GWAS Tutorial

1. Setting up work environment

The first step of performing a GWAS is to load in our depedencies and set up our work environment.

In [1]:
```
"""
Import statements allow us to reuse code written previously by ourselves or others.
Here we are importing the "Hail" library which is the core strategy we are going to be using to organize our data and to eventually perform statistical analyses.
"""
import hail as hl
from hail.plot import show
from pprint import pprint
import ipywidgets as widgets
from IPython.display import display, clear_output
%matplotlib inline
hl.stop()
hl.plot.output_notebook()
hl.init()
```

BokehJS 1.4.0 successfully loaded.

```
2023-02-02 21:35:17.553 WARN  NativeCodeLoader:60 - Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Running on Apache Spark version 3.1.3
SparkUI available at http://hub-testing.c.metal-bonus-375300.internal:4040
Welcome to
     __  __     <>__
    / /_/ /__  __/ /
   / __  / _ `/ / /
  /_/ /_/\_,_/_/_/   version 0.2.108-fc03e9d5dc08
LOGGING: writing to /home/jupyter-test-user/easyGWAS/notebooks/hail-20230202-2135-0.2.108-fc03e9d5dc08.log
```

2. Loading in the data

After we finish loading our dependencies, we can go ahead and start loading the data, starting with our genotype data (stored in a folder called "1kg.mt") and our phenotype data (stored in a file called "1kg_annotations.txt").

In [2]:
```
# Loading in the genotype data from our "data" folder and storing it in a variable called "mt", short for "MatrixTable" (one of the key innovations of the Hail library)
mt = hl.read_matrix_table('data/1kg.mt')

# Loading in the phenotype data from our "data" folder and storing it in a variable called "table"
table = hl.import_table('data/1kg_annotations.txt', impute=True).key_by('Sample')
```

```
2023-02-02 21:35:29.935 Hail: INFO: wrote table with 3501 rows in 1 partition to /tmp/persist_table5qwLADlMpD
2023-02-02 21:35:31.958 Hail: INFO: Reading table to impute column types
2023-02-02 21:35:33.654 Hail: INFO: Finished type imputation
  Loading field 'Sample' as type str (imputed)
  Loading field 'Population' as type str (imputed)
  Loading field 'SuperPopulation' as type str (imputed)
  Loading field 'isFemale' as type bool (imputed)
  Loading field 'PurpleHair' as type bool (imputed)
  Loading field 'CaffeineConsumption' as type int32 (imputed)
```

Now that our data is loaded in, we can combine the two to form a consolidated dataset containing all the relevant information we are going to use for our analyses.

In [3]:
```
# We can use the "annotate_cols" function to add our phenotype data in the "table" variable
mt = mt.annotate_cols(pheno = table[mt.s])
```

It is always a good idea to take a look at our data to see what format we are working with and the available information we have. One way to do this is by using the "describe" method. An example is shown below:

In [4]:
```
# Describing the format of the "mt variable" using an interactive widget
mt.describe(widget = True)
```

```
VBox(children=(HBox(children=(Button(description='globals', layout=Layout(height='30px', width='65px'), style=…
Tab(children=(VBox(children=(HTML(value='<p><big>Global fields, with one value in the dataset.</big></p>\n<p>C…
```

After running the cell above, we can now interact with the four main components of our dataset (globals, rows, cols, and entries). In Hail, each row consist of one specific genetic variant and each column consist of one specific individual. An entry is an intersection of a row and a column and contains information about a specific variant for a particular individual (such as the genetic call).

Additionally from the interactive cell above, we can see that in the "col" tab, we have two fields that we can access: "s" and "pheno". If we expand the "pheno" field, we can see what information we have available for each individual. In this dataset, we have access to the following variables for each individual: "Population", "SuperPopulation", "isFemale", "PurpleHair", and "CaffeineConsumption".

Feel free to explore the "row" and "entry" tabs to learn more about those parts of our dataset.

3. Quality Control

After we load and explore our dataset, the next step is to perform some quality control (QC) so that we have a clean dataset prior to statistical analysis. In a GWAS, there are quite a few quality control measures we have to do. For this tutorial, we will focus on just a few QC measures. In order to organize ourselves, let's split these quality control measures into two categories: QC on the sample data and QC on the variant data.

Let's begin with QC on the sample (phenotype) data:

1. Remove individuals with high levels of missingness (people who we do not have enough data for)

Let's continue on to QC on the variant (genotype) data:

1. Remove variants with low minor allele frequency (MAF)
2. Remove variants that deviate from Hardy–Weinberg equilibrium (HWE)

Hail has a few QC methods that can help us get started. These methods, **sample_qc** and **variant_qc**, extract quality-related information from our MatrixTable and store them into variables that we can later reference. While we are at it, let's also create another variable (filtered_mt) that will hold the filtered version of our MatrixTable after QC.

In [5]:
```
mt = hl.sample_qc(mt)
mt = hl.variant_qc(mt)
filtered_mt = mt
```

We can now use Hail's filter_rows and filter_cols methods to filter out bad samples and bad variants from our MatrixTable. To make it easier to follow, running the cell below creates sliders that you can drag to select different threshold values for our QC conditions above. Feel free to change the values and then click on the button to apply the QC filters. Notice how the number of samples and variants change depending on our threshold values.

When you are done experimenting, configure the sliders to the following QC values and click on the button:

1. Sample Call Rate = 0.97
2. Minor Allele Frequency = 0.01
3. Hardy Weinberg Equilibrium = 1.00e-6

In [6]:
```
call_rate_slider = widgets.FloatSlider(min=0.90, max=1.00, step=0.01, value=0.97, layout = widgets.Layout(width='500px'), description = "Sample Call Rate:", style=dict(description_width='initial'))
maf_slider = widgets.FloatSlider(min=0.01, max=0.10, step=0.01, value=0.01, layout = widgets.Layout(width='500px'), description = "Minor Allele Frequency:", style=dict(description_width='initial'))
hwe_slider = widgets.FloatLogSlider(value=6, base=10, min=-10, max=-6, step=1, readout_format='.2e', layout = widgets.Layout(width='500px'), description = "Hardy Weinberg Equilbrium:", style=dict(description_width='initial'))
output = widgets.Output()
button = widgets.Button(description = "Apply QC filter", button_style = "primary")

display(call_rate_slider, maf_slider, hwe_slider)
display(button)

def on_button_click(a):
    global mt
    global filtered_mt
    filtered_mt = mt
    with output:
        clear_output()
        call_rate_value = call_rate_slider.value
        maf_value = maf_slider.value
        hwe_value = hwe_slider.value
        filtered_mt = filtered_mt.filter_cols((filtered_mt.sample_qc.dp_stats.mean >= 4) & (filtered_mt.sample_qc.call_rate >= call_rate_value))
        ab = filtered_mt.AD[1] / hl.sum(filtered_mt.AD)
        filter_condition_ab = ((filtered_mt.GT.is_hom_ref() & (ab <= 0.1)) |
                        (filtered_mt.GT.is_het() & (ab >= 0.25) & (ab <= 0.75)) |
                        (filtered_mt.GT.is_hom_var() & (ab >= 0.9)))
        filtered_mt = filtered_mt.filter_entries(filter_condition_ab)
        filtered_mt = filtered_mt.filter_rows(filtered_mt.variant_qc.AF[1] > maf_value)
        filtered_mt = filtered_mt.filter_rows(filtered_mt.variant_qc.p_value_hwe > hwe_value)
        print('After filtering: Samples: %d  Variants: %d' % (filtered_mt.count_cols(), filtered_mt.count_rows()))
button.on_click(on_button_click)
display(output)
```

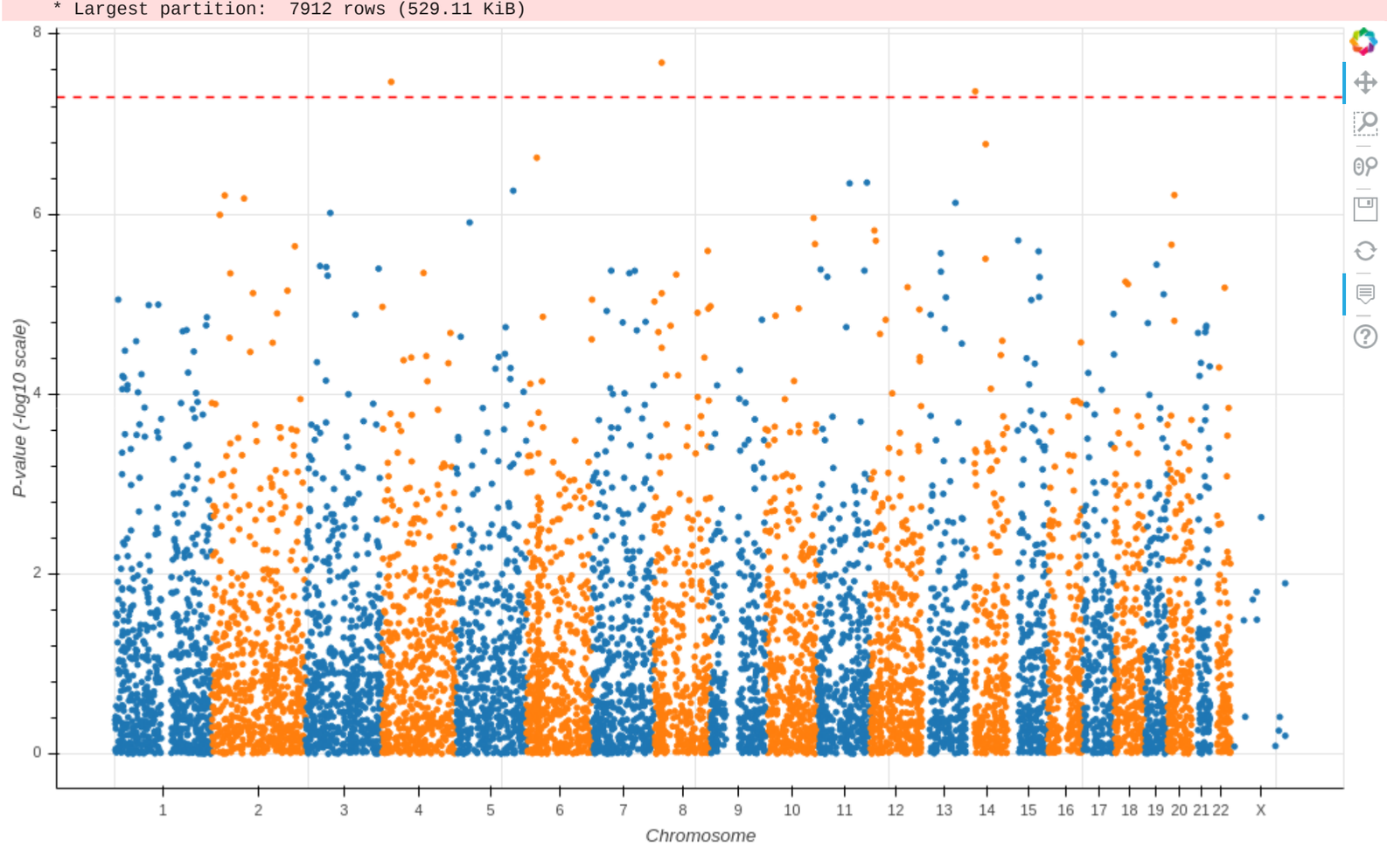
```
FloatSlider(value=0.97, description='Sample Call Rate:', layout=Layout(width='500px'), max=1.0, min=0.9, step=…
FloatSlider(value=0.01, description='Minor Allele Frequency:', layout=Layout(width='500px'), max=0.1, min=0.01…
FloatLogSlider(value=1e-06, description='Hardy Weinberg Equilbrium:', layout=Layout(width='500px'), max=-6.0, …
Button(button_style='primary', description='Apply QC filter', style=ButtonStyle())
Output()
```

4. Initial GWAS

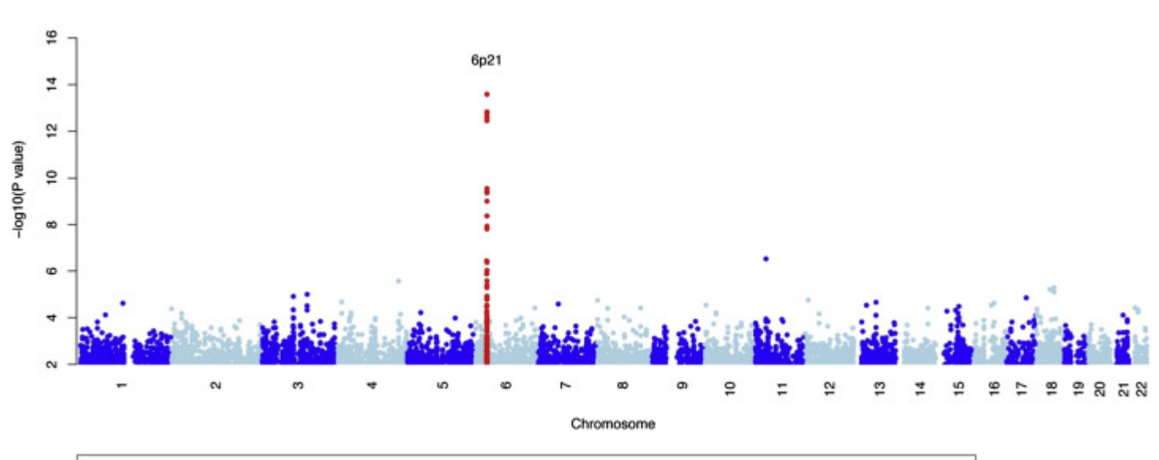
Now that we are done filtering our data, we can go ahead and perform the actual GWAS! In Hail, a GWAS can be performed using the **hl.linear_regression_rows or hl.logistic_regression_rows** methods. The decision to use which method depends on what kind of phenotype we want to investigate. For this tutorial, let us investigate the CaffeineConsumption phenotype (a synthetic phenotype created just for the purpose of this tutorial). As the CaffeineConsumption phenotype is an integer and not a boolean, we will use the **hl.linear_regression_rows** method for our analysis.

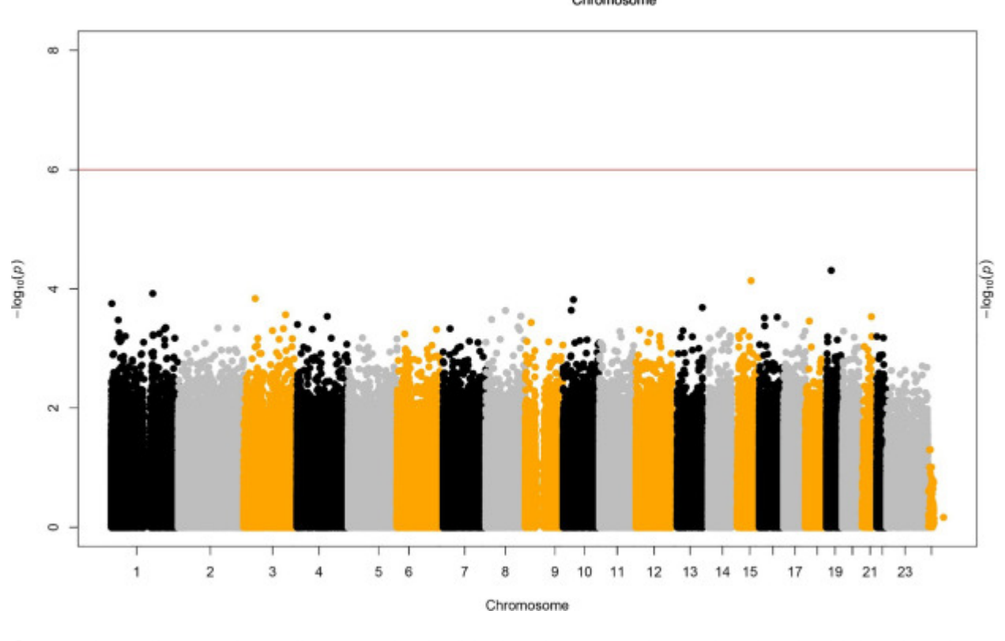
In [7]:
```
gwas = hl.linear_regression_rows(
    y=filtered_mt.pheno.CaffeineConsumption,
    x=filtered_mt.GT.n_alt_alleles(),
    covariates=[1.0, filtered_mt.pheno.isFemale])
p = hl.plot.manhattan(gwas.p_value)
show(p)
```

```
2023-02-02 21:36:19.555 Hail: INFO: linear_regression_rows: running on 250 samples for 1 response variable y,
    with input variable x, and 2 additional covariates...
2023-02-02 21:36:23.680 Hail: INFO: wrote table with 7912 rows in 1 partition to /tmp/persist_tableCLnlM4sTWU
    Total size: 529.13 KiB
    * Rows: 529.11 KiB
    * Globals: 11.00 B
    * Smallest partition: 7912 rows (529.11 KiB)
    * Largest partition:  7912 rows (529.11 KiB)
```



The image above is called a Manhattan plot, named after the city skyline of Manhattan, NY. Each point represents one particular variant in our dataset. Variants that have higher y-values are more statistically significant. The dashed horizontal line presents our significance threshold (5e-8). Examples of well controlled GWAS's Manhattan plots from other published studies are provided below:



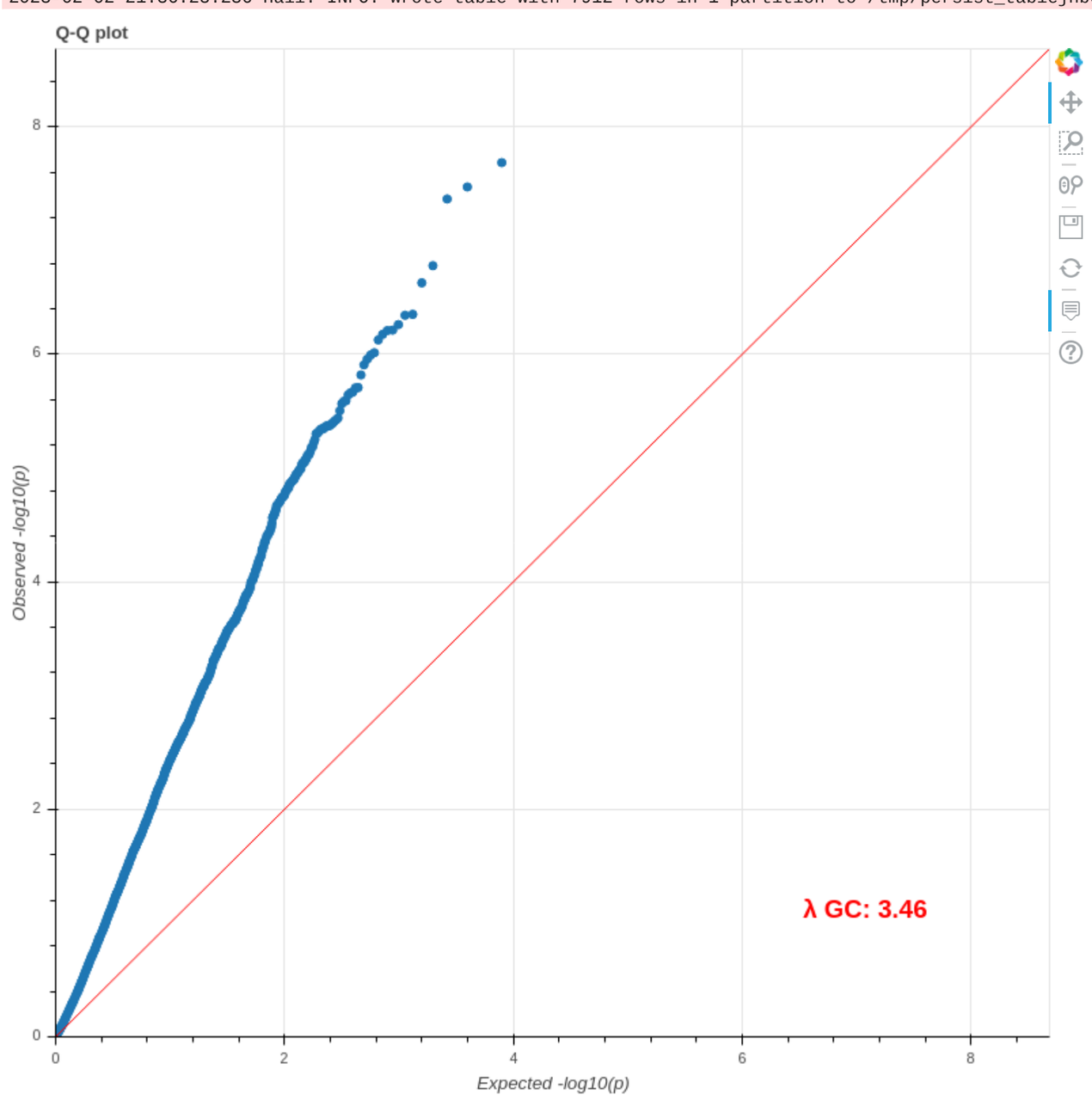


Source: https://www.sciencedirect.com/topics/biochemistry-genetics-and-molecular-biology/manhattan-plot

In both of the examples above, most of the variants are not significant and are tightly packed together. However, this is not what we see in our GWAS. What's wrong? Is there something in our dataset that is adding noise to our GWAS? We can investigate for potential confounding effects by using a Quantile-Quantile Plot.

In [8]:
```
p = hl.plot.qq(gwas.p_value)
show(p)
```

```
2023-02-02 21:36:25.869 Hail: INFO: Ordering unsorted dataset with network shuffle
2023-02-02 21:36:27.158 Hail: INFO: wrote table with 7912 rows in 1 partition to /tmp/persist_table98kOpigCJl
2023-02-02 21:36:28.236 Hail: INFO: wrote table with 7912 rows in 1 partition to /tmp/persist_tablejhb0K2QTZT
```



A Quantile-Quantile plot (or QQ-plot) is a graph that represents of the deviation of the observed p-values from the null hypothesis. The GWAS p-values for each SNP are sorted from largest to smallest and plotted against expected values from a theoretical χ2-distribution. ***If the observed values correspond to the expected values, all points are on or near the middle line between the x-axis and the y-axis (null hypothesis).*** In our case, we see that our points deviate greatly from the middle line. This is evidence of potential confounding present in the dataset.

4. Ancestry, Population Stratification and Principal Component Analysis (PCA)

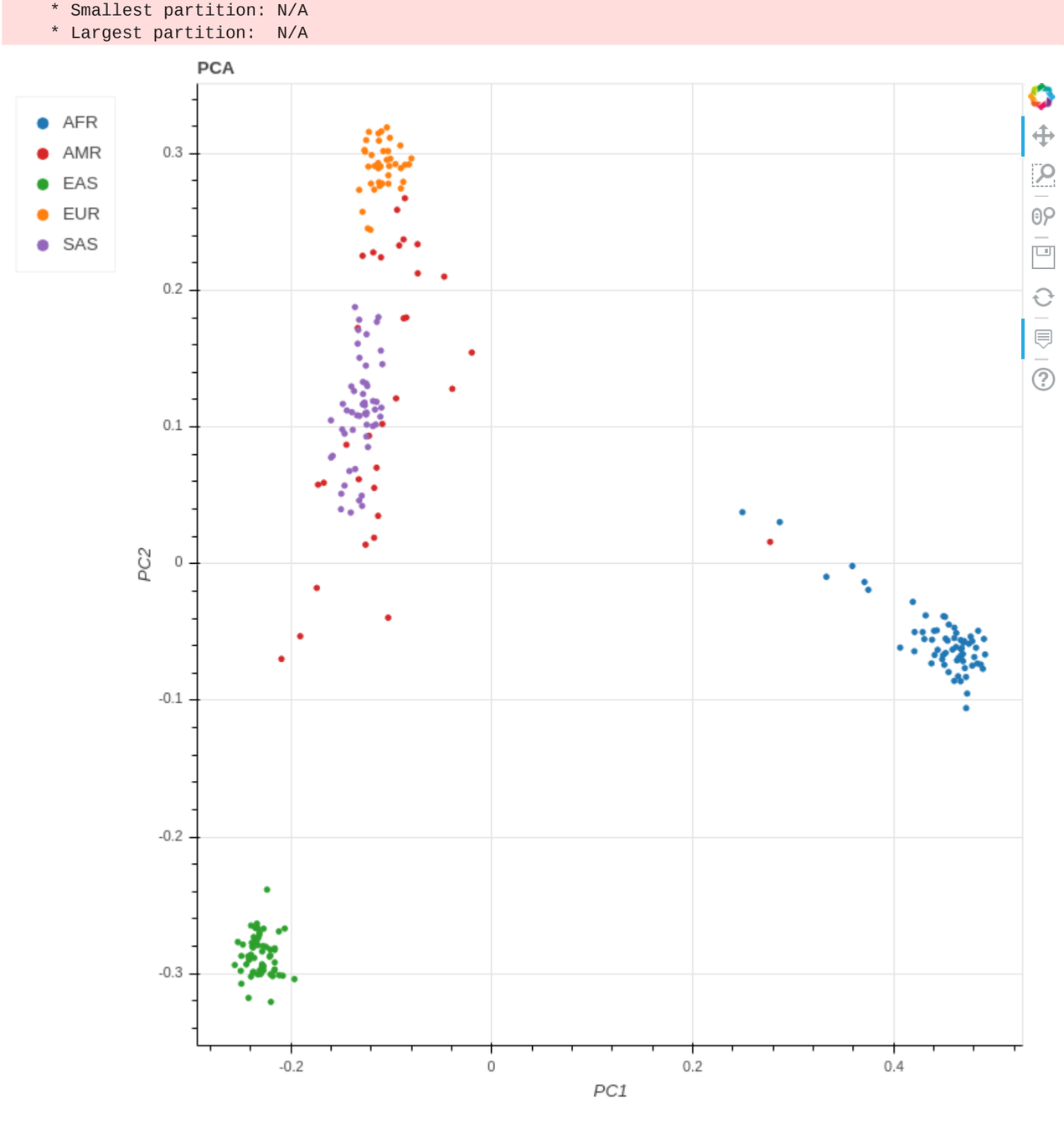
One's ancestry has a large role in determining the genetic variants present in one's genome. Depending on the ancestry, variants can have differing influence on the phenotype being explored. Often times in genomic studies, researchers will control for ancestry in their statistical experiments. We do this in order to account for population stratification (differences in allele frequencies between cases and controls due to systematic differences in ancestry rather than association of genes with disease). One approach to control for ancestry is to perform Principal Component Analysis (PCA).

Hail allows one to easily perform PCA using a built in function. Let's perform PCA and plot the results to see if we notice anything.

In [9]:
```
eigenvalues, pcs, _ = hl.hwe_normalized_pca(filtered_mt.GT)
filtered_mt = filtered_mt.annotate_cols(scores = pcs[filtered_mt.s].scores)

p = hl.plot.scatter(filtered_mt.scores[0],
                    filtered_mt.scores[1],
                    label=filtered_mt.pheno.SuperPopulation,
                    title='PCA', xlabel='PC1', ylabel='PC2')
show(p)
```

```
2023-02-02 21:36:34.483 Hail: INFO: hwe_normalize: found 7902 variants after filtering out monomorphic sites.
2023-02-02 21:36:39.074 Hail: INFO: pca: running PCA with 10 components...) / 1]
2023-02-02 21:36:46.843 Hail: INFO: wrote table with 0 rows in 0 partitions to /tmp/persist_tablegevp8b2qBy
    Total size: 21.32 KiB
    * Rows: 0.00 B
    * Globals: 21.32 KiB
    * Smallest partition: N/A
    * Largest partition:  N/A
```



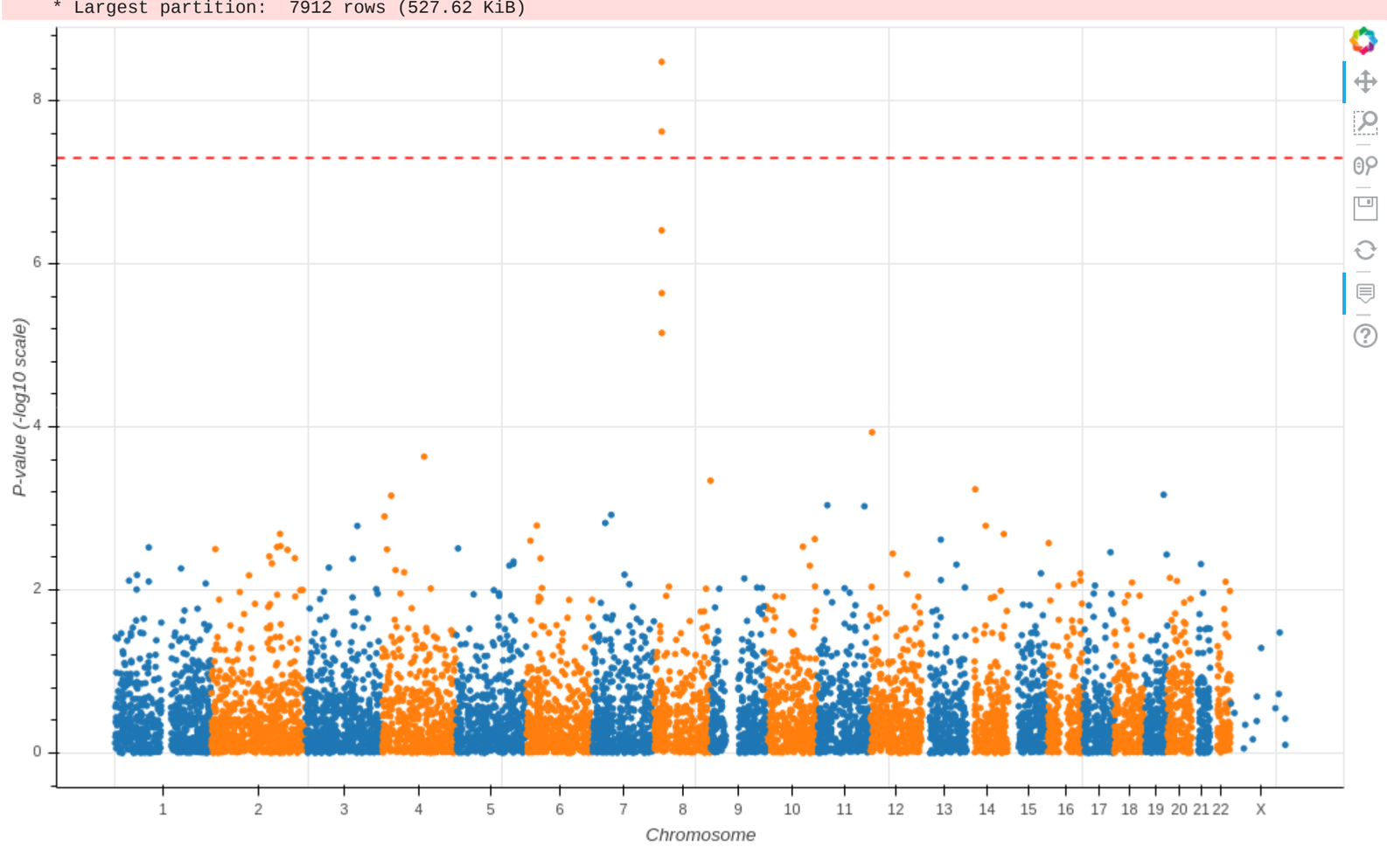
The scatter plot above plots each sample according to their first two principal components and colors the samples by their ancestry SuperPopulation (AFR = Africa, AMR = Admixed American, EAS = Eastern Asian, EUR = European, and SAS = South Asian). As we can see, several clusters emerge. Without going into too much detail, to account for ancestry potentially playing a part in our linear regression, we simply add our calculated principal components as co-variates in our statistical test.

5. Final GWAS controlling for Population Stratification

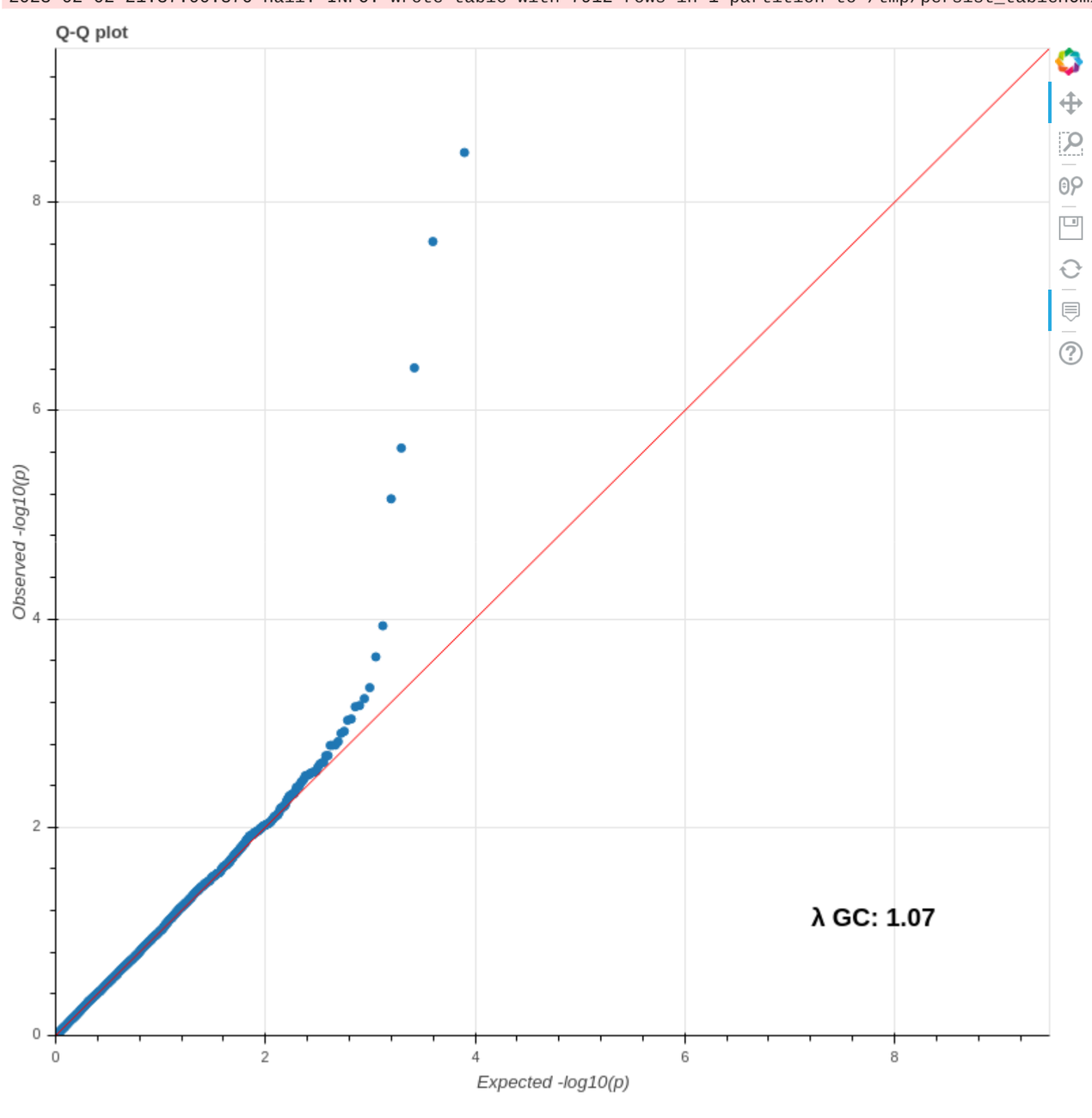
Let us add in the first few principal components as co-variates in our statistical test and re-run our GWAS.

In [10]:
```
gwas = hl.linear_regression_rows(
    y=filtered_mt.pheno.CaffeineConsumption,
    x=filtered_mt.GT.n_alt_alleles(),
    covariates=[1.0, filtered_mt.pheno.isFemale, filtered_mt.scores[0], filtered_mt.scores[1], filtered_mt.scores[2]]) # This is where we added in additional covariates (in our case the principal components)
p = hl.plot.manhattan(gwas.p_value)
show(p)
p = hl.plot.qq(gwas.p_value)
show(p)
```

```
2023-02-02 21:36:56.040 Hail: INFO: linear_regression_rows: running on 250 samples for 1 response variable y,
    with input variable x, and 5 additional covariates...
2023-02-02 21:36:58.221 Hail: INFO: wrote table with 7912 rows in 1 partition to /tmp/persist_tableICZV47nSzY
    Total size: 527.63 KiB
    * Rows: 527.62 KiB
    * Globals: 11.00 B
    * Smallest partition: 7912 rows (527.62 KiB)
    * Largest partition:  7912 rows (527.62 KiB)
```




```
2023-02-02 21:36:59.670 Hail: INFO: Ordering unsorted dataset with network shuffle
2023-02-02 21:37:00.211 Hail: INFO: wrote table with 7912 rows in 1 partition to /tmp/persist_tableXREXkBIAAS
2023-02-02 21:37:00.876 Hail: INFO: wrote table with 7912 rows in 1 partition to /tmp/persist_tableH6mxCFuwMO
```



Now that's a much better looking skyline. From the QQ-plot, we see that most points do fall on the center line. From the Manhattan plot, we see that most variants are not significant but we do have some on Chromosome 8 that have met our significance threshold. Remember this is a synthetic phenotype so there probably are not any variants on Chromosome 8 that are associated with CaffieneConsumption, but it is always nice to see results!

Feel free to use your cursor and hover over the variants to find more information about their chromosomal position and p-value. If you simply want a table with that information, you can run the cell below.

In [11]:
```
gwas.order_by("p_value").show(10)
```

```
2023-02-02 21:37:02.322 Hail: INFO: Ordering unsorted dataset with network shuffle
```

locus	alleles	n	sum_x	y_transpose_x	beta	standard_error	t_stat	p_value
locus<GRCh37>	array<str>	int32	float64	float64	float64	float64	float64	float64
8:19600329	["A","G"]	250	2.72e+02	1.31e+03	7.48e-01	1.22e-01	6.14e+00	3.33e-09
8:19619751	["G","A"]	250	1.07e+02	5.38e+02	8.78e-01	1.52e-01	5.77e+00	2.38e-08
8:19651161	["T","C"]	250	2.09e+02	1.02e+03	6.18e-01	1.18e-01	5.22e+00	3.89e-07
8:19826373	["G","A"]	250	2.61e+02	1.18e+03	6.26e-01	1.29e-01	4.84e+00	2.29e-06
8:19943027	["G","A"]	250	8.22e+01	4.07e+02	9.30e-01	2.03e-01	4.59e+00	7.03e-06
12:4702230	["G","A"]	250	2.13e+01	9.81e+01	1.31e+00	3.34e-01	3.92e+00	1.16e-04
4:108530885	["C","T"]	250	1.87e+02	8.92e+02	5.10e-01	1.36e-01	3.74e+00	2.31e-04
8:145665407	["C","T"]	250	2.40e+02	9.99e+02	-4.76e-01	1.34e-01	-3.55e+00	4.56e-04
14:22133648	["G","A"]	250	3.53e+02	1.49e+03	-5.17e-01	1.48e-01	-3.49e+00	5.82e-04
19:49164944	["G","A"]	250	5.39e+00	8.79e+00	-2.29e+00	6.66e-01	-3.44e+00	6.78e-04

showing top 10 rows

6. Congratulations!

You have just completed an entire GWAS from start to finish! To summarize:

1. We first set up our work environment and loaded in our data
2. We then performed some sample and variant QC to filter our dataset
3. We performed an initial GWAS and were met with noisy results
4. We applied PCA and accounted for ancestry playing a role in impacting our phenotype
5. We performed a final GWAS accounting for ancestry

Feel free to explore some of the other notebooks available or rerun this experiment with your own data!