Genome-Wide Association Study using Hail

By: Bruno Ambrozio (https://about.me/bambrozio) - 18/08/2019

> Credits/Kudos go for Hail, at Docs > Hail Tutorials > GWAS Tutorial (https://hail.is/docs/0.2/tutorials/01-genome-wide-association-study.html)

This notebook consolidates bullet points of the Hail Tutorial main contents. It aims to manipulate and query genetic dataset through a genome-wide SNP association test, and control for confounding caused by population stratification.

Preliminary

- Initiate hail.

In [1]:
```python
import hail as hl
hl.init()
```

```
Running on Apache Spark version 2.4.1
SparkUI available at http://brunos-mbp.mul.ie.ibm.com:4040
Welcome to
     __  __     <>__
    / /_/ /__  __/ /
   / __  / _ `/ / /
  /_/ /_/\_,_/_/_/   version 0.2.25-04344d214361
LOGGING: writing to /Users/bambrozi/workspace/github.com/bambrozio/bioinformatics/hail/hail-20191021-1320-0.
2.25-04344d214361.log
```

In [2]:
```
# Libs
from hail.plot import show
from pprint import pprint
hl.plot.output_notebook()
```

(https://bokeh.pydata.org) BokehJS 1.2.0 successfully loaded.

In [3]:
```
# 1000 Genomes data sample as the dataset
hl.utils.get_1kg('~/Download/hail-gwas-data/')
```

```
2019-10-21 13:20:35 Hail: INFO: 1KG files found
```

- Importing data from VCF (The data in a VCF file is naturally represented as a Hail MatrixTable.)

In [4]:
```
hl.import_vcf('~/Download/hail-gwas-data/1kg.vcf.bgz').write('data/1kg.mt', overwrite=True)
```

```
2019-10-21 13:20:38 Hail: INFO: Coerced sorted dataset
2019-10-21 13:20:41 Hail: INFO: wrote matrix table with 10961 rows and 284 columns in 2 partitions to data/1
kg.mt
```

In [5]:
```
mt = hl.read_matrix_table('data/1kg.mt')
```

- Data overview

In [6]:
```
mt.rows().select().show(5)
```

locus	alleles
locus<GRCh37>	array<str>
1:904165	["G","A"]
1:909917	["G","A"]
1:986963	["C","T"]
1:1563691	["T","G"]
1:1707740	["T","G"]

showing top 5 rows

In [7]:
```
mt.row_key.show(5)
```

locus	alleles
locus<GRCh37>	array<str>
1:904165	["G","A"]
1:909917	["G","A"]
1:986963	["C","T"]
1:1563691	["T","G"]
1:1707740	["T","G"]

showing top 5 rows

In [8]:
```
# Peek at the first few sample IDs:
mt.s.show(5)
```

s
str
"HG00096"
"HG00099"
"HG00105"
"HG00118"
"HG00129"

showing top 5 rows

In [9]:
```
mt.entry.take(5)
```

Out[9]:
```
[Struct(GT=Call(alleles=[0, 0], phased=False), AD=[4, 0], DP=4, GQ=12, PL=[0, 12, 147]),
 Struct(GT=Call(alleles=[0, 0], phased=False), AD=[8, 0], DP=8, GQ=24, PL=[0, 24, 315]),
 Struct(GT=Call(alleles=[0, 0], phased=False), AD=[8, 0], DP=8, GQ=23, PL=[0, 23, 230]),
 Struct(GT=Call(alleles=[0, 0], phased=False), AD=[7, 0], DP=7, GQ=21, PL=[0, 21, 270]),
 Struct(GT=Call(alleles=[0, 0], phased=False), AD=[5, 0], DP=5, GQ=15, PL=[0, 15, 205])]
```

In [10]:
```
mt.entry.show(5)
```

locus	alleles	HG00096.GT	HG00096.AD	HG00096.DP	HG00096.GQ	HG00096.PL	HG00099.GT	HG00099.AD	HG00099.DP	HG00099.
locus<GRCh37>	array<str>	call	array<int32>	int32	int32	array<int32>	call	array<int32>	int32	in
1:904165	["G","A"]	0/0	[4,0]	4	12	[0,12,147]	0/0	[8,0]	8	
1:909917	["G","A"]	0/0	[4,0]	4	12	[0,12,160]	0/0	[8,0]	8	
1:986963	["C","T"]	0/0	[3,0]	3	9	[0,9,98]	0/0	[1,0]	1	
1:1563691	["T","G"]	NA	NA	NA	NA	NA	0/0	[2,0]	2	
1:1707740	["T","G"]	0/1	[2,3]	5	67	[82,0,67]	0/1	[4,2]	6	

showing top 5 rows

showing the first 3 of 284 columns

Pre-processing

- Using a text file to annotate the columns in a MatrixTable
- Imported this file into Hail with `import_table`. This function produces a Table object. (similar to Pandas or R dataframe, but not limited by the memory as it's distributed with Spark).

In [11]:
```
table = (hl.import_table('~/Download/hail-gwas-data/1kg_annotations.txt', impute=True)
         .key_by('Sample'))
table.describe()
```

```
----------------------------------------
Global fields:
    None
----------------------------------------
Row fields:
    'Sample': str
    'Population': str
    'SuperPopulation': str
    'isFemale': bool
    'PurpleHair': bool
    'CaffeineConsumption': int32
----------------------------------------
Key: ['Sample']
----------------------------------------

2019-10-21 13:20:45 Hail: INFO: Reading table to impute column types
2019-10-21 13:20:45 Hail: INFO: Finished type imputation
  Loading column 'Sample' as type 'str' (imputed)
  Loading column 'Population' as type 'str' (imputed)
  Loading column 'SuperPopulation' as type 'str' (imputed)
  Loading column 'isFemale' as type 'bool' (imputed)
  Loading column 'PurpleHair' as type 'bool' (imputed)
  Loading column 'CaffeineConsumption' as type 'int32' (imputed)
```

In [12]:
```
table.show(width=100)
```

Sample	Population	SuperPopulation	isFemale	PurpleHair	CaffeineConsumption
str	str	str	bool	bool	int32
"HG00096"	"GBR"	"EUR"	false	false	4
"HG00097"	"GBR"	"EUR"	true	true	4
"HG00098"	"GBR"	"EUR"	false	false	5
"HG00099"	"GBR"	"EUR"	true	false	4
"HG00100"	"GBR"	"EUR"	true	false	5
"HG00101"	"GBR"	"EUR"	false	true	1
"HG00102"	"GBR"	"EUR"	true	true	6
"HG00103"	"GBR"	"EUR"	false	true	5
"HG00104"	"GBR"	"EUR"	true	false	5
"HG00105"	"GBR"	"EUR"	false	false	4

showing top 10 rows

In [13]:
```
print(mt.col.dtype)
```

```
struct{s: str}
```

- add sample annotations to the dataset through a column in the MatrixTable.
- `annotate_cols` method to join the table with the MatrixTable containing the dataset.

In [14]:
```
mt = mt.annotate_cols(pheno = table[mt.s])
mt.col.describe()
```

```
--------------------------------------------------------
Type:
        struct {
        s: str,
        pheno: struct {
            Population: str,
            SuperPopulation: str,
            isFemale: bool,
            PurpleHair: bool,
            CaffeineConsumption: int32
        }
    }
--------------------------------------------------------
Source:
    <hail.matrixtable.MatrixTable object at 0x11ffa2750>
Index:
    ['column']
--------------------------------------------------------
```

- Query functions and the Hail Expression Language
    - gathering statistics (Expressions as arguments).

In [15]:
```
pprint(table.aggregate(hl.agg.counter(table.SuperPopulation)))
```

```
{'AFR': 1018, 'AMR': 535, 'EAS': 617, 'EUR': 669, 'SAS': 661}
```

In [16]:
```
pprint(table.aggregate(hl.agg.stats(table.CaffeineConsumption)))
```

```
{'max': 10.0,
 'mean': 3.9837142857142855,
 'min': -1.0,
 'n': 3500,
 'stdev': 1.7021055628070711,
 'sum': 13943.0}
```

- These metrics aren’t perfectly representative of the samples in the dataset, because:

```
In [17]: table.count()
Out[17]: 3500
```

```
In [18]: mt.count_cols()
Out[18]: 284
```

- There are fewer samples in the dataset than in the full thousand genomes cohort. Use `aggregate_cols` to get the metrics for only the samples in the dataset

```
In [19]: mt.aggregate_cols(hl.agg.counter(mt.pheno.SuperPopulation))
Out[19]: {'AFR': 76, 'EAS': 72, 'AMR': 34, 'SAS': 55, 'EUR': 47}
```

```
In [20]: pprint(mt.aggregate_cols(hl.agg.stats(mt.pheno.CaffeineConsumption)))

{'max': 9.0,
 'mean': 4.415492957746479,
 'min': 0.0,
 'n': 284,
 'stdev': 1.577763427465917,
 'sum': 1254.0}
```

- Calculate the counts of each of the 12 possible unique SNPs (4 choices for the reference base * 3 choices for the alternate base).
    1. get the alternate allele of each variant
    2. count the occurences of each unique ref/alt pair.

```
In [21]: snp_counts = mt.aggregate_rows(hl.agg.counter(hl.Struct(ref=mt.alleles[0], alt=mt.alleles[1])))
         pprint(snp_counts)
```

```
{Struct(ref='A', alt='C'): 454,
 Struct(ref='C', alt='A'): 496,
 Struct(ref='C', alt='T'): 2436,
 Struct(ref='T', alt='A'): 79,
 Struct(ref='A', alt='T'): 76,
 Struct(ref='G', alt='T'): 480,
 Struct(ref='T', alt='G'): 468,
 Struct(ref='G', alt='A'): 2387,
 Struct(ref='G', alt='C'): 112,
 Struct(ref='C', alt='G'): 150,
 Struct(ref='T', alt='C'): 1879,
 Struct(ref='A', alt='G'): 1944}
```

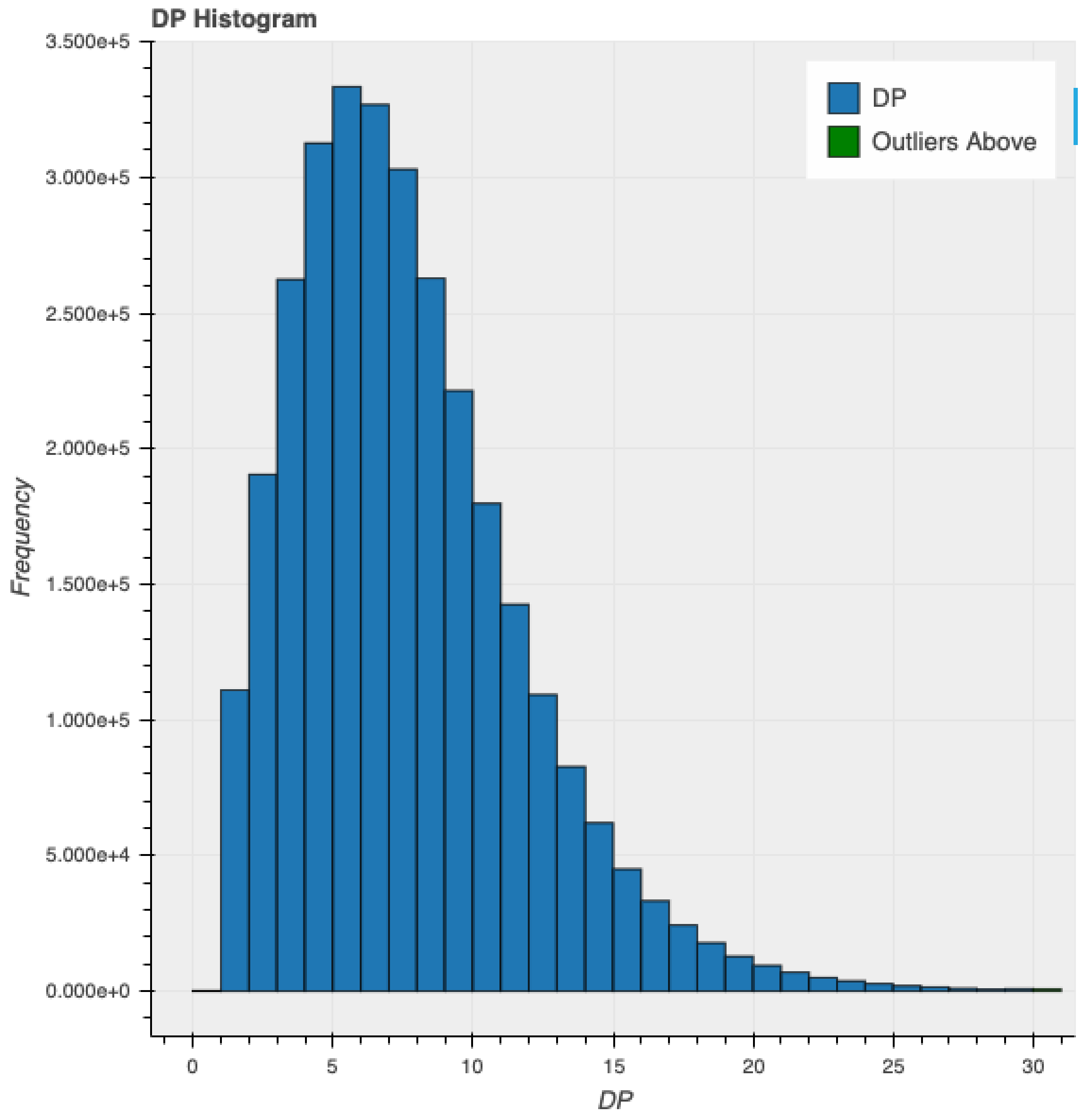
- list the counts in descending order using Python’s Counter class.

```
In [22]: from collections import Counter
         counts = Counter(snp_counts)
         counts.most_common()
```

```
Out[22]: [(Struct(ref='C', alt='T'), 2436),
          (Struct(ref='G', alt='A'), 2387),
          (Struct(ref='A', alt='G'), 1944),
          (Struct(ref='T', alt='C'), 1879),
          (Struct(ref='C', alt='A'), 496),
          (Struct(ref='G', alt='T'), 480),
          (Struct(ref='T', alt='G'), 468),
          (Struct(ref='A', alt='C'), 454),
          (Struct(ref='C', alt='G'), 150),
          (Struct(ref='G', alt='C'), 112),
          (Struct(ref='T', alt='A'), 79),
          (Struct(ref='A', alt='T'), 76)]
```

```
In [ ]:
```

In [23]:
```
p = hl.plot.histogram(mt.DP, range=(0,30), bins=30, title='DP Histogram', legend='DP')
show(p)
```



(http

Quality Control

- sequencing dataset.
- iterative process, and is different for every project.
    - based on the ability to understand the properties of a dataset.
- However, by practicing open science and discussing the QC process and decisions with others, we can establish a set of best practices as a community.
- `sample_qc` function is a Hail function which produces a set of useful metrics and stores them in a column field.

In [24]:
```
mt.col.describe()
```

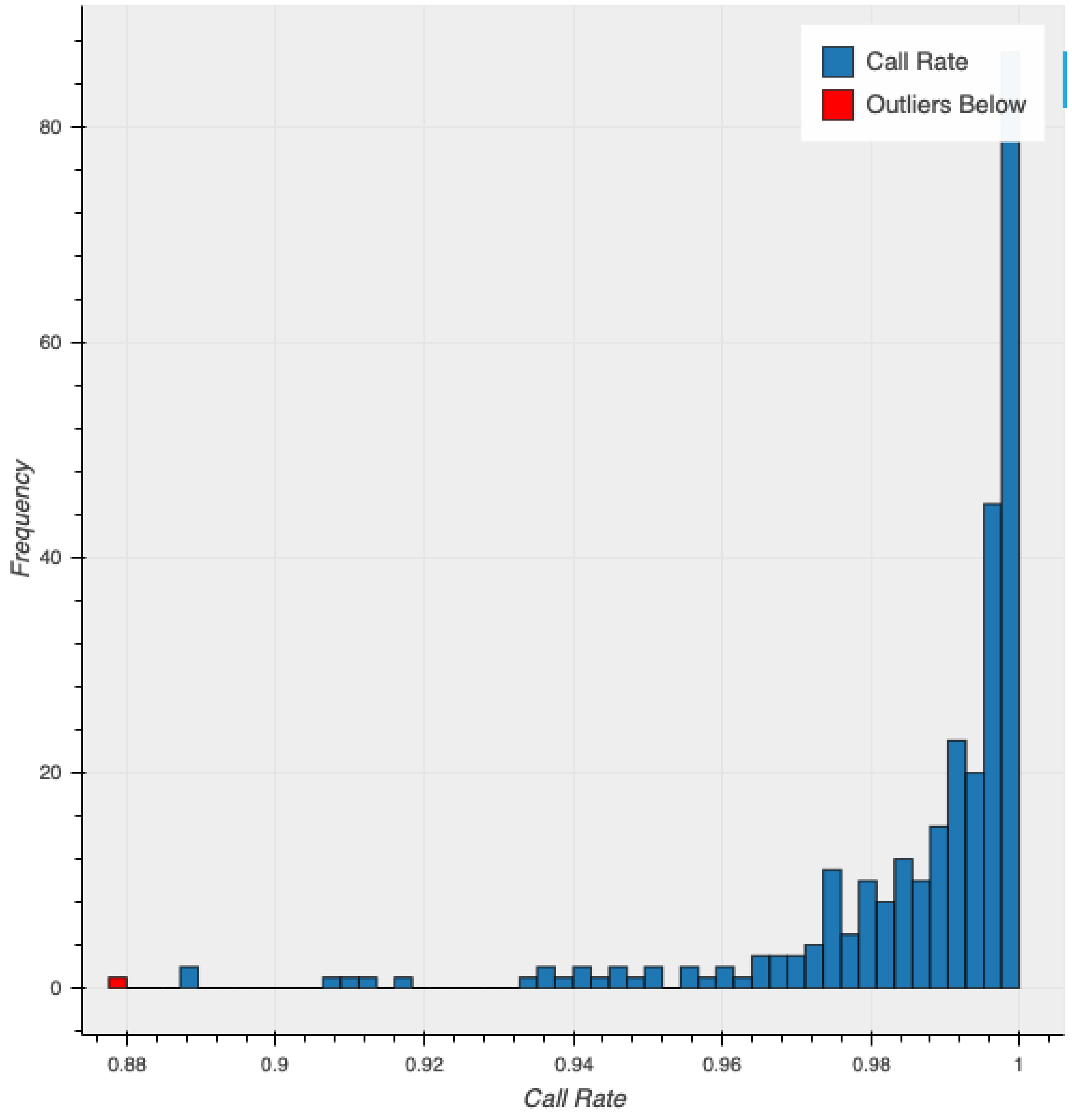
```
--------------------------------------------------------
Type:
        struct {
        s: str,
        pheno: struct {
            Population: str,
            SuperPopulation: str,
            isFemale: bool,
            PurpleHair: bool,
            CaffeineConsumption: int32
        }
    }
--------------------------------------------------------
Source:
    <hail.matrixtable.MatrixTable object at 0x11ffa2750>
Index:
    ['column']
--------------------------------------------------------
```

In [25]:
```
mt = hl.sample_qc(mt)
```

In [26]:
```
mt.col.describe()
```

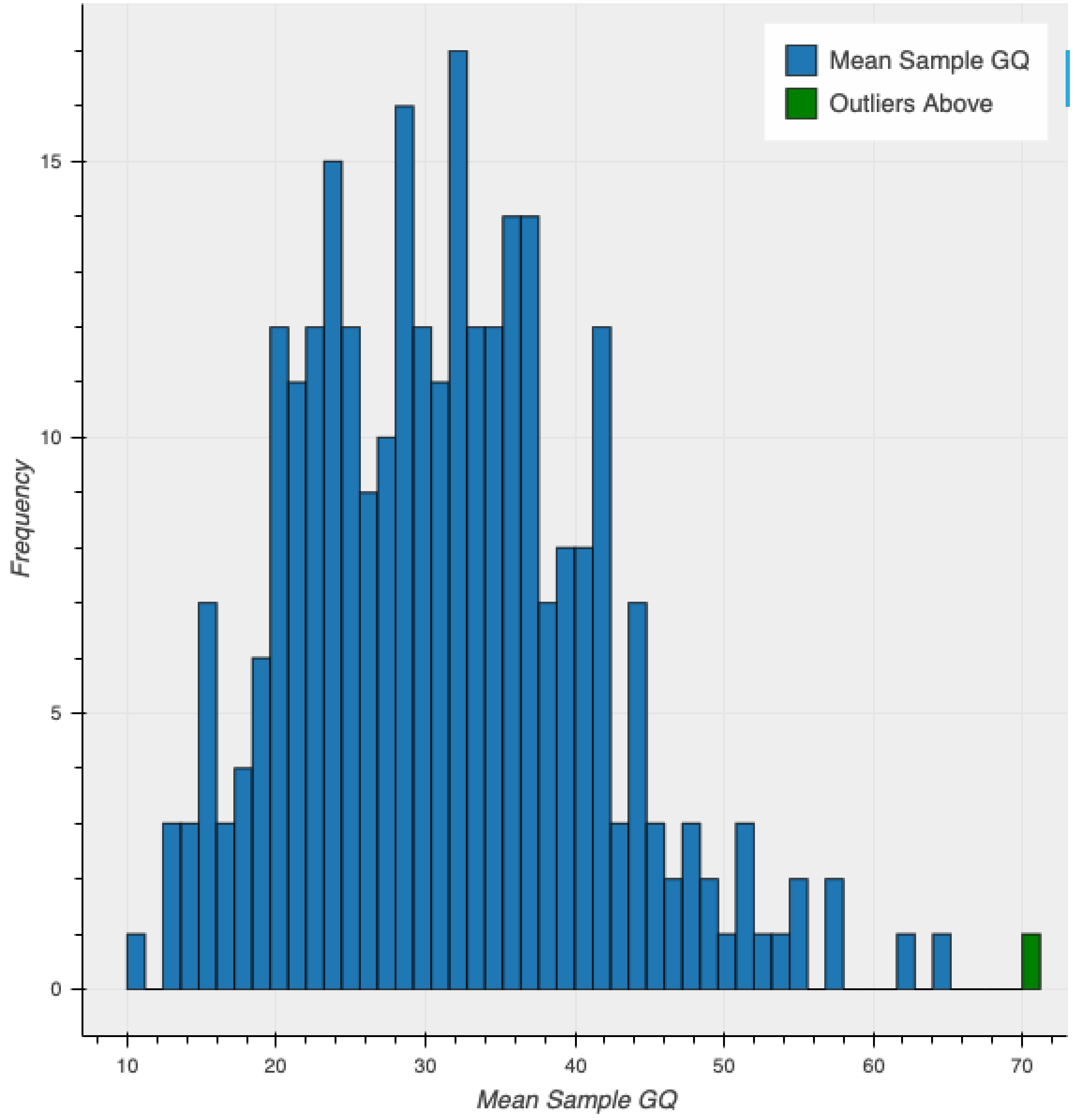
```
--------------------------------------------------------
Type:
        struct {
        s: str,
        pheno: struct {
            Population: str,
            SuperPopulation: str,
            isFemale: bool,
            PurpleHair: bool,
            CaffeineConsumption: int32
        },
        sample_qc: struct {
            dp_stats: struct {
                mean: float64,
                stdev: float64,
                min: float64,
                max: float64
            },
            gq_stats: struct {
                mean: float64,
                stdev: float64,
                min: float64,
                max: float64
            },
            call_rate: float64,
            n_called: int64,
            n_not_called: int64,
            n_filtered: int64,
            n_hom_ref: int64,
            n_het: int64,
            n_hom_var: int64,
            n_non_ref: int64,
            n_singleton: int64,
            n_snp: int64,
            n_insertion: int64,
            n_deletion: int64,
            n_transition: int64,
            n_transversion: int64,
            n_star: int64,
            r_ti_tv: float64,
            r_het_hom_var: float64,
            r_insertion_deletion: float64
        }
    }
--------------------------------------------------------
Source:
    <hail.matrixtable.MatrixTable object at 0x11ff70050>
Index:
    ['column']
--------------------------------------------------------
```

In [27]:
```
p = hl.plot.histogram(mt.sample_qc.call_rate, range=(.88,1), legend='Call Rate')
show(p)
```



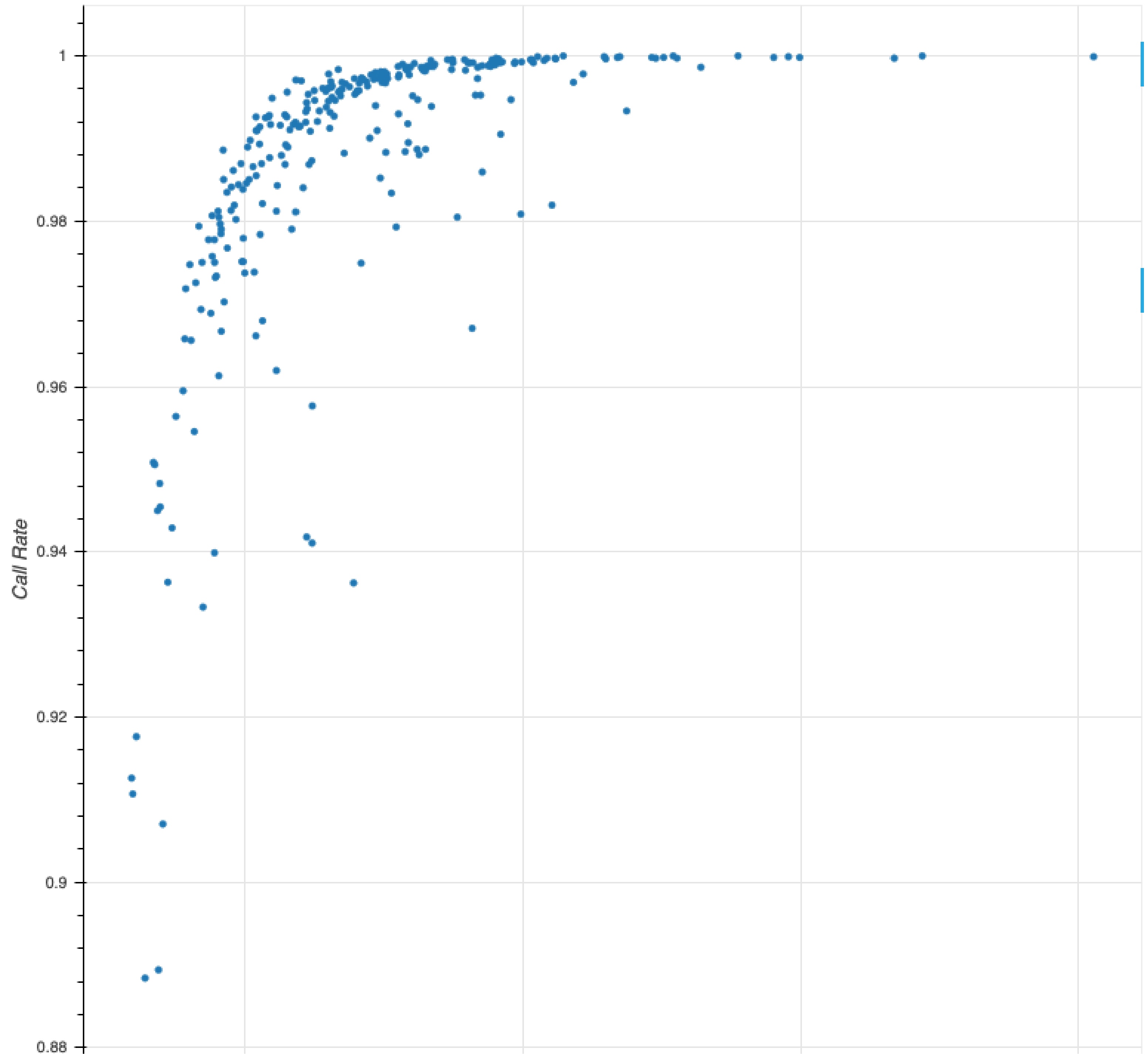
(http

```
In [28]: p = hl.plot.histogram(mt.sample_qc.gq_stats.mean, range=(10,70), legend='Mean Sample GQ')
         show(p)
```

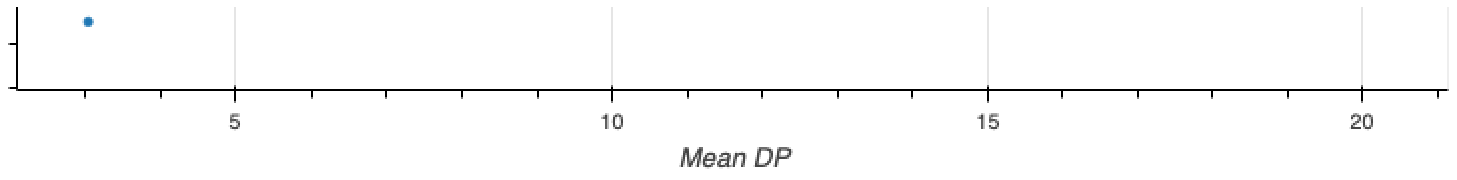


(http

In [29]:
```
# Often these metrics are correlated
p = hl.plot.scatter(mt.sample_qc.dp_stats.mean, mt.sample_qc.call_rate, xlabel='Mean DP', ylabel='Call Rate')
show(p)
```



(http



- Removing outliers from the dataset will generally improve association results. We can make arbitrary cutoffs and use them to filter.

In [30]:
```
mt = mt.filter_cols((mt.sample_qc.dp_stats.mean >= 4) & (mt.sample_qc.call_rate >= 0.97))
print('After filter, %d/284 samples remain.' % mt.count_cols())
```

```
After filter, 250/284 samples remain.
```

- enotype QC:
  - filter out genotypes where the reads aren't where they should be: if we find a genotype called homozygous reference with >10% alternate reads, a genotype called homozygous alternate with >10% reference reads, or a genotype called heterozygote without a ref / alt balance near 1:1, it is likely to be an error.
- In a low-depth dataset like 1KG, it is hard to detect bad genotypes using this metric, since a read ratio of 1 alt to 10 reference can easily be explained by binomial sampling. However, in a high-depth dataset, a read ratio of 10:100 is a sure cause for concern!

In [31]:
```
ab = mt.AD[1] / hl.sum(mt.AD)

filter_condition_ab = ((mt.GT.is_hom_ref() & (ab <= 0.1)) |
                        (mt.GT.is_het() & (ab >= 0.25) & (ab <= 0.75)) |
                        (mt.GT.is_hom_var() & (ab >= 0.9)))

fraction_filtered = mt.aggregate_entries(hl.agg.fraction(~filter_condition_ab))
print(f'Filtering {fraction_filtered * 100:.2f}% entries out of downstream analysis.')
mt = mt.filter_entries(filter_condition_ab)
```

```
Filtering 3.64% entries out of downstream analysis.
```

In [32]:
```
mt = hl.variant_qc(mt)
mt.row.describe()
```

```
--------------------------------------------------------
Type:
        struct {
        locus: locus<GRCh37>,
        alleles: array<str>,
        rsid: str,
        qual: float64,
        filters: set<str>,
        info: struct {
            AC: array<int32>,
            AF: array<float64>,
            AN: int32,
            BaseQRankSum: float64,
            ClippingRankSum: float64,
            DP: int32,
            DS: bool,
            FS: float64,
            HaplotypeScore: float64,
            InbreedingCoeff: float64,
            MLEAC: array<int32>,
            MLEAF: array<float64>,
            MQ: float64,
            MQ0: int32,
            MQRankSum: float64,
            QD: float64,
            ReadPosRankSum: float64,
            set: str
        },
        variant_qc: struct {
            dp_stats: struct {
                mean: float64,
                stdev: float64,
                min: float64,
                max: float64
            },
            gq_stats: struct {
                mean: float64,
                stdev: float64,
                min: float64,
                max: float64
            },
            AC: array<int32>,
            AF: array<float64>,
            AN: int32,
            homozygote_count: array<int32>,
            call_rate: float64,
            n_called: int64,
            n_not_called: int64,
            n_filtered: int64,
            n_het: int64,
            n_non_ref: int64,
            het_freq_hwe: float64,
            p_value_hwe: float64
        }
    }
--------------------------------------------------------
Source:
    <hail.matrixtable.MatrixTable object at 0x1200a1190>
Index:
    ['row']
--------------------------------------------------------
```

- These statistics look good: no need to filter this dataset. Most datasets require thoughtful quality control, though. The `filter_rows` method can help.

GWAS

- restrict to variants that are common (cutoff of 1%) and not so far from Hardy-Weinberg equilibrium as to suggest sequencing error

In [33]:
```
mt = mt.filter_rows(mt.variant_qc.AF[1] > 0.01)
mt = mt.filter_rows(mt.variant_qc.p_value_hwe > 1e-6)
print('Samples: %d  Variants: %d' % (mt.count_cols(), mt.count_rows()))
```

```
Samples: 250  Variants: 7849
```

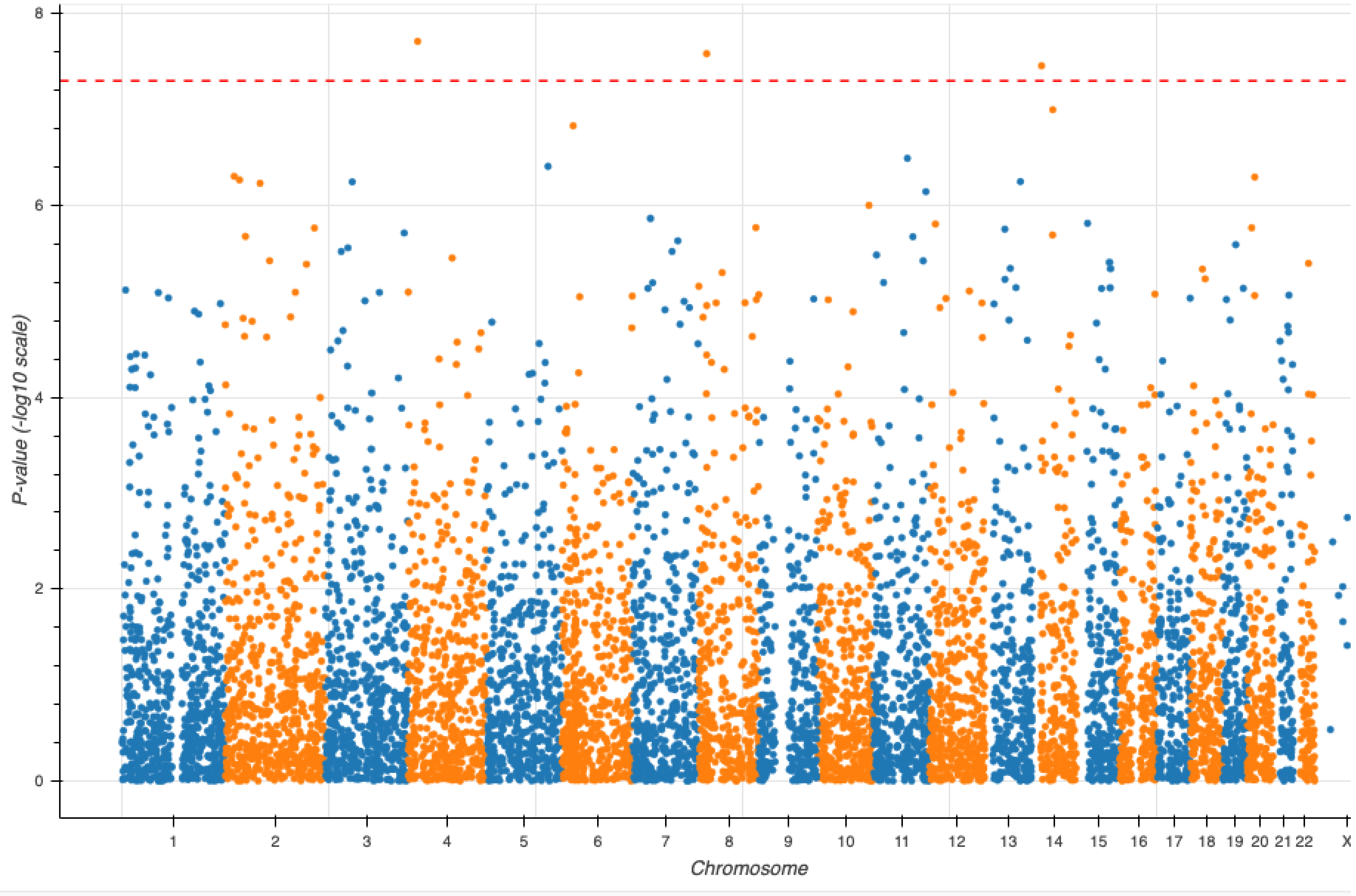
- These filters removed about 15% of sites (we started with a bit over 10,000). This is NOT representative of most sequencing datasets! We have already downsampled the full thousand genomes dataset to include more common variants than we'd expect by chance.
- In Hail, the association tests accept column fields for the sample phenotype and covariates. Since we've already got our phenotype of interest (caffeine consumption) in the dataset, we are good to go:

In [34]:
```
gwas = hl.linear_regression_rows(y=mt.pheno.CaffeineConsumption,
                                 x=mt.GT.n_alt_alleles(),
                                 covariates=[1.0])
gwas.row.describe()
```

```
2019-10-21 13:21:02 Hail: INFO: linear_regression_rows: running on 250 samples for 1 response variable y,
    with input variable x, and 1 additional covariate...

--------------------------------------------------------
Type:
        struct {
        locus: locus<GRCh37>,
        alleles: array<str>,
        n: int32,
        sum_x: float64,
        y_transpose_x: float64,
        beta: float64,
        standard_error: float64,
        t_stat: float64,
        p_value: float64
    }
--------------------------------------------------------
Source:
    <hail.table.Table object at 0x120056f50>
Index:
    ['row']
--------------------------------------------------------
```

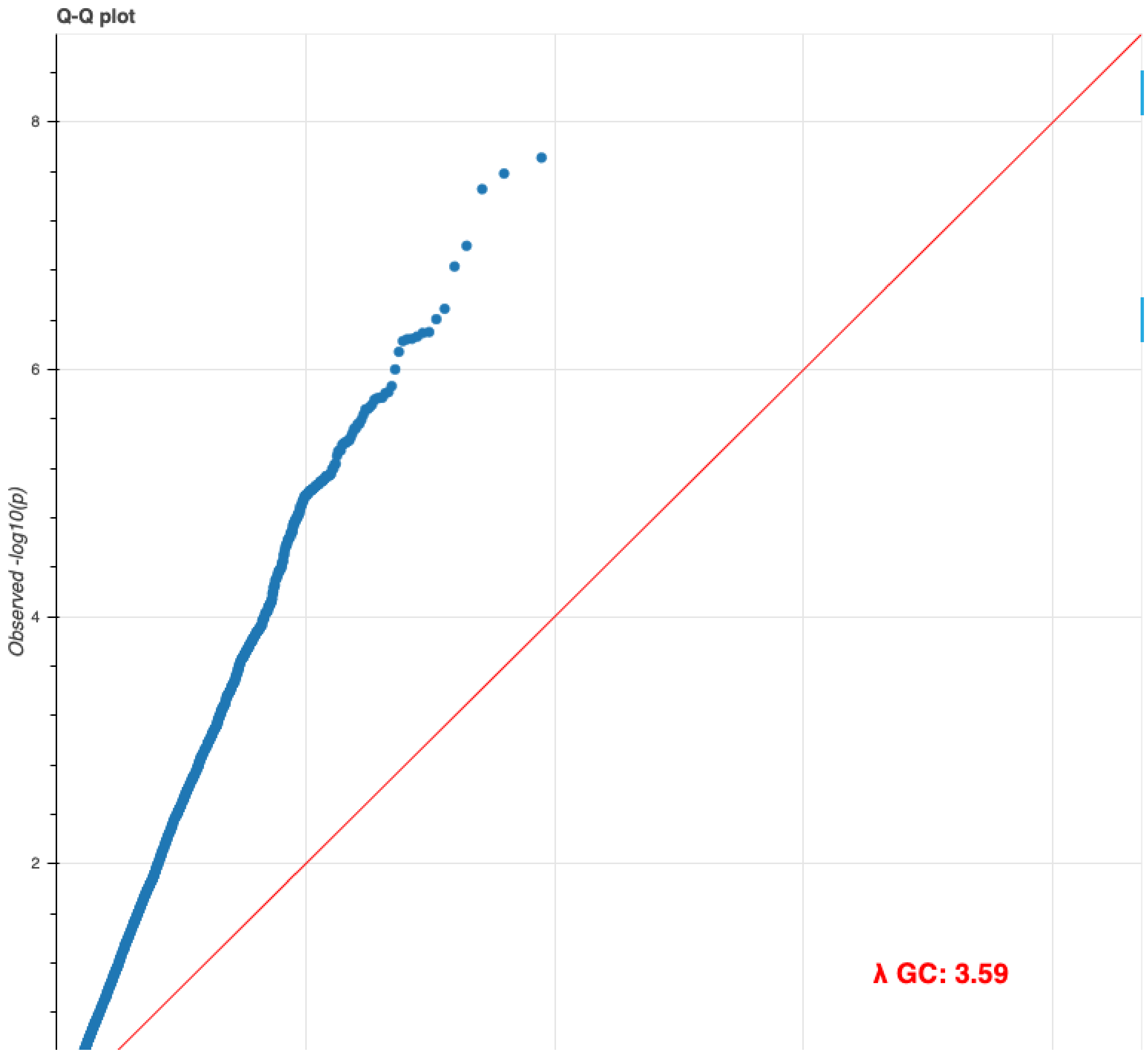
In [35]:
```
p = hl.plot.manhattan(gwas.p_value)
show(p)
```



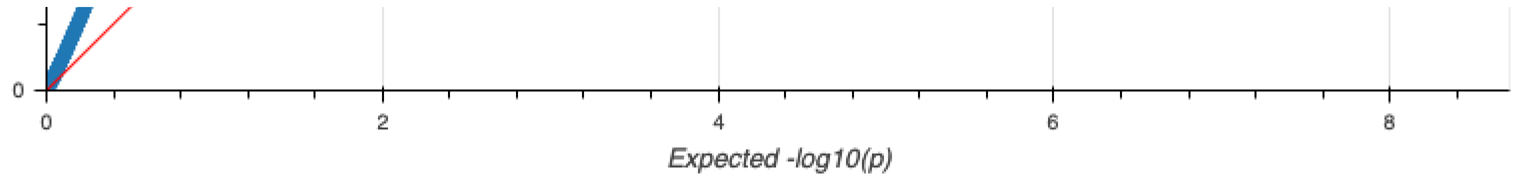
- check whether GWAS was well controlled using a Q-Q (quantile-quantile) plot.

In [36]:
```
p = hl.plot.qq(gwas.p_value)
show(p)
```

```
2019-10-21 13:21:12 Hail: INFO: Ordering unsorted dataset with network shuffle
```



(http



Dealing with the confounded

- The observed p-values drift away from the expectation immediately. Either every SNP in our dataset is causally linked to caffeine consumption (unlikely), or there's a confounder.
- We didn't tell you, but sample **ancestry was actually used to simulate this phenotype.** This leads to a stratified distribution of the phenotype. The solution is to include ancestry as a covariate in our regression.
- The `linear_regression_rows` function can also take column fields to use as covariates. We already annotated our samples with reported ancestry, but it is good to be skeptical of these labels due to human error. Genomes don't have that problem! **Instead of using reported ancestry, we will use genetic ancestry by including computed principal components in our model.**
- The `pca` function produces eigenvalues as a list and sample PCs as a Table, and can also produce variant loadings when asked. The `hwe_normalized_pca` function does the same, using HWE-normalized genotypes for the PCA.

In [37]:
```
eigenvalues, pcs, _ = hl.hwe_normalized_pca(mt.GT)
pprint(eigenvalues)
```

```
2019-10-21 13:21:21 Hail: INFO: hwe_normalized_pca: running PCA using 7841 variants.
2019-10-21 13:21:26 Hail: INFO: pca: running PCA with 10 components...

[18.023779471846865,
 9.98894555036327,
 3.5383122629171146,
 2.6577590783729916,
 1.5966032147658367,
 1.5416611649602372,
 1.5029872248781735,
 1.472081637853113,
 1.4678188487330768,
 1.447783520133495]
```

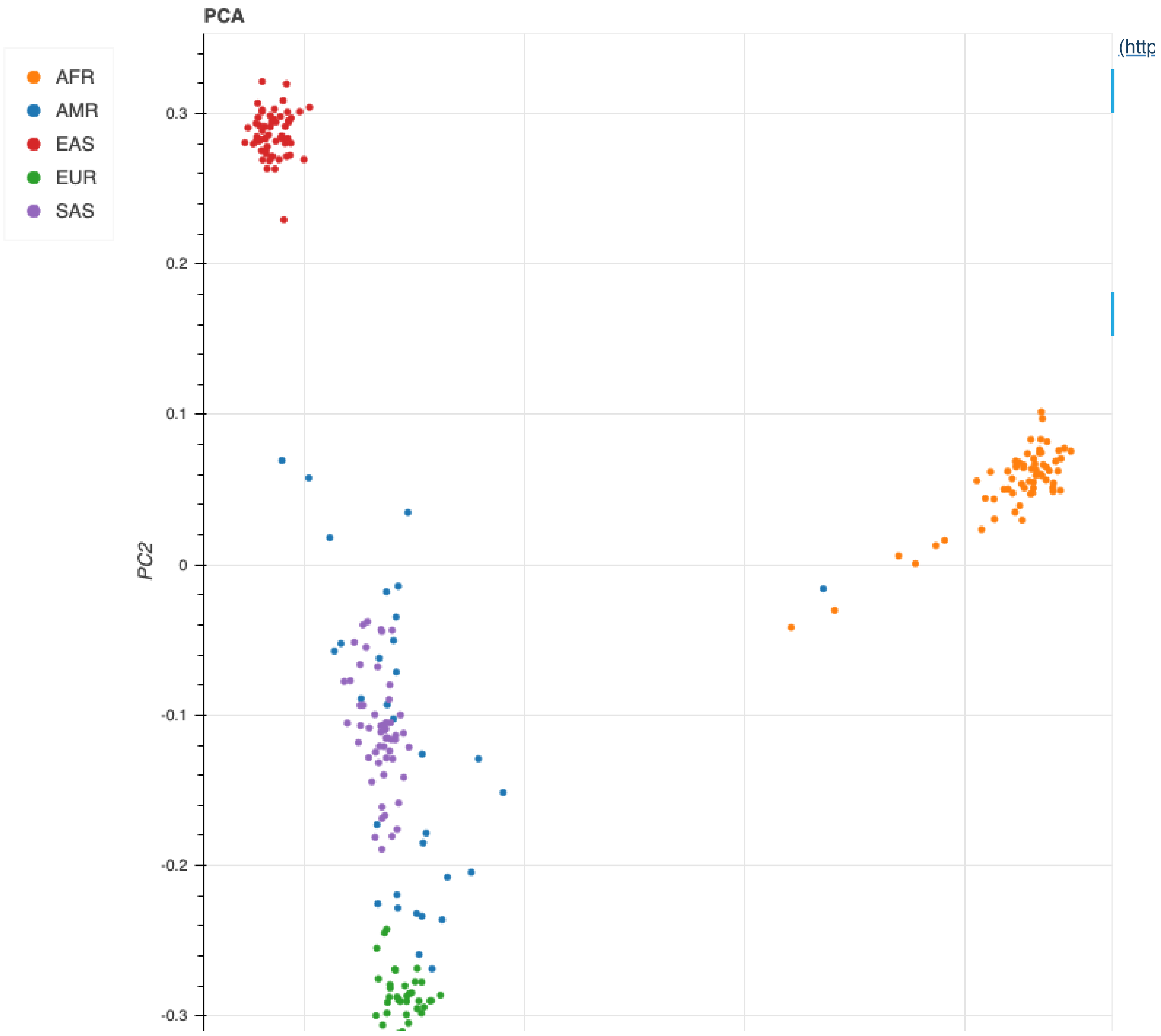
In [38]:
```
pcs.show(5, width=100)
```

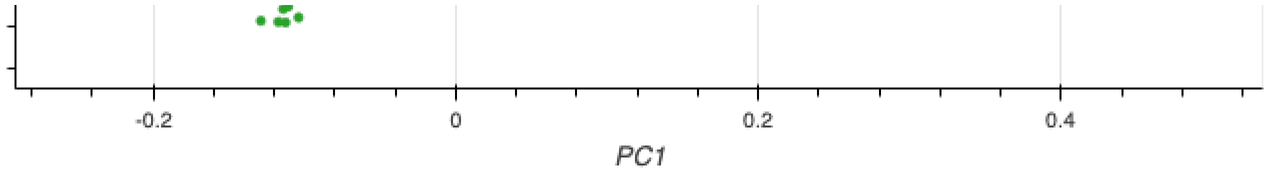
```
        s                                                                                                   scores
      str                                                                                           array<float64>
"HG00096"    [-1.22e-01,-2.81e-01,1.11e-01,-1.28e-01,6.81e-02,-3.72e-03,-2.66e-02,4.99e-03,-9.33e-02,-1.48...
"HG00099"   [-1.13e-01,-2.90e-01,1.08e-01,-7.04e-02,4.20e-02,3.33e-02,1.61e-02,-1.15e-03,3.29e-02,2.33e-02]
"HG00105"   [-1.08e-01,-2.80e-01,1.03e-01,-1.05e-01,9.40e-02,1.27e-02,3.14e-02,3.08e-02,1.06e-02,-1.93e-02]
"HG00118"    [-1.25e-01,-2.98e-01,7.21e-02,-1.07e-01,2.89e-02,8.09e-03,-4.70e-02,-3.32e-02,-2.59e-04,8.49e...
"HG00129"  [-1.07e-01,-2.87e-01,9.72e-02,-1.16e-01,1.38e-02,1.87e-02,-8.37e-02,-4.87e-02,3.73e-02,2.11e-02]

showing top 5 rows
```

- Now that we've got principal components per sample, we may as well plot them! Human history exerts a strong effect in genetic datasets. Even with a 50MB sequencing dataset, we can recover the major human populations.

In [39]:
```
mt = mt.annotate_cols(scores = pcs[mt.s].scores)
p = hl.plot.scatter(mt.scores[0],
                    mt.scores[1],
                    label=mt.pheno.SuperPopulation,
                    title='PCA', xlabel='PC1', ylabel='PC2')
show(p)
```


PCA
AFR
AMR
EAS
EUR
SAS
PC2
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
(http



- Now we can rerun the linear regression, controlling for sample sex and the first few principal components. We'll do this with input variable the number of alternate alleles as before, and again with input variable the genotype dosage derived from the PL field.

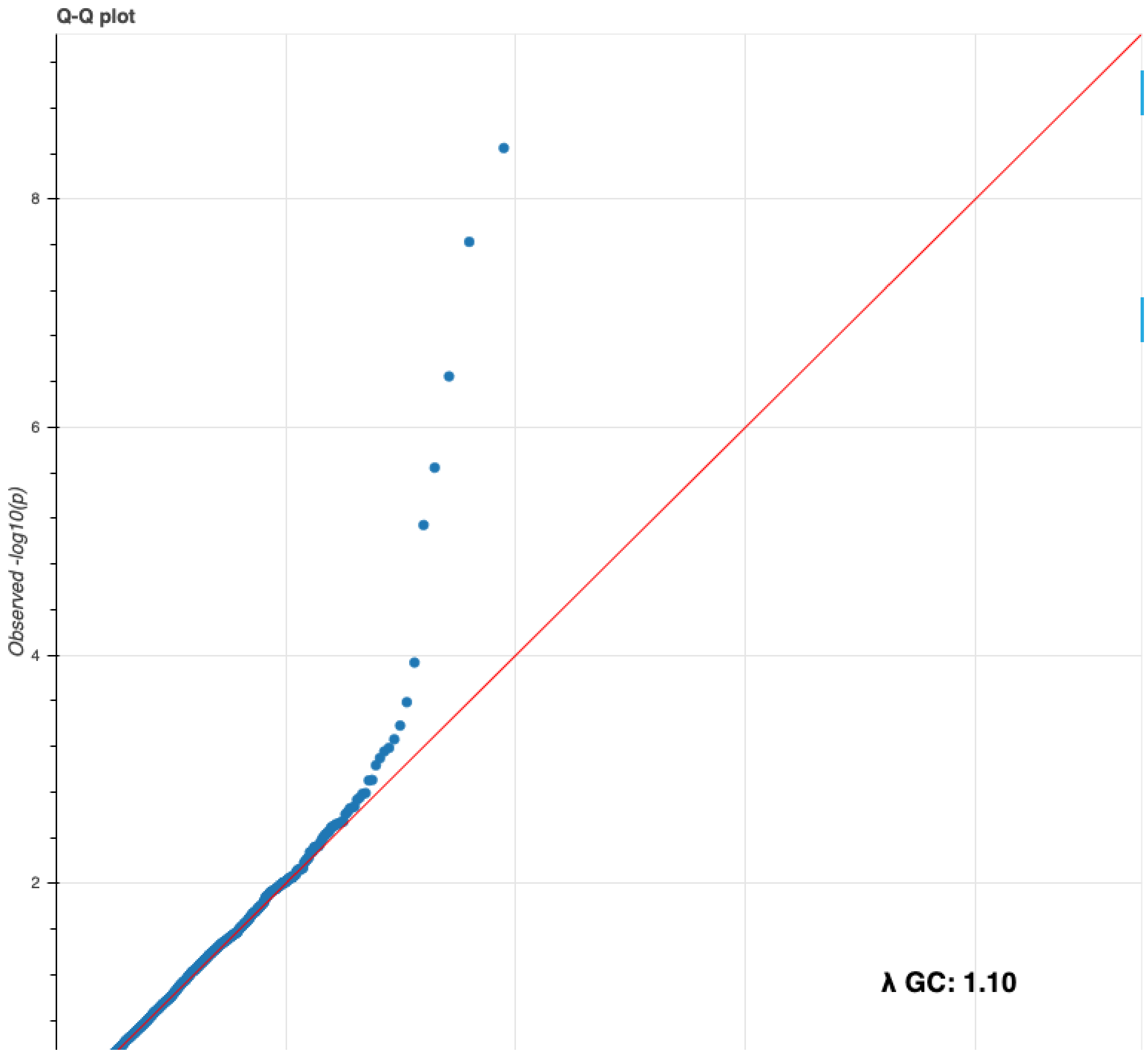
In [40]:
```
gwas = hl.linear_regression_rows(
    y=mt.pheno.CaffeineConsumption,
    x=mt.GT.n_alt_alleles(),
    covariates=[1.0, mt.pheno.isFemale, mt.scores[0], mt.scores[1], mt.scores[2]])
```

```
2019-10-21 13:21:38 Hail: INFO: linear_regression_rows: running on 250 samples for 1 response variable y,
    with input variable x, and 5 additional covariates...
```

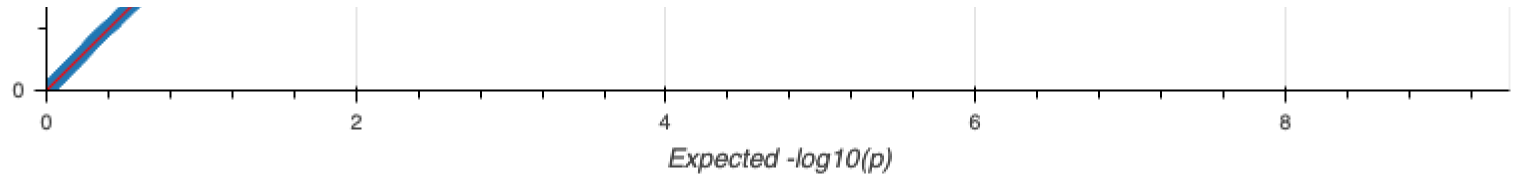
- We'll first make a Q-Q plot to assess inflation…

In [41]:
```
p = hl.plot.qq(gwas.p_value)
show(p)
```

```
2019-10-21 13:21:41 Hail: INFO: Ordering unsorted dataset with network shuffle
```

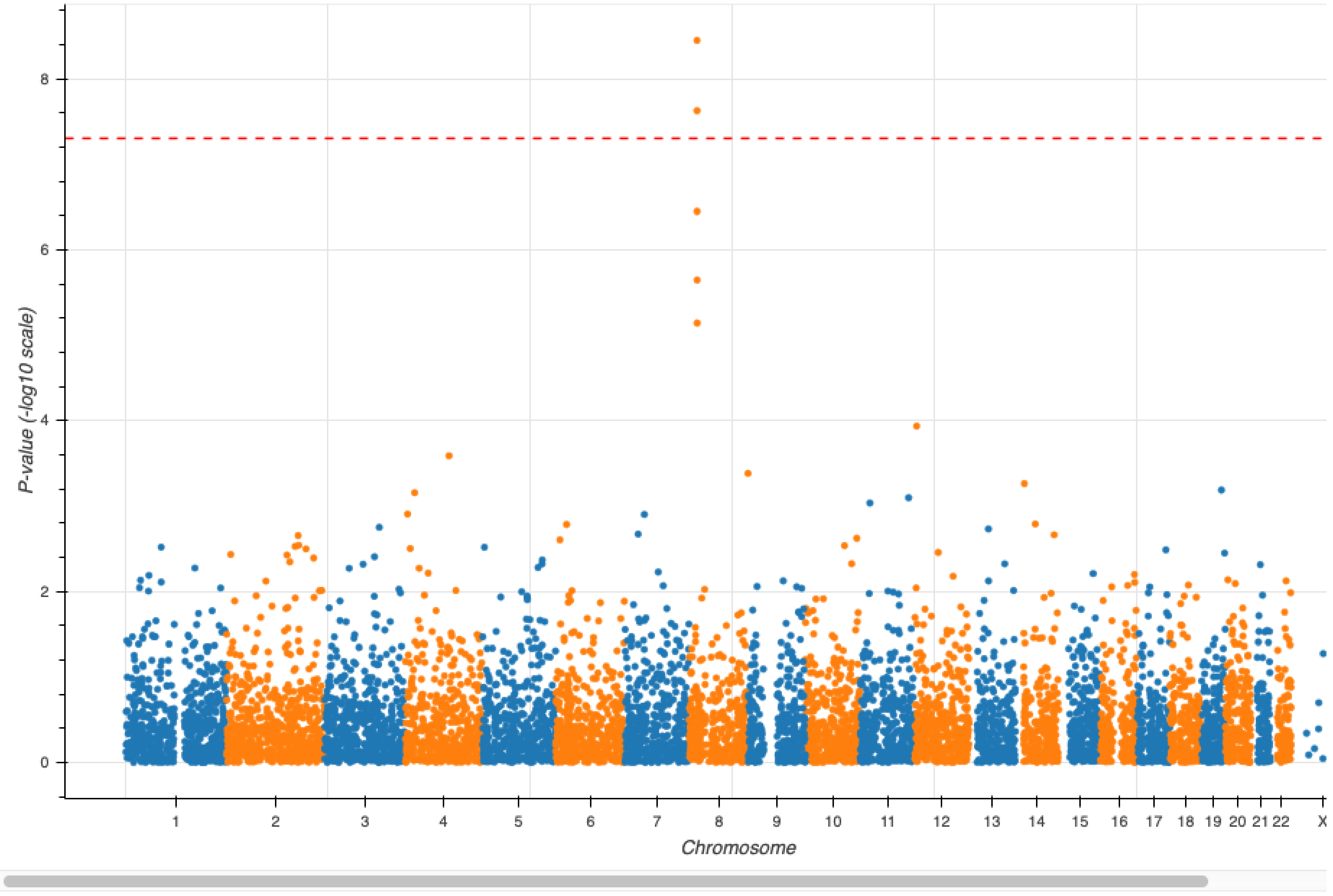


(http



- This shape is indicative of a well-controlled (but not especially well-powered) study.

In [42]:
```
p = hl.plot.manhattan(gwas.p_value)
show(p)
```



- We have found a caffeine consumption locus.

Rare variant analysis

- how one can use the expression language to group and count by any arbitrary properties in row and column fields.
- Hail also implements the sequence kernel association test (SKAT).

```
In [43]: entries = mt.entries()
         results = (entries.group_by(pop = entries.pheno.SuperPopulation, chromosome = entries.locus.contig)
               .aggregate(n_het = hl.agg.count_where(entries.GT.is_het())))
         results.show()
```

```
2019-10-21 13:21:48 Hail: WARN: entries(): Resulting entries table is sorted by '(row_key, col_key)'.
    To preserve row-major matrix table order, first unkey columns with 'key_cols_by()'
2019-10-21 13:21:53 Hail: INFO: Ordering unsorted dataset with network shuffle
```

pop	chromosome	n_het
str	str	int64
"AFR"	"1"	11276
"AFR"	"10"	7160
"AFR"	"11"	6875
"AFR"	"12"	7048
"AFR"	"13"	4678
"AFR"	"14"	4313
"AFR"	"15"	3904
"AFR"	"16"	4593
"AFR"	"17"	3718
"AFR"	"18"	4171

showing top 10 rows

- group by minor allele frequency bin and hair color, and calculate the mean GQ

In [44]:
```
entries = entries.annotate(maf_bin = hl.cond(entries.info.AF[0]<0.01, "< 1%",
                             hl.cond(entries.info.AF[0]<0.05, "1%-5%", ">5%")))

results2 = (entries.group_by(af_bin = entries.maf_bin, purple_hair = entries.pheno.PurpleHair)
      .aggregate(mean_gq = hl.agg.stats(entries.GQ).mean,
                 mean_dp = hl.agg.stats(entries.DP).mean))
results2.show()
```

```
2019-10-21 13:21:58 Hail: INFO: Ordering unsorted dataset with network shuffle
```

af_bin	purple_hair	mean_gq	mean_dp
str	bool	float64	float64
"1%-5%"	false	2.48e+01	7.43e+00
"1%-5%"	true	2.46e+01	7.47e+00
"< 1%"	false	2.35e+01	7.55e+00
"< 1%"	true	2.35e+01	7.53e+00
">5%"	false	3.70e+01	7.65e+00
">5%"	true	3.73e+01	7.70e+00

In [ ]: