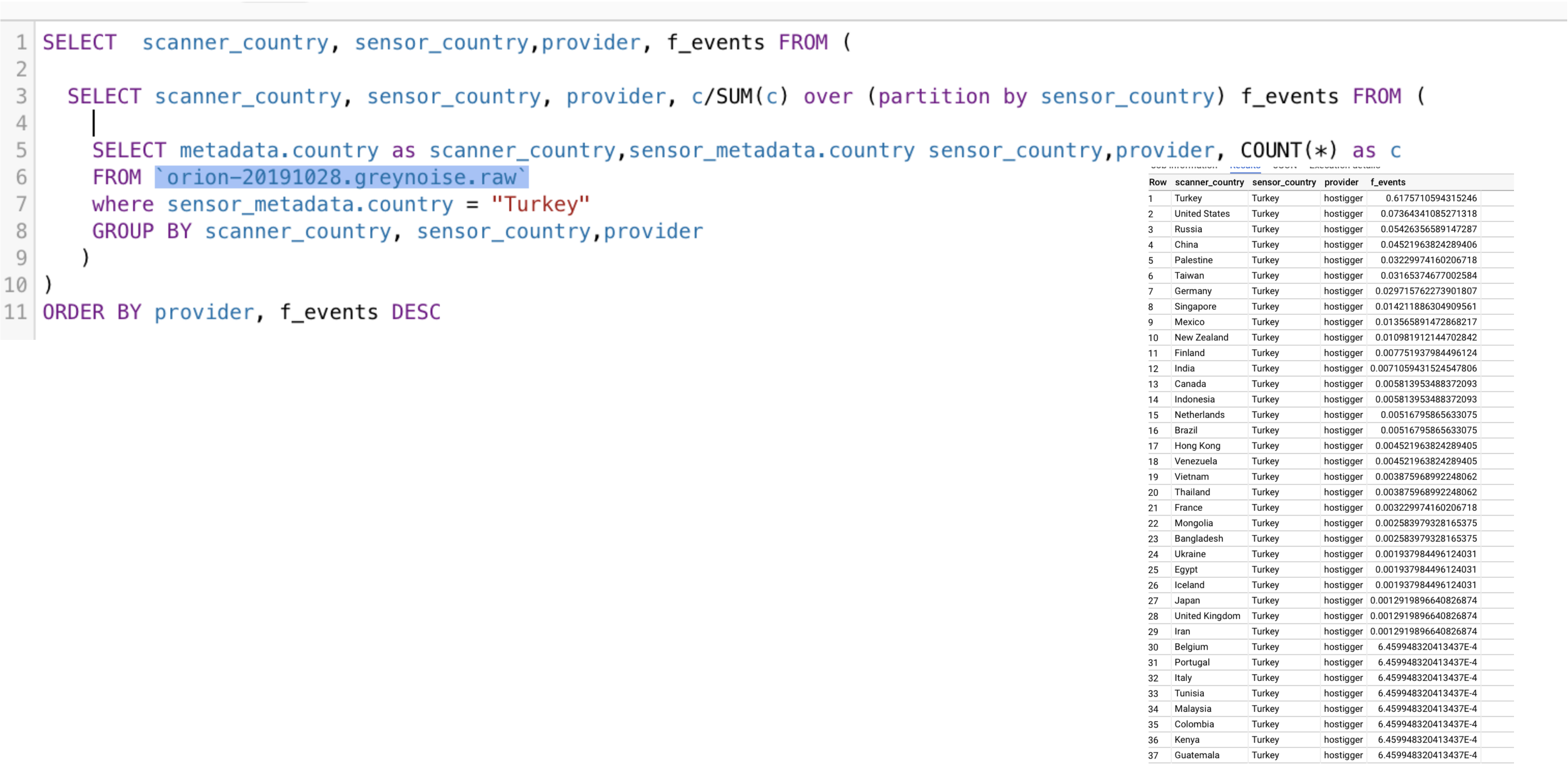
Are some countries more likely to scan hosts in particular countries?



```
# select top 1 scanner countries per sensor countries
SELECT  scanner_country, sensor_country, f_events FROM (
  # calculate fraction of events per sensor country, and row number
  SELECT scanner_country, sensor_country, c/SUM(c) over (partition by sensor_country) f_events, ROW_NUMBER() over (partition by sensor_country order by c DESC) r FROM
    # sum up total traffic per scanner and sensor country
    SELECT metadata.country as scanner_country,sensor_metadata.country sensor_country, COUNT(*) as c
    FROM `orion-20191028.greynoise.raw`
    GROUP BY scanner_country, sensor_country
   )
)
where r = 1
ORDER BY scanner_country, f_events DESC
```

Row	scanner_country	sensor_country	f_events
1	China	Japan	0.24226110363391656
2	Germany	United Arab Emirates	0.537985368598762
3	Netherlands	Switzerland	0.5303703703703704
4	Netherlands	France	0.48093083387201035
5	Netherlands	South Africa	0.44857142857142857
6	Netherlands	Israel	0.42613416052733616
7	Netherlands	Australia	0.2882686436982119
8	South Korea	Singapore	0.27550705240399465
9	Turkey	Turkey	0.6175710594315246
10	United States	Bahrain	0.524731182795699
11	United States	Finland	0.5097276264591439
12	United States	Netherlands	0.5062580923608114
13	United States	Taiwan	0.4714022140221402
14	United States	Ireland	0.43890977443609025
15	United States	Belgium	0.4162077104642014

Is the Turkey scanning Turkey thing an artifact of cloud provider?



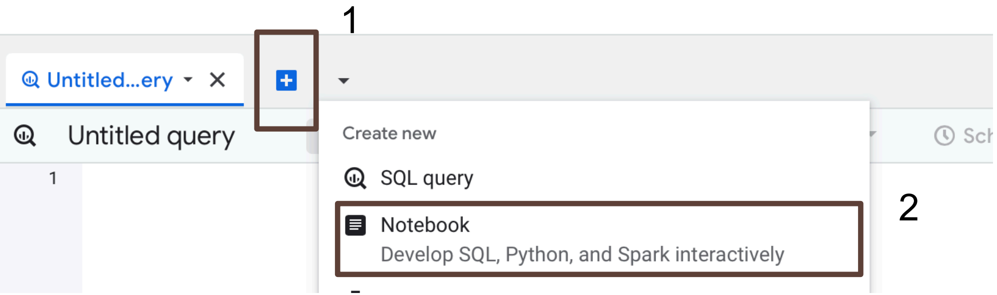
```
SELECT  scanner_country, sensor_country,provider, f_events FROM (

  SELECT scanner_country, sensor_country, provider, c/SUM(c) over (partition by sensor_country) f_events FROM (
    |
    SELECT metadata.country as scanner_country,sensor_metadata.country sensor_country,provider, COUNT(*) as c
    FROM `orion-20191028.greynoise.raw`
    where sensor_metadata.country = "Turkey"
    GROUP BY scanner_country, sensor_country,provider
   )
)
ORDER BY provider, f_events DESC
```

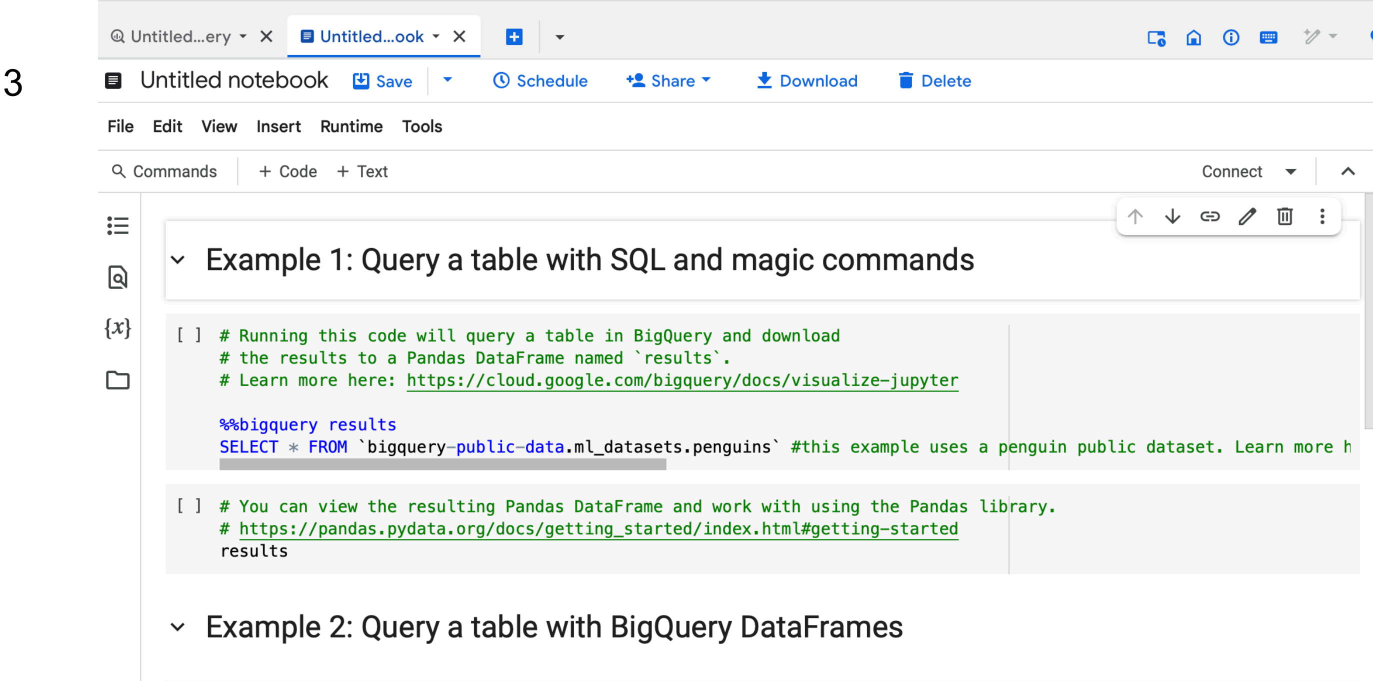
Row	scanner_country	sensor_country	provider	f_events
1	Turkey	Turkey	hostigger	0.6175710594315246
2	United States	Turkey	hostigger	0.07364341085271318
3	Russia	Turkey	hostigger	0.05426356589147287
4	China	Turkey	hostigger	0.04521963824289406
5	Palestine	Turkey	hostigger	0.03229974160206718
6	Taiwan	Turkey	hostigger	0.03165374677002584
7	Germany	Turkey	hostigger	0.029715762273901807
8	Singapore	Turkey	hostigger	0.014211886304909561
9	Mexico	Turkey	hostigger	0.013565891472868217
10	New Zealand	Turkey	hostigger	0.010981912144702842
11	Finland	Turkey	hostigger	0.007751937984496124
12	India	Turkey	hostigger	0.0071059431524547806
13	Canada	Turkey	hostigger	0.005813953488372093
14	Indonesia	Turkey	hostigger	0.005813953488372093
15	Netherlands	Turkey	hostigger	0.00516795865633075
16	Brazil	Turkey	hostigger	0.00516795865633075
17	Hong Kong	Turkey	hostigger	0.004521963824289405
18	Venezuela	Turkey	hostigger	0.004521963824289405
19	Vietnam	Turkey	hostigger	0.003875968992248062
20	Thailand	Turkey	hostigger	0.003875968992248062
21	France	Turkey	hostigger	0.003229974160206718
22	Mongolia	Turkey	hostigger	0.002583979328165375
23	Bangladesh	Turkey	hostigger	0.002583979328165375
24	Ukraine	Turkey	hostigger	0.001937984496124031
25	Egypt	Turkey	hostigger	0.001937984496124031
26	Iceland	Turkey	hostigger	0.001937984496124031
27	Japan	Turkey	hostigger	0.0012919896640826874
28	United Kingdom	Turkey	hostigger	0.0012919896640826874
29	Iran	Turkey	hostigger	0.0012919896640826874
30	Belgium	Turkey	hostigger	6.459948320413437E-4
31	Portugal	Turkey	hostigger	6.459948320413437E-4
32	Italy	Turkey	hostigger	6.459948320413437E-4
33	Tunisia	Turkey	hostigger	6.459948320413437E-4
34	Malaysia	Turkey	hostigger	6.459948320413437E-4
35	Colombia	Turkey	hostigger	6.459948320413437E-4
36	Kenya	Turkey	hostigger	6.459948320413437E-4
37	Guatemala	Turkey	hostigger	6.459948320413437E-4

So many other useful functions in BQ!

- ARRAY_AGG( ) - aggregates fields into arrays
- TO_JSON_STRING - allows for arrays and other objects to be printed in a csv output
- NET package (e.g., NET.IP_TRUNC - truncates to subnetworks (e.g., /24))
- LEAD/LAG function - allows values in neighboring rows to be compared

Analyze data in BQ Notebooks





Example 1: Query a table with SQL and magic commands

```
[ ] # Running this code will query a table in BigQuery and download
    # the results to a Pandas DataFrame named `results`.
    # Learn more here: https://cloud.google.com/bigquery/docs/visualize-jupyter

    %%bigquery results
    SELECT * FROM `bigquery-public-data.ml_datasets.penguins` #this example uses a penguin public dataset. Learn more h
```

```
[ ] # You can view the resulting Pandas DataFrame and work with using the Pandas library.
    # https://pandas.pydata.org/docs/getting_started/index.html#getting-started
    results
```

Example 2: Query a table with BigQuery DataFrames

Questions?

- ChatGPT is great for writing queries! But often overcomplicates them…and changes field names