INTRODUCTION TO FOCUS AREA

Project 4

Minie Jung, Kunaphas Kongkitimanon, Chang Wan-Ju

Abstract

The goal of the project: Understanding SQL queries and getting familiar with the BigQuery API.

The main result of the project: The query size estimation and the query description that associates IT and biological perspective

Keywords: SQL; Variant Call Format; How to find what you want from big data

An estimation of the time: 16h

Project evaluation: 3

The number of words:
3335 (errors:3) words

Task 1

Counting total rows in table

```
query = """
    SELECT
      COUNT(1) AS number_of_rows
    FROM
      `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823`
"""

query_job = client.query(
    query,
    location="US",
)  # API request - starts the query

print(bq_assistant.estimate_query_size(query))

df = query_job.to_dataframe()
df
```

Figure 1 Query1

0.0

1 entry Filter

index	number_of_rows
0	105923159

Show 25 per page

Figure 2 Result of query1

The query counts whole rows in dataset. The number of rows means there are 105923159 reference bases in the dataset.

Counting variant calls in the table

Each query returns the result which is 182104652. It means that there is about 1.7 variant calls per row in the dataset. There are 3 ways to count variant calls in the table.

Summing the lengths of call arrays

```
query = """
    SELECT SUM(ARRAY_LENGTH(call)) AS number_of_calls
    FROM `bigquery-public-
data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823`
"""

query_job = client.query(
    query,
    # Location must match that of the dataset(s) referenced in the query.
    location="US",
)

print(bq_assistant.estimate_query_size(query))

df = query_job.to_dataframe()
df
```

Figure 3 Query2

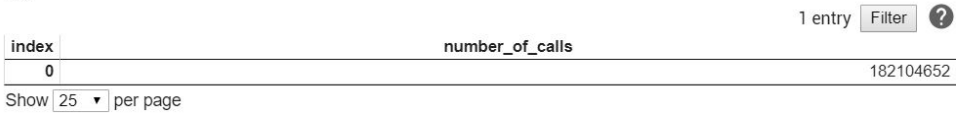

Figure 4 Result of query2

In this query, it counts the length of array of variant calls and sums all lengths of each row.

JOINing each row

```
query = """
    SELECT COUNT(call) AS number_of_calls
    FROM `bigquery-public-
data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823` v, v.call
"""

query_job = client.query(
    query,
    # Location must match that of the dataset(s) referenced in the query.
    location="US",
)

print(bq_assistant.estimate_query_size(query))

df = query_job.to_dataframe()
df
```

Figure 5 Query3

0.0

1 entry Filter

index	number_of_calls
0	182104652

Show 25 per page

Figure 6 Result of query3

There is JOIN function which is represented in ',' operator in this query. Table v and v.call can be joined and the query counts total number of variant calls.

Counting name in a call column

```
query = """
    SELECT COUNT(call.name) AS number_of_calls
    FROM `bigquery-public-
data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823` v, v.call call
"""
query_job = client.query(
    query,
    # Location must match that of the dataset(s) referenced in the query.
    location="US",
)

print(bq_assistant.estimate_query_size(query))

df = query_job.to_dataframe()
df
```

Figure 7 Query4

1.5263835601508617

1 entry Filter

index	number_of_calls
0	182104652

Show 25 per page

Figure 8 Result of query4

This query counts the name values in call column in joined table.

Counting variant and non-variant segment

In the sum of the lengths of call array which contain variant and non-variant segment, so we calculate the number of variant segments and non-variant separately.

Variants

```
query = """
    SELECT COUNT(1) AS number_of_real_variants
    FROM `bigquery-public-
data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823` v, v.call call
    WHERE EXISTS (SELECT 1 FROM UNNEST(v.alternate_bases) AS alt WHERE alt.alt NOT IN ("<NON_REF>",
"<*>"))
    """

query_job = client.query(
    query,
    # Location must match that of the dataset(s) referenced in the query.
    location="US",
)

print(bq_assistant.estimate_query_size(query))

df = query_job.to_dataframe()
df
```

Figure 9 Query5

0.5340448450297117

1 entry Filter

index	number_of_real_variants
0	38549388

Show 25 per page

Figure 10 Result of query5

In this query, it counts variants by counting calls which is not <NON_REF> or <*> in alternative_base column. It takes roughly 500MB and it because the query needs to scan alternative_base column but can ignore values which have <NON_REF> or <*>. The result means there are 38549388 variants of reference bases in human genome.

Non-variant

```
query = """
    SELECT COUNT(1) AS number_of_non_variants
    FROM `bigquery-public-
data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823` v, v.call call
    WHERE NOT EXISTS (SELECT 1 FROM UNNEST(v.alternate_bases) AS alt WHERE alt.alt NOT IN
("<NON_REF>", "<*>"))
    """

query_job = client.query(
    query,
    # Location must match that of the dataset(s) referenced in the query.
    location="US",
)

print(bq_assistant.estimate_query_size(query))

df = query_job.to_dataframe()
df
```

Figure 11 Query6

0.5340448450297117

1 entry Filter

index	number_of_non_variants
0	143555264

Show 25 per page

Figure 12 Result of query6

In here, we also scan same column with former query but it counts calls which is <NON_REF> or <*> in alternate_bases column. It also takes about 500MB and the result represents the number of non-variants of reference bases in human genome.

Counting the variants called by each sample

```
query = """
    SELECT call.name AS call_name, COUNT(call.name) AS call_count_for_call_set
    FROM `bigquery-public-
data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823` v, v.call
    GROUP BY call_name
    ORDER BY call_name
    """

query_job = client.query(
    query,
    # Location must match that of the dataset(s) referenced in the query.
    location="US",
)

print(bq_assistant.estimate_query_size(query))

df = query_job.to_dataframe()
df
```

Figure 13 Query7

1.5263835601508617

1 to 6 of 6 entries Filter

index	call_name	call_count_for_call_set
0	NA12877	31592135
1	NA12878	28012646
2	NA12889	31028550
3	NA12890	30636087
4	NA12891	33487348
5	NA12892	27347886

Show 25 per page

Figure 14 Result of query7

This query counts number of rows for each call.name. It scanned all row of joined table(182104652) and it took roughly 1.5GB to scan and count. In the result, there are 6 rows and each row is call.name, pedigree of each call. Also, there are variants and non-variants.

```
query = """
    SELECT call.name AS call_name, COUNT(call.name) AS call_count_for_call_set
    FROM `bigquery-public-
data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823` v, v.call
    WHERE EXISTS (SELECT 1 FROM UNNEST(v.alternate_bases) AS alt WHERE alt.alt NOT IN ("<NON_REF>",
"<*>"))
    GROUP BY call_name
    ORDER BY call_name
    """

query_job = client.query(
    query,
    # Location must match that of the dataset(s) referenced in the query.
    location="US",
)

print(bq_assistant.estimate_query_size(query))

df = query_job.to_dataframe()
df
```

Figure 15 Query8

2.0604284051805735

1 to 6 of 6 entries Filter

index	call_name	call_count_for_call_set
0	NA12877	6284275
1	NA12878	6397315
2	NA12889	6407532
3	NA12890	6448600
4	NA12891	6516669
5	NA12892	6494997

Show 25 per page

Figure 16 Result of query8

We can filter out non-variants using this query. To filter, it scan alternate_base column and return 1 for each row which is not <NON_REF> or <*>. It took more memory than counting calls by each sample, because this query needs to compare alternate_base value. In this result, we can obtain variants classified by pedigree.

Filtering true variants by genotype

```
query = """
    SELECT call.name AS call_name, COUNT(call.name) AS call_count_for_call_set
    FROM `bigquery-public-
data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823` v, v.call
    WHERE EXISTS (SELECT 1 FROM UNNEST(call.genotype) AS gt WHERE gt > 0) AND NOT EXISTS (SELECT 1
FROM UNNEST(call.genotype) AS gt WHERE gt < 0)
    GROUP BY call_name
    ORDER BY call_name
    """

query_job = client.query(
    query,
    # Location must match that of the dataset(s) referenced in the query.
    location="US",
)

print(bq_assistant.estimate_query_size(query))

df = query_job.to_dataframe()
df
```

Figure 17 Query9

4.239954333752394

1 to 6 of 6 entries Filter

index	call_name	call_count_for_call_set
0	NA12877	4486610
1	NA12878	4502017
2	NA12889	4422706
3	NA12890	4528725
4	NA12891	4424094
5	NA12892	4495753

Show 25 per page

Figure 18 Result of query9

After filtering non-variants, there are no-calls in the table so we can select true variants using this query with genotype column. Genotype "-1" is used in cases where the genotype is not called. So it took about 4.2GB, which is more than selecting variants, because it need to scan 3 columns. After this query, we can obtain the table about the number of true variants by each pedigree.

Counting samples in the table

```
query = """
    SELECT COUNT(DISTINCT call.name) AS number_of_callsets
    FROM `bigquery-public-
data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823` v, v.call
    """

query_job = client.query(
    query,
    # Location must match that of the dataset(s) referenced in the query.
    location="US",
)
print(bq_assistant.estimate_query_size(query))

df = query_job.to_dataframe()
df
```

Figure 19 Query10

1.5263835601508617

1 entry Filter

index	number_of_callsets
0	6

Show 25 per page

Figure 20 Result of query10

In this query, we count the row of table of counting number of calls by each sample. It takes about 1.5 GB which is as same as scanning one column(counting the number of calls by each sample). The result means there are 6 pedigrees in this dataset.

Counting variants per chromosome

```
query = """
    SELECT reference_name, COUNT(reference_name) AS number_of_variant_rows
    FROM `bigquery-public-
data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823` v
    WHERE EXISTS (SELECT 1 FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt WHERE gt > 0)
    GROUP BY reference_name
    ORDER BY
      CASE
        WHEN SAFE_CAST(REGEXP_REPLACE(reference_name, '^chr', '') AS INT64) < 10
          THEN CONCAT('0', REGEXP_REPLACE(reference_name, '^chr', ''))
          ELSE REGEXP_REPLACE(reference_name, '^chr', '')
      END
    """

query_job = client.query(
    query,
    # Location must match that of the dataset(s) referenced in the query.
    location="US",
)

print(bq_assistant.estimate_query_size(query))

df = query_job.to_dataframe()
df
```

Figure 21 Query11

3.341447635553777

1 to 24 of 24 entries Filter

index	reference_name	number_of_variant_rows
0	chr1	615000
1	chr2	646401
2	chr3	542315
3	chr4	578600
4	chr5	496202
5	chr6	512152
6	chr7	459506
7	chr8	416376
8	chr9	344985
9	chr10	396773
10	chr11	391260
11	chr12	382841
12	chr13	298044
13	chr14	258756
14	chr15	234569
15	chr16	247671
16	chr17	224403
17	chr18	227200
18	chr19	192538
19	chr20	168958
20	chr21	121882
21	chr22	112338
22	chrX	231125
23	chrY	15357

Show 25 per page

Figure 22 Result of query11

We can calculate how many calls(genotype > 0) are in each chromosome with this query. It took about 3.3GB even though scanning 2 columns, because this query needs to convert string type data to integer. The result means, for example, there are 615000 variants in chromosome 1.

Counting high-quality variants per sample

Querying calls with multiple FILTER values

```
query = """
    SELECT call_filter, COUNT(call_filter) AS number_of_calls
    FROM `bigquery-public-
data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823` v, v.call,
UNNEST(call.FILTER) AS call_filter
    GROUP BY call_filter
    ORDER BY number_of_calls
    """

query_job = client.query(
    query,
    # Location must match that of the dataset(s) referenced in the query.
    location="US",
)

print(bq_assistant.estimate_query_size(query))

df = query_job.to_dataframe()
df
```

Figure 23 Query12

0.24804932065308094

1 to 2 of 2 entries Filter

index	call_filter	number_of_calls
0	RefCall	11681534
1	PASS	26867854

Show 25 per page

Figure 24 Result of query12

In the dataset, there is a column which contains filter values. The PASS value of filter column means that a variant call is of a high quality. This query scan only variants(38549388) so took roughly 240MB. The result means there are 11681534 high quality variants.

FILTERing for high quality variant calls

```
query = """
    SELECT reference_name, start_position, end_position, reference_bases, call.name AS call_name,
(SELECT STRING_AGG(call_filter) FROM UNNEST(call.FILTER) AS call_filter) AS filters,
ARRAY_LENGTH(call.FILTER) AS filter_count
    FROM `bigquery-public-
data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823` v, v.call
    WHERE EXISTS (SELECT 1 FROM UNNEST(call.FILTER) AS call_filter WHERE call_filter = 'PASS') AND
ARRAY_LENGTH(call.FILTER) > 1
    ORDER BY filter_count DESC, reference_name, start_position, end_position, reference_bases,
call_name
    LIMIT 10
    """

query_job = client.query(
    query,
    # Location must match that of the dataset(s) referenced in the query.
    location="US",
)

print(bq_assistant.estimate_query_size(query))

df = query_job.to_dataframe()
df
```

Figure 25 Query13

4.281298340298235

No entries Filter

index	reference_name	start_position	end_position	reference_bases	call_name	filters	filter_count

Show 25 per page

Figure 26 Result of query13

Scanning the FILTER column and filtering out the calls which have lower quality by omitting the calls that do not contain a PASS value.

Counting all high quality calls for each sample

```
query = """
    SELECT call.name AS call_name, COUNT(1) AS number_of_calls
    FROM `bigquery-public-
data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823` v, v.call
    WHERE NOT EXISTS (SELECT 1 FROM UNNEST(call.FILTER) AS call_filter WHERE call_filter != 'PASS')
    GROUP BY call_name
    ORDER BY call_name
    """

query_job = client.query(
    query,
    # Location must match that of the dataset(s) referenced in the query.
    location="US",
)

print(bq_assistant.estimate_query_size(query))

df = query_job.to_dataframe()
df
```

Figure 27 Query14

1.7744328808039427

1 to 6 of 6 entries Filter

index	call_name	number_of_calls
0	NA12877	29795946
1	NA12878	26118774
2	NA12889	29044992
3	NA12890	28717437
4	NA12891	31395995
5	NA12892	25349974

Show 25 per page

Figure 28 Result of query14

This query scan 2 columns, call.name and call.filter, and count the number of calls which have 'PASS' value in call.filter. It ignored rows which have non PASS of filter value so it took 1.7GB. After this query, we can obtain the number of high quality calls by each pedigree but it still contains non variants.

Counting all high quality true variant calls for each sample

```
query = """
    SELECT call.name AS call_name, COUNT(1) AS number_of_calls
    FROM `bigquery-public-
data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823` v, v.call
    WHERE NOT EXISTS (SELECT 1 FROM UNNEST(call.FILTER) AS call_filter WHERE call_filter != 'PASS')
AND EXISTS (SELECT 1 FROM UNNEST(call.genotype) as gt WHERE gt > 0)
    GROUP BY call_name
    ORDER BY call_name
    """

query_job = client.query(
    query,
    # Location must match that of the dataset(s) referenced in the query.
    location="US",
)

print(bq_assistant.estimate_query_size(query))

df = query_job.to_dataframe()
df
```

Figure 29 Query15

4.488003654405475

1 to 6 of 6 entries Filter

index	call_name	number_of_calls
0	NA12877	4486610
1	NA12878	4502017
2	NA12889	4422706
3	NA12890	4528725
4	NA12891	4424094
5	NA12892	4495753

Show 25 ▾ per page

Figure 30 Result of query15

In this query, we can get a table with the number of high quality true variants by each pedigree. This query scans call.name, call.filter, and genotype so it takes about 4.5GB.

Condensing queries

```
query = """
    SELECT reference_name, COUNT(reference_name) AS number_of_variant_rows
    FROM `bigquery-public-
data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823` v
    WHERE EXISTS (SELECT 1 FROM UNNEST(v.call) AS call WHERE EXISTS (SELECT 1 FROM
UNNEST(call.genotype) AS gt WHERE gt > 0))
    GROUP BY reference_name
    ORDER BY reference_name
    """

query_job = client.query(
    query,
    # Location must match that of the dataset(s) referenced in the query.
    location="US",
)

print(bq_assistant.estimate_query_size(query))

df = query_job.to_dataframe()
df
```

Figure 31 Query16

3.341447635553777

1 to 24 of 24 entries Filter

index	reference_name	number_of_variant_rows
0	chr1	615000
1	chr10	396773
2	chr11	391260
3	chr12	382841
4	chr13	298044
5	chr14	258756
6	chr15	234569
7	chr16	247671
8	chr17	224403
9	chr18	227200
10	chr19	192538
11	chr2	646401
12	chr20	168958
13	chr21	121882
14	chr22	112338
15	chr3	542315
16	chr4	578600
17	chr5	496202
18	chr6	512152
19	chr7	459506
20	chr8	416376
21	chr9	344985
22	chrX	231125
23	chrY	15357

Show 25 per page

Figure 32 Result of query16

This query counts variants by each chromosome. It scans genotype and reference name, also order the result by chromosome so it takes about 3.4GB. It uses EXIST to count variants which have genotype value larger than 0. The result table is sorted but reference name is string so this query cannot sort it properly.

```
query = """
    SELECT reference_name, COUNT(reference_name) AS number_of_variant_rows
    FROM `bigquery-public-
data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823` v
    WHERE EXISTS (SELECT 1 FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt WHERE gt > 0)
    GROUP BY reference_name
    ORDER BY reference_name
    """

query_job = client.query(
    query,
    # Location must match that of the dataset(s) referenced in the query.
    location="US",
)

print(bq_assistant.estimate_query_size(query))

df = query_job.to_dataframe()
df
```

Figure 33 Query17

3.341447635553777

1 to 24 of 24 entries Filter

index	reference_name	number_of_variant_rows
0	chr1	615000
1	chr10	396773
2	chr11	391260
3	chr12	382841
4	chr13	298044
5	chr14	258756
6	chr15	234569
7	chr16	247671
8	chr17	224403
9	chr18	227200
10	chr19	192538
11	chr2	646401
12	chr20	168958
13	chr21	121882
14	chr22	112338
15	chr3	542315
16	chr4	578600
17	chr5	496202
18	chr6	512152
19	chr7	459506
20	chr8	416376
21	chr9	344985
22	chrX	231125
23	chrY	15357

Show 25 per page

Figure 34 Result of query17

In this query, we can obtain the same result with former query but it uses JOIN to count variants which have genotype value larger than 0. This query is more condensed than the former one.

```
query = """
    SELECT REGEXP_REPLACE(reference_name, '^chr', '') AS chromosome, COUNT(reference_name) AS
number_of_variant_rows
    FROM `bigquery-public-
data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823` v
    WHERE EXISTS (SELECT 1 FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt WHERE gt > 0)
    GROUP BY chromosome
    ORDER BY chromosome
    """

query_job = client.query(
    query,
    # Location must match that of the dataset(s) referenced in the query.
    location="US",
)

print(bq_assistant.estimate_query_size(query))

df = query_job.to_dataframe()
df
```

Figure 35 Query18

3.341447635553777

1 to 24 of 24 entries Filter

index	chromosome	number_of_variant_rows
0	1	615000
1	10	396773
2	11	391260
3	12	382841
4	13	298044
5	14	258756
6	15	234569
7	16	247671
8	17	224403
9	18	227200
10	19	192538
11	2	646401
12	20	168958
13	21	121882
14	22	112338
15	3	542315
16	4	578600
17	5	496202
18	6	512152
19	7	459506
20	8	416376
21	9	344985
22	X	231125
23	Y	15357

Show 25 per page

Figure 36 Result of query18

To sort the result properly, this query delete ‘chr’ from each reference_name. But the values are still string, so we need to change them to integer.

```
query = """
    SELECT CAST(REGEXP_REPLACE(reference_name, '^chr', '') AS INT64) AS chromosome,
COUNT(reference_name) AS number_of_variant_rows
    FROM `bigquery-public-
data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823` v
    WHERE EXISTS (SELECT 1 FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt WHERE gt > 0)
    GROUP BY chromosome
    ORDER BY chromosome
    """

query_job = client.query(query)

print(bq_assistant.estimate_query_size(query))

df = query_job.to_dataframe()
df
```

Figure 37 Query19

```
3.341447635553777
---------------------------------------------------------------------------
BadRequest                                Traceback (most recent call last)
<ipython-input-34-39556dd8c955> in <module>()
     11 print(bq_assistant.estimate_query_size(query))
     12
---> 13 df = query_job.to_dataframe()
     14 df

                                    14 frames

/usr/local/lib/python3.6/dist-packages/google/cloud/_http.py in api_request(self, method, path, query_params,
    391
    392         if not 200 <= response.status_code < 300:
--> 393             raise exceptions.from_http_response(response)
    394
    395         if expect_json and response.content:

BadRequest: 400 GET https://bigquery.googleapis.com/bigquery/v2/projects/unique-apricot-258711/queries/eb8c0

(job ID: eb8c0854-15af-45e7-af25-d69902e278c9)

                                        -----Query Job SQL Follows-----

    |    .    |    .    |    .    |    .    |    .    |    .    |    .    |    .    |    .    |    .    |
   1:
   2:    SELECT CAST(REGEXP_REPLACE(reference_name, '^chr', '') AS INT64) AS chromosome, COUNT(reference_name
   3:    FROM `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823` v
   4:    WHERE EXISTS (SELECT 1 FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt WHERE gt > 0)
   5:    GROUP BY chromosome
   6:    ORDER BY chromosome
   7:
    |    .    |    .    |    .    |    .    |    .    |    .    |    .    |    .    |    .    |    .    |
```

Figure 38 Result of query19

In this query, it tries to convert string to integer, but the result is error. Because in reference_name, there are some characters such as ‘X’ and ‘Y’.

```
query = """
    SELECT CASE WHEN SAFE_CAST(REGEXP_REPLACE(reference_name, '^chr', '') AS INT64) < 10 THEN
CONCAT('0', REGEXP_REPLACE(reference_name, '^chr', '')) ELSE REGEXP_REPLACE(reference_name, '^chr',
'') END AS chromosome, COUNT(reference_name) AS number_of_variant_rows
    FROM `bigquery-public-
data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823` v
    WHERE EXISTS (SELECT 1 FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt WHERE gt > 0)
    GROUP BY chromosome
    ORDER BY chromosome
    """

query_job = client.query(
    query,
    # Location must match that of the dataset(s) referenced in the query.
    location="US",
)

print(bq_assistant.estimate_query_size(query))

df = query_job.to_dataframe()
df
```

Figure 39 Query20

3.341447635553777

1 to 24 of 24 entries Filter

index	chromosome	number_of_variant_rows
0	01	615000
1	02	646401
2	03	542315
3	04	578600
4	05	496202
5	06	512152
6	07	459506
7	08	416376
8	09	344985
9	10	396773
10	11	391260
11	12	382841
12	13	298044
13	14	258756
14	15	234569
15	16	247671
16	17	224403
17	18	227200
18	19	192538
19	20	168958
20	21	121882
21	22	112338
22	X	231125
23	Y	15357

Show 25 per page

Figure 40 Result of query20

So this query uses SAFE_CAST function which returns NULL for chromosomes X and Y instead of error. Also it puts '0' before the number(only for 1 to 9) and removes 'chr'. Therefore, the result table is sorted correctly.

```
query = """
    SELECT reference_name, COUNT(reference_name) AS number_of_variant_rows
    FROM `bigquery-public-
data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823` v
    WHERE EXISTS (SELECT 1 FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt WHERE gt > 0)
    GROUP BY reference_name
    ORDER BY
      CASE
        WHEN SAFE_CAST(REGEXP_REPLACE(reference_name, '^chr', '') AS INT64) < 10
          THEN CONCAT('0', REGEXP_REPLACE(reference_name, '^chr', ''))
          ELSE REGEXP_REPLACE(reference_name, '^chr', '')
      END
    """

query_job = client.query(
    query,
    # Location must match that of the dataset(s) referenced in the query.
    location="US",
)

print(bq_assistant.estimate_query_size(query))

df = query_job.to_dataframe()
df
```

Figure 41 Query21

3.341447635553777

1 to 24 of 24 entries Filter

index	reference_name	number_of_variant_rows
0	chr1	615000
1	chr2	646401
2	chr3	542315
3	chr4	578600
4	chr5	496202
5	chr6	512152
6	chr7	459506
7	chr8	416376
8	chr9	344985
9	chr10	396773
10	chr11	391260
11	chr12	382841
12	chr13	298044
13	chr14	258756
14	chr15	234569
15	chr16	247671
16	chr17	224403
17	chr18	227200
18	chr19	192538
19	chr20	168958
20	chr21	121882
21	chr22	112338
22	chrX	231125
23	chrY	15357

Show 25 per page

Figure 42 Result of query21

To improve the output, CAST function is moved to ORDER BY to maintain the values of reference_name in its original form.

```
query_f = """
CREATE TEMPORARY FUNCTION SortableChromosome(reference_name STRING)
  RETURNS STRING AS (
  -- Remove the leading "chr" (if any) in the reference_name
  -- If the chromosome is 1 - 9, prepend a "0" since
  -- "2" sorts after "10", but "02" sorts before "10".
  CASE
    WHEN SAFE_CAST(REGEXP_REPLACE(reference_name, '^chr', '') AS INT64) < 10
      THEN CONCAT('0', REGEXP_REPLACE(reference_name, '^chr', ''))
      ELSE REGEXP_REPLACE(reference_name, '^chr', '')
  END
);

SELECT
  reference_name,
  COUNT(reference_name) AS number_of_variant_rows
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_20180823` v
WHERE
  EXISTS (SELECT 1
            FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt
           WHERE gt > 0)
GROUP BY
  reference_name
ORDER BY SortableChromosome(reference_name)
"""

query_f_job = client.query(
    query_f,
    # Location must match that of the dataset(s) referenced in the query.
    location="US",
)

print(bq_assistant.estimate_query_size(query_f))

dff = query_f_job.to_dataframe()
dff
```

Figure 43 Query22

Result

3.341447635553777

1 to 24 of 24 entries Filter

index	reference_name	number_of_variant_rows
0	chr1	615000
1	chr2	646401
2	chr3	542315
3	chr4	578600
4	chr5	496202
5	chr6	512152
6	chr7	459506
7	chr8	416376
8	chr9	344985
9	chr10	396773
10	chr11	391260
11	chr12	382841
12	chr13	298044
13	chr14	258756
14	chr15	234569
15	chr16	247671
16	chr17	224403
17	chr18	227200
18	chr19	192538
19	chr20	168958
20	chr21	121882
21	chr22	112338
22	chrX	231125
23	chrY	15357

Show 25 per page

Figure 44 Result of query22

In this query, there is a user-defined function about sorting reference_name. This query is similar to the query which put CAST function at ORDER BY. It represents a same result which is the number of variants by each chromosome.

Task 2

In task 2, we select some of queries from 6 sections to report, and we also give a brief summary and roughly explanation on each query. However, some queries does not contain raw results but query estimation size, due to our credits exceed the maximum free-trial amount.

Exploring the sample information data

Samples Vs Sample in Variants Table

```
QUERY = """
# Count the number of samples in the phenotypic data
SELECT
   COUNT(sample) AS all_samples,
   SUM(IF(In_Phase1_Integrated_Variant_Set = TRUE, 1, 0)) AS samples_in_variants_table
FROM
   `genomics-public-data.1000_genomes.sample_info`
   """
df = client.query(QUERY).to_dataframe()
df
```

1 entry Filter

index	all_samples	samples_in_variants_table
0	3500	1092

Show 25 per page

Figure 45 This query count samples and variants

Compare total variants by all samples and the samples that have variants in phase 1 in the database. The process simply count and sum on all 1000 genome records, so this query process around 3e-05 bytes.

Gender ratio

```
QUERY = """
# Compute sample count and ratio by gender
SELECT
  gender,
  gender_count,
  RATIO_TO_REPORT(gender_count)
OVER
  (
  ORDER BY
    gender_count) AS gender_ratio
FROM (
  SELECT
    gender,
    COUNT(gender) AS gender_count,
  FROM
    [genomics-public-data:1000_genomes.sample_info]
  WHERE
    In_Phase1_Integrated_Variant_Set = TRUE
  GROUP BY
    gender)
  """
```

1 to 2 of 2 entries Filter

index	gender	gender_count	gender_ratio
0	male	525	0.480769230769230
1	female	567	0.519230769230769

Show 25 per page

Figure 46 This query computes sample count and ratio by gender

This query computes sample count and ratio by gender using "RATIO_TO_REPORT" function to calculate the sex ratio in the analysis. The process consumes roughly 25612 bytes.

The ratios of ethnicities grouped by super population

```
QUERY = """
# Ratios of ethnicities grouped by super population
SELECT super_population, super_population_description, super_population_count,
  RATIO_TO_REPORT(super_population_count)
OVER
  ( ORDER BY super_population_count) AS super_population_ratio
from(
  SELECT
    super_population,
    super_population_description,
    COUNT(population) AS super_population_count
  FROM
    [genomics-public-data:1000_genomes.sample_info]
  WHERE
    In_Phase1_Integrated_Variant_Set = TRUE
  GROUP BY
    super_population,
    super_population_description)
 """
df = client.query(QUERY , job_config=bigquery.QueryJobConfig(use_legacy_sql = True)).to_dataframe()
df
# Count Plot (a.k.a. Bar Plot)
sns.catplot(x='super_population',y='super_population_count', data=df, palette=type_colors,
kind="bar")
# Rotate x-labels
plt.xticks(rotation=-45)
```

1 to 4 of 4 entries Filter

index	super_population	super_population_description	super_population_count	super_population_ratio
0	AMR	American	181	0.1657509157509574
1	AFR	African	246	0.2252747252747528
2	EAS	East Asian	286	0.2619047619047619
3	EUR	European	379	0.3470695970695706

Show 25 per page

Figure 47 The query determines ratios of ethnicities grouped by the super population.

Firstly, we group rows that have the same values on "super_population" and "super_population_description ", and then using "RATIO_TO_REPORT" to calculate the ethnicity ratio in the analysis. The process consumes roughly 73291 bytes.

What is the distribution of family sizes?

```
QUERY = """
# Compute the distribution of family sizes
SELECT
num_family_members AS family_size,
COUNT(num_family_members) AS num_families_of_size
FROM (
  SELECT
  family_id,
  COUNT(family_id) AS num_family_members
  FROM
  `genomics-public-data.1000_genomes.sample_info`
  WHERE
  In_Phase1_Integrated_Variant_Set = TRUE
  GROUP BY
  family_id)
GROUP BY
family_size
 """
```

1 to 4 of 4 entries Filter

index	family_size	num_families_of_size
0	1	636
1	2	204
2	3	12
3	4	3

Show 25 per page

Figure 48 The query is showing the distribution of family sizes.

First, we create a table that counts the number of members in each family as a SUBQUERY; then we query from the SUBQUERY table to generate a size of family and groups the row that the same value in family size. This query will process 26072 bytes.

Basic Plots of 1k Genome Data

Min/Max Chromosomal Positions of Variants

In this table, we can understand the higher minimum and maximum variant may possible occur in which chromosome. We use "MIN" to find a minimum

```
QUERY = """
SELECT
   INTEGER(reference_name) AS chromosome,
   MIN(start) AS min,
   MAX(start) AS max
 FROM
   [genomics-public-data:1000_genomes.variants]
 OMIT RECORD IF
   reference_name IN ("X", "Y", "MT")
 GROUP BY
   chromosome
   """
#print(bq_assistant.estimate_query_size(QUERY))
df = client.query(QUERY, job_config=bigquery.QueryJobConfig(use_legacy_sql = True)).to_dataframe()
df
```

1 to 22 of 22 entries Filter

index	chromosome	min	max
0	11	70854	134946450
1	5	11939	180885154
2	21	9411242	48119750
3	16	60053	90292810
4	13	19020012	115109851
5	20	60478	62965161
6	8	10421	146303865

Figure 49 The query is showing the max and min start position of each chromosome in the 1000 genome.

length and "MAX" to get a maximum length and group by chromosome. However, we remove "X", "Y", and "MT" chromosome by using "OMIT RECORD IF". Since the process has to scan all chromosomes and finding max/min of the starting point for all 1000 genome records, so this query process around 0.45 GB for the data analysis.

Quality score of calls (at least, of INDELs)

Analysis the quality which represents the possibility of mutation. We

```
QUERY = """
SELECT
  vt AS variant_type,
  quality,
  COUNT(1) AS cnt
FROM
  [genomics-public-data:1000_genomes.variants]
OMIT RECORD IF
  reference_name IN ("X", "Y", "MT")
GROUP BY
  variant_type,
  quality
  """
#print(bq_assistant.estimate_query_size(QUERY))
df = client.query(QUERY , job_config=bigquery.QueryJobConfig(use_legacy_sql = True)).to_dataframe()
df
```

1 to 25 of 3467 entries Filter

index	variant_type	quality	cnt
0	INDEL	897.0	230
1	INDEL	218.0	3476
2	INDEL	534.0	1323
3	INDEL	402.0	1261
4	INDEL	560.0	838
5	INDEL	117.0	2847
6	INDEL	1732.0	13

Figure 50 This query is showing the total amount of each quality score per each variant type.

use "COUNT(1)" method to mark each quality score per variant type as one, and then group by quality score and variant type to see the total count for each quality score per each variant type. However, we remove "X", "Y", and "MT" chromosome by using "OMIT RECORD IF". Since the process has to scan all variant types in 1000 genome records and count each record before proceeding with aggregation, so this query process around 0.51 GB for the data analysis.

Likelihood Scores for each Allele

```
QUERY = """
SELECT
  variant_type,
  likelihood,
  COUNT(1) AS cnt
FROM (
  SELECT
    variant_type,
    ROUND(100 * IF(gl > -0.5, gl,
          -0.5)) AS likelihood,
  FROM
     FLATTEN((SELECT
         reference_name,
         vt AS variant_type,
         call.call_set_name AS genome,
         call.phaseset AS phaseset,
         call.genotype_likelihood AS gl,
         NTH(1, call.genotype) WITHIN call AS first_allele,
         NTH(2, call.genotype) WITHIN call AS second_allele
       FROM [genomics-public-data:1000_genomes.variants])
     , call)
  WHERE
       (first_allele <= second_allele
         AND POSITION(gl) = 1 +
         (second_allele *
           (second_allele + 1) / 2) +
         first_allele)
        OR (second_allele < first_allele
         AND POSITION(gl) = 1 +
         (first_allele *
           (first_allele + 1) / 2) +
         second_allele)
  OMIT RECORD IF
       reference_name in ("X", "Y", "MT")
    OR phaseset IS NULL
)
GROUP BY
  variant_type,
  likelihood
  """
```

Figure 51 This query is showing the likelihood scores for each allele.

Firstly, The query performs “FLATTEN ” function to get “first_allele” and “second_allele” columns. Next, we create a likelihood score by using “IF” function to get a genotype and apply “ROUND” to convert results into decimals. However, we apply “WHERE” clause and remove records If they belong to“X”, “Y”, and “MT” chromosome by using “OMIT RECORD IF”. Finally, The query use “COUNT” function and “GROUP BY” function on “variant_type” and “likelihood” columns to returns the number of rows in each group. Since the query process has to perform nested “FROM” query and “FLATTEN” function, so this query process around 1722.62 GB for the data analysis.

Types of SNP pairs (looks at both alleles)

```
QUERY = """
SELECT
  reference_bases AS reference,
  CONCAT(
    IF(first_allele=0,
       reference_bases,
       alternate_bases),
    "|",
    IF(second_allele=0,
       reference_bases,
       alternate_bases)
    ) AS alleles,
  COUNT(1) AS cnt,
FROM
     FLATTEN((SELECT
         reference_name,
         reference_bases,
         alternate_bases,
         vt,
         NTH(1, call.genotype) WITHIN call AS first_allele,
         NTH(2, call.genotype) WITHIN call AS second_allele
       FROM [genomics-public-data:1000_genomes.variants])
     , call)
OMIT RECORD IF
     reference_name IN ("X", "Y", "MT")
  OR first_allele < 0
  OR second_allele < 0
  OR vt != "SNP"
GROUP BY
  reference,
  alleles
  """
```

1 to 25 of 40 entries Filter

index	reference	alleles	cnt
0	A	A|G	190193924
1	A	A|C	47649777
2	G	G|T	52164964
3	T	T|C	190350406
4	C	C|G	52163897
5	T	G|T	47380525
6	A	A|T	42098551

Figure 52 This query is showing types of SNP pairs.

Firstly, the query is creating a nested query from "call" and returns the data with a flattened output. Next, we create alleles column by using "CONCAT" for building allele pairs and the "IF" function for creating an output (e.g. if "first_allele" or "second_allele" is equal to 0 then it is reference base; otherwise it is an alternative base.) before concatenation. However, we combine the "OR" conditions; removing records If they belong to"X", "Y", and "MT" chromosome by using "OMIT RECORD IF", "first_allele" and "second_allele" are less than zero, or variant type does not equal to SNP. Since the query process has to query nested data and return the data with a flattened output before proceeding with aggregation, so this query process around 688.13 GB for the data analysis.

Length of Insertion/Deletion

```
QUERY = """
SELECT
  CASE
    WHEN LENGTH(alternate_bases) -
         LENGTH(reference_bases) > 50
      THEN 51
    WHEN LENGTH(alternate_bases) -
         LENGTH(reference_bases) < -50
      THEN -51
    ELSE
      LENGTH(alternate_bases) -
      LENGTH(reference_bases)
END AS length,
  COUNT(1) AS cnt
FROM
     FLATTEN((SELECT
         reference_name,
         reference_bases,
         alternate_bases,
         vt,
         NTH(1, call.genotype) WITHIN call AS first_allele,
         NTH(2, call.genotype) WITHIN call AS second_allele
       FROM [genomics-public-data:1000_genomes.variants])
     , call)
WHERE
      first_allele =
      POSITION(alternate_bases)
  AND LENGTH(alternate_bases) -
      LENGTH(reference_bases) != 0
OMIT RECORD IF
      reference_name IN ("X", "Y", "MT")
  AND vt != "INDEL"
GROUP BY
  length
  """
```

Figure 53 This query is showing the length of insertion/deletion.

First, The query selects all chromosomes, alternate bases, reference bases, and variant types. Moreover, It also flat with "call" to create "first_allele" and "second_allele" columns. Next, we use "WHERE" clause to find a base length that not equal to zero and then removes records if they belong to "X", "Y", and "MT" chromosome by using "OMIT RECORD IF". We create value in length column by using "CASE" statements(e.g. if the length of alternative bases minus length of reference bases is greater than 50, then print the number 51) Finally, The "GROUP BY" function performed on length columns and "COUNT" function count each record. Since the query process has to perform "FLATTEN" function through all variants, so this query process around 688.13GB for the data analysis.

Frequency of Variant Types Per Chromosome

```
QUERY = """
SELECT
  INTEGER(reference_name) AS chromosome,
  vt AS variant_type,
  COUNT(1) AS cnt
 FROM
   [genomics-public-data:1000_genomes.variants]
 OMIT RECORD IF
   reference_name IN ("X", "Y", "MT")
 GROUP BY
   chromosome,
   variant_type
   """
df = client.query(QUERY, job_config=bigquery.QueryJobConfig(use_legacy_sql = True)).to_dataframe()
df
```

1 to 25 of 66 entries Filter

index	chromosome	variant_type	cnt
0	6	INDEL	92844
1	15	INDEL	41178
2	18	SV	395
3	11	SV	724
4	5	SNP	2438833
5	3	INDEL	98512
6	9	INDEL	57263

Figure 54 This query is showing the frequency of variant types per chromosome.

The query selects all chromosomes and variant_type. However, we remove records If they belong to"X", "Y", and "MT" chromosome by using "OMIT RECORD IF" before performing aggregation operation to group and count for each of those groups. Since the query process has to perform aggregation operation to count the number of variant types per chromosome, so this query process around 0.33 GB for the data analysis.

SNP distribution in Genomes

```
QUERY= """
SELECT
  variant_info.genome AS genome,
  CONCAT(SUBSTR(sample_info.population_description,
                0, 20), "...") AS population,
  sample_info.super_population_description
                AS super_population,
  SUM(variant_info.single) AS cnt1,
  SUM(variant_info.double) AS cnt2
FROM (
  FLATTEN((
  SELECT
    call.call_set_name AS genome,
    SOME(call.genotype > 0) AND NOT EVERY(call.genotype > 0) WITHIN call AS single,
    EVERY(call.genotype > 0) WITHIN call AS double,
   FROM [genomics-public-data:1000_genomes.variants]
   OMIT RECORD IF
     reference_name IN ("X", "Y", "MT"))
  , call)
  ) AS variant_info
  JOIN
    [genomics-public-data:1000_genomes.sample_info] AS sample_info
  ON
    variant_info.genome = sample_info.sample
GROUP BY
  genome,
  population,
  super_population
  """
```

Figure 55 This query is showing SNP distribution in genomes.

The query use “SOME”, “NOT EVERY”, and “EVERY” function to create “single” and “double” columns from “call” if a row of the “genotype” from “call” matched the condition. Moreover, we remove records If they belong to“X”, “Y”, and “MT” chromosome by using “OMIT RECORD IF” function. Next, we perform “JOIN” operation with the flattened records. After that, we use “SUBSTR” function on the population description that has text length is greater than 20 and concatenate the filtered population description with “...” string. In addition, we use “SUM” function to both “single” and “double” column to create SNP distribution, and we use “GROUP BY” method to group rows that have the same values based on genome, population, and super_population column Since the query process has to perform “JOIN” and “FLATTEN” operation, so this query process around 1722.62 GB for the data analysis.

Understanding Alternate Alleles in 1,000 Genomes VCF Data

Is (reference_ name, start, reference_ bases) a unique key in the 1,000 Genomes Data?

No, by the table of results, we can observe that there are several types in reference_base column and different start positions which means the variants may occur in different bases and location.

```
# Find variants on chromosome 17 that reside on the same start with the same reference base
Query = """
SELECT
  reference_name,
  start,
  reference_bases,
  COUNT(start) AS num_alternates
FROM
  [genomics-public-data:1000_genomes.variants]
WHERE
  reference_name = '17'
GROUP BY
  reference_name,
  start,
  reference_bases
HAVING
  num_alternates > 1
ORDER BY
  reference_name,
  start,
  reference_bases
  """
```

1 to 25 of 417 entries Filter

index	reference_name	start	reference_bases	num_alternates
0	17	184672	G	2
1	17	211031	C	2
2	17	240039	G	2
3	17	443435	A	2
4	17	533535	A	2
5	17	557990	A	2
6	17	710852	T	2

Figure 56 This query is showing a total amount of variants are called on Chr17.

The "WHERE" clause applies the condition to individual rows for selecting "Chr17". Before the rows are summarized into groups by the "GROUP BY" clause, "COUNT" function count on starting position as "num_alternates". The "HAVING" clause applies the condition where "num_alternates" is greater than one to the groups after the rows are grouped into groups. The query tries to get only Chr17 however, the process has to scan all records and then apply the condition to the records, so this query process around 0.64 GB for the data analysis.

Alternate alleles

After we find variants on chromosome 17, the single alternate allele is more than two alternate alleles. Then, observing the specific start location, which is two alternate alleles. Comparing with the reference base, we can know the change of alternate allele. In variant type column has two values, SNP and INDEL. INDEL(Insertion Deletion) represents this alternate is more than one base pair but smaller than 50 bp. SNP(Single Nucleotide Polymorphism) represents it only one base pair has alternate. Through the "alt" column, we can get the results which in "vt" column.

```
QUERY = """
# Get three particular start on chromosome 17 that have alternate variants.
SELECT
  reference_name,
  start,
  reference_bases,
  GROUP_CONCAT(alternate_bases) WITHIN RECORD AS alt,
  GROUP_CONCAT(names) WITHIN RECORD AS names,
  vt,
FROM
  [genomics-public-data:1000_genomes.variants]
WHERE
  reference_name = '17'
  AND (start = 48515942
    OR start = 48570613
    OR start = 48659342)
ORDER BY
  start,
  reference_bases,
  alt
    """
```

1 to 6 of 6 entries Filter

index	reference_name	start	reference_bases	alt	names	vt
0	17	48515942	T	G	rs8076712,rs8076712	SNP
1	17	48515942	T	TG	rs113432301,rs113432301	INDEL
2	17	48570613	A	AT	rs201827568,rs201827568	INDEL
3	17	48570613	A	T	rs9896330,rs9896330	SNP
4	17	48659342	C	CTGGT	rs148905490,rs148905490	INDEL
5	17	48659342	C	T	rs113983760,rs113983760	SNP

Show 25 per page

Figure 57 This query is showing an alternative base, rs number(assertion ID in dbSNP), and variant type of called variant on Chr17.

The "WHERE" clause applies the multiple conditions to individual rows for selecting "Chr17" and having a starting position equal to one of 48515942, 48570613, or 48659342. The "GROUP_CONCAT" function returns a string result with the concatenated values from a group. The "ORDER BY" clause, sort all filtered records based on "start", "reference_bases", and "alt" column. The query tries to get the only Chr17; however, the process has to scan all records and then apply the condition to the records. Moreover, "SELECT" function select multiple string columns, so this query process around 1.95 GB for the data analysis.

A unique (reference_ name, start, reference_ bases) tuple, are the variants always SNPs?

Observably, the answer is no because we can see that except SNP, there has INDEL and SV (Structural variation) which is the alternate segment that is greater than 50 bp. Through this analysis, we can understand that the change of variants is not a single type.

```
QUERY= """
# Count by variant type the number of variants on chromosome 17 unique for a
# start and reference base
SELECT
  vt,
  COUNT(vt) AS num_variant_type
FROM
  [genomics-public-data:1000_genomes.variants] AS variants
JOIN EACH (
  SELECT
    reference_name,
    start,
    reference_bases,
    COUNT(start) AS num_alternates
  FROM
    [genomics-public-data:1000_genomes.variants]
  WHERE
    reference_name = '17'
  GROUP EACH BY
    reference_name,
    start,
    reference_bases
  HAVING
    num_alternates = 1) AS singles
ON
  variants.reference_name = singles.reference_name
  AND variants.start = singles.start
  AND variants.reference_bases = singles.reference_bases
WHERE
  variants.reference_name = '17'
GROUP EACH BY
  vt
ORDER BY
  vt
"""
```

1 to 3 of 3 entries Filter

index	vt	num_variant_type
0	INDEL	38754
1	SNP	1006702
2	SV	443

Show 25 per page

Figure 58 This query is proving that unique (reference_ name, start, reference_ bases) tuple on Chr17's variants are not always SNP.

Firstly, create a variant table the same as "Finding variants on chromosome 17 that reside on the same start with the same reference base" query and name it as "singles". Then use the "JOIN" function to combine rows from "singles" and "variants" table and comparing rows within the same table. After we get a unique (reference_name, start, reference_bases) tuple, we count a variant type and then perform "GROUP BY" clause following by "ORDER BY" clause. The query tries to get the only Chr17; however, the process used "JOIN" operation with the conditions to establish a connection between two tables. Moreover, "WHERE" function is used later after "JOIN" function, so this query process around 0.84 GB for the data analysis.

Reproducing the output of vcfstats

How many private variants each sample has?

In this table, only HG00152 has 3 private variants and HG00236 has 2 private variants. Other samples are one. In this case, it means that the common change of BRCA1 may be one variant. First, the query use case state-

```
QUERY = """
# Compute the number of variants within BRCA1 for a particular sample that are shared by
# no other samples.
SELECT
  COUNT(sample_id) AS private_variants_count,
  sample_id
FROM
  (
  SELECT
    reference_name,
    start,
    reference_bases,
    IF(0 < call.genotype,
      call.call_set_name,
      NULL) AS sample_id,
    SUM(IF(0 < call.genotype,
        1,
        0)) WITHIN RECORD AS num_samples_with_variant
  FROM
    [genomics-public-data:1000_genomes.variants]
  WHERE
    reference_name = '17'
    AND start BETWEEN 41196311
    AND 41277499
  HAVING
    num_samples_with_variant = 1
    AND sample_id IS NOT NULL)
GROUP EACH BY
  sample_id
ORDER BY
  sample_id
  """
  #print(bq_assistant.estimate_query_size(QUERY))
df = client.query(QUERY , job_config=bigquery.QueryJobConfig(use_legacy_sql = True)).to_dataframe()
df
```

1 to 25 of 187 entries Filter

index	private_variants_count	sample_id
0	1	HG00106
1	1	HG00109
2	1	HG00143
3	3	HG00152
4	1	HG00160
5	1	HG00186
6	1	HG00231

Figure 59 This query computes the number of variants within BRCA1 for a particular sample that is shared by no other samples.

ment(e.g "IF") goes through conditions (e.g "call.genotype") and returns a value as "sample_id" when the first condition is met. Moreover, another column, namely "num_samples_with_variant", use SUM from the returned value, and then apply "HAVING" function with a particular condition (e.g "num_samples_with_variant = 1") to exclude the variants shared by other samples. Finally, we count the remaining variant as a private variant, and using "GROUP BY" function to return one record for each group. The query tries to get the only BRCA1 gene; however, the process selects multiple columns at first stage use "IF" operation to print out the "call.call_set_name" as a "sample_id", so this query process around 1078.62 GB for the data analysis.

Reproducing 1,000 Genomes allele frequencies for variants in BRCA1

Computes the frequency of both the reference and alternate SNPs within BRCA1 for all samples within 1,000 Genomes.

Through analyzed the percentage of alternate SNPs, we can know that the frequency of each variant in different SNP position.

```
QUERY = """
# The following query computes the allelic frequency for BRCA1 variants in the
# 1,000 Genomes dataset and also includes the pre-computed value from the dataset.
SELECT
  reference_name,
  start,
  reference_bases,
  alternate_bases,
  SUM(ref_count)+SUM(alt_count) AS num_sample_alleles,
  SUM(ref_count) AS ref_cnt,
  SUM(alt_count) AS alt_cnt,
  SUM(ref_count)/(SUM(ref_count)+SUM(alt_count)) AS ref_freq,
  SUM(alt_count)/(SUM(ref_count)+SUM(alt_count)) AS alt_freq,
  alt_freq_from_1KG
FROM (
  SELECT
    reference_name,
    start,
    reference_bases,


          call.call_set_name,
          call.genotype,
        FROM
          [genomics-public-data:1000_genomes.variants]
        WHERE
          reference_name = '17'
          AND start BETWEEN 41196311
          AND 41277499
          AND vt='SNP'
          ),
        call),
      alt))
GROUP BY
  reference_name,
  start,
  reference_bases,
  alternate_bases,
  alt,
  alt_freq_from_1KG
ORDER BY
  reference_name,
  start,
  reference_bases,
  alt,
  alternate_bases
    """

#print(bq_assistant.estimate_query_size(QUERY))
df = client.query(QUERY , job_config=bigquery.QueryJobConfig(use_legacy_sql = True)).to_dataframe()
df
```

1 to 25 of 879 entries Filter

index	reference_name	start	END	reference_bases	alt	vt	chi_squared_score	total_count	hom_ref_count	expected_hom_ref_count	het_count	expected_het_count	hom_
0	17	41196362	41196363	C	T	SNP	34.473	1092	1082	1081.03	9	10.94	
1	17	41196367	41196368	C	T	SNP	0.0	1092	1091	1091.0	1	1.0	
2	17	41196371	41196372	T	C	SNP	0.0	1092	1091	1091.0	1	1.0	
3	17	41196402	41196403	A	G	SNP	0.0	1092	1091	1091.0	1	1.0	
4	17	41196407	41196408	G	A	SNP	2.782	1092	529	517.17	445	468.66	
5	17	41196581	41196582	C	T	SNP	0.028	1092	1081	1081.03	11	10.94	
6	17	41196624	41196625	G	C	SNP	0.0	1092	1091	1091.0	1	1.0	

Figure 60 This query computes the frequency of both the reference and alternate SNPs within BRCA1 for all samples within 1,000 Genomes.

Firstly, The process selects the "BRCA1" gene by using multiple "WHERE" clause on starting position, variant type, and chromosome position. Next, using nested "FLATTEN" operation to get the summation of all reference genotype and alternative genotype. After that, The "SUM" operation is performed on "ref_count" and "alt_count" columns, and the formula to calculate allele frequency is applied to generate "ref_freq" and "alt_freq" column as a final result. Finally, we also perform sorting on the fetched data. The query tries to get the only BRCA1 gene; however, the process use nested "FROM" as well as "FLATTEN", so this query process around 689.10 GB for the data analysis.

Reproducing the Hardy-Weinberg Equilibrium test

```
QUERY = """

SELECT
  reference_name,
  start,
  END,
  reference_bases,
  alt,
  vt,
  ROUND(POW(hom_ref_count - expected_hom_ref_count,
      2)/expected_hom_ref_count +
    POW(hom_alt_count - expected_hom_alt_count,
      2)/expected_hom_alt_count +
    POW(het_count - expected_het_count,
      2)/expected_het_count,
    3) AS chi_squared_score,
  total_count,
  hom_ref_count,
  ROUND(expected_hom_ref_count,
    2) AS expected_hom_ref_count,
  het_count,
  ROUND(expected_het_count,
    2) AS expected_het_count,
  hom_alt_count,
  ROUND(expected_hom_alt_count,
    2) AS expected_hom_alt_count,
  ROUND(alt_freq,
    4) AS alt_freq,
  alt_freq_from_1KG,
FROM (
  SELECT
    reference_name,
    start,
    END,
    reference_bases,
    alt,
    vt,
    alt_freq_from_1KG,
    hom_ref_freq + (.5 * het_freq) AS hw_ref_freq,
    1 - (hom_ref_freq + (.5 * het_freq)) AS alt_freq,
    POW(hom_ref_freq + (.5 * het_freq),
      2) * total_count AS expected_hom_ref_count,
    POW(1 - (hom_ref_freq + (.5 * het_freq)),
      2) * total_count AS expected_hom_alt_count,
    2 * (hom_ref_freq + (.5 * het_freq))
    * (1 - (hom_ref_freq + (.5 * het_freq)))
    * total_count AS expected_het_count,
    total_count,
    hom_ref_count,
    het_count,
    hom_alt_count,
    hom_ref_freq,
    het_freq,
    hom_alt_freq,
```

```
FROM (
  SELECT
    reference_name,
    start,
    END,
    reference_bases,
    alt,
    vt,
    alt_freq_from_1KG,
    # 1000 genomes data IS bi-allelic so there IS only ever a single alt
    # We also exclude calls _where one _or both alleles were NOT called (-1)
    SUM((0 = first_allele
        OR 1 = first_allele)
      AND (0 = second_allele
        OR 1 = second_allele)) WITHIN RECORD AS total_count,
    SUM(0 = first_allele
      AND 0 = second_allele) WITHIN RECORD AS hom_ref_count,
    SUM((0 = first_allele
        AND 1 = second_allele)
      OR (1 = first_allele
        AND 0 = second_allele)) WITHIN RECORD AS het_count,
    SUM(1 = first_allele
      AND 1 = second_allele) WITHIN RECORD AS hom_alt_count,
    SUM(0 = first_allele
      AND 0 = second_allele) / SUM((0 = first_allele
        OR 1 = first_allele)
      AND (0 = second_allele
        OR 1 = second_allele)) WITHIN RECORD AS hom_ref_freq,
    SUM((0 = first_allele
        AND 1 = second_allele)
      OR (1 = first_allele
        AND 0 = second_allele)) / SUM((0 = first_allele
        OR 1 = first_allele)
      AND (0 = second_allele
        OR 1 = second_allele)) WITHIN RECORD AS het_freq,
    SUM(1 = first_allele
      AND 1 = second_allele) / SUM((0 = first_allele
        OR 1 = first_allele)
      AND (0 = second_allele
        OR 1 = second_allele)) WITHIN RECORD AS hom_alt_freq,
  FROM (
    SELECT
      reference_name,
      start,
      END,
      reference_bases,
      GROUP_CONCAT(alternate_bases) WITHIN RECORD AS alt,
      vt,
      # Also return the pre-computed allelic frequency to help us check our work
      af AS alt_freq_from_1KG,
      NTH(1,
        call.genotype) WITHIN call AS first_allele,
      NTH(2,
        call.genotype) WITHIN call AS second_allele,
    FROM
```

```
        [genomics-public-data:1000_genomes.variants]
      WHERE
        reference_name = '17'
        AND start BETWEEN 41196311
        AND 41277499
        )))
ORDER BY
  reference_name,
  start
    """

#print(bq_assistant.estimate_query_size(QUERY))
df = client.query(QUERY , job_config=bigquery.QueryJobConfig(use_legacy_sql = True)).to_dataframe()
df
```

1 to 25 of 879 entries Filter

index	reference_name	start	END	reference_bases	alt	vt	chi_squared_score	total_count	hom_ref_count	expected_hom_ref_count	het_count	expected_het_count	hom_
0	17	41196362	41196363	C	T	SNP	34.473	1092	1082	1081.03	9	10.94	
1	17	41196367	41196368	C	T	SNP	0.0	1092	1091	1091.0	1	1.0	
2	17	41196371	41196372	T	C	SNP	0.0	1092	1091	1091.0	1	1.0	
3	17	41196402	41196403	A	G	SNP	0.0	1092	1091	1091.0	1	1.0	
4	17	41196407	41196408	G	A	SNP	2.782	1092	529	517.17	445	468.66	
5	17	41196581	41196582	C	T	SNP	0.028	1092	1081	1081.03	11	10.94	
6	17	41196624	41196625	G	C	SNP	0.0	1092	1091	1091.0	1	1.0	

Figure 61 This query computes the Hardy-Weinberg Equilibrium for the variants found in BRCA1 and then computing the chi-squared score for the observed versus expected counts for the calls

Firstly, The process selects “BRCA1” gene by using multiple “WHERE” clause on starting position and “reference_name” which is a chromosome position. Next, the “SUM” operation, “OR” operator, and “AND” operator is used to get a homozygous and heterozygous frequency. Moreover, they also exclude calls where one or both alleles were not called (e.g. -1). After that, they apply the hardy-weinberg equilibrium equation and then the chi-squared equation to the selected value. Finally, they apply “ORDER BY” function to sort the results based on “reference_name” and “start” columnThe query tries to get the only BRCA1 gene; however, the process use nested “FROM”, so this query process around 689.10 GB for the data analysis.