In [2]:
```
import numpy as np
import scipy
import pandas
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set_style('white')
sns.set_style('ticks')
sns.set_context('notebook')
import h5py
import allel; print('scikit-allel', allel.__version__)
```

```
scikit-allel 1.3.8
```

VCF to HDF5

In [3]:
```
allel.vcf_to_hdf5('/users/mcevoysu/scratch/output/vcf_filtering/Qsuber/ra
```

Get data

In [4]:
```
callset_var_fn = '/users/mcevoysu/scratch/output/scikit-allel/Qsuber/raw_
callset_var = h5py.File(callset_var_fn, mode='r')
```

In [5]:
```
calldata_var = callset_var['calldata']
list(calldata_var)
```

Out[5]:
```
['AD', 'DP', 'GQ', 'GT', 'MIN_DP', 'PGT', 'PID', 'PL', 'PS', 'RGQ', 'S
B']
```

In [6]:
```
list(callset_var['variants'])
```

Out[6]:
```
['AC',
 'AF',
 'ALT',
 'AN',
 'BaseQRankSum',
 'CHROM',
 'DP',
 'END',
 'ExcessHet',
 'FILTER_LowQual',
 'FILTER_PASS',
 'FS',
 'ID',
 'InbreedingCoeff',
 'MLEAC',
 'MLEAF',
 'MQ',
 'MQRankSum',
 'POS',
 'QD',
 'QUAL',
 'RAW_MQandDP',
 'REF',
 'ReadPosRankSum',
 'SOR',
 'altlen',
 'is_snp',
 'numalt']
```

Make datasets

In [7]:
```
variants = allel.VariantChunkedTable(callset_var['variants'])
variants
```

Out[7]:

```
<VariantChunkedTable shape=(262868,) dtype=[('AC', '<i4', (3,)), ('AF', '<f4', (3,)), ('ALT', 'O', (3,)), ('AN', '<i4'), ('BaseQRankSum', '<f4'), ('CHROM', 'O'), ('DP', '<i4'), ('END', '<i4'), ('ExcessHet', '<f4'), ('FILTER_LowQual', '?'), ('FILTER_PASS', '?'), ('FS', '<f4'), ('ID', 'O'), ('InbreedingCoeff', '<f4'), ('MLEAC', '<i4', (3,)), ('MLEAF', '<f4', (3,)), ('MQ', '<f4'), ('MQRankSum', '<f4'), ('POS', '<i4'), ('QD', '<f4'), ('QUAL', '<f4'), ('RAW_MQandDP', '<i4', (2,)), ('REF', 'O'), ('ReadPosRankSum', '<f4'), ('SOR', '<f4'), ('altlen', '<i4', (3,)), ('is_snp', '?'), ('numalt', '<i4')] nbytes=44.9M cbytes=9.7M cratio=4.6 values=h5py._hl.group.Group>
```

	AC	AF	ALT	AN	BaseQRankSum	CHROM	DP	END	Ex
0	[1 -1 -1]	[0.001276 nan nan]	[b'T' b'' b'']	780	-0.842	b'chr01'	5250	-1	
1	[2 -1 -1]	[0.002551 nan nan]	[b'C' b'' b'']	780	-0.842	b'chr01'	5199	-1	(
2	[6 -1 -1]	[0.007653 nan nan]	[b'*' b'' b'']	780	-2.539	b'chr01'	4987	-1	(
...									
262865	[2 -1 -1]	[0.002551 nan nan]	[b'A' b'' b'']	780	nan	b'unanchored'	4	-1	
262866	[2 -1 -1]	[0.002551 nan nan]	[b'G' b'' b'']	780	nan	b'unanchored'	4	-1	
262867	[2 -1 -1]	[0.002551 nan nan]	[b'A' b'' b'']	780	nan	b'unanchored'	4	-1	

In [8]:

```python
variants_np = variants[:]
rawsnps = variants_np.query('(is_snp == True)')
rawsnps
```

Out[8]:

```
<VariantTable shape=(174360,) dtype=(numpy.record, [('AC', '<i4', (3,)), ('AF', '<f4', (3,)), ('ALT', 'O', (3,)), ('AN', '<i4'), ('BaseQRankSum', '<f4'), ('CHROM', 'O'), ('DP', '<i4'), ('END', '<i4'), ('ExcessHet', '<f4'), ('FILTER_LowQual', '?'), ('FILTER_PASS', '?'), ('FS', '<f4'), ('ID', 'O'), ('InbreedingCoeff', '<f4'), ('MLEAC', '<i4', (3,)), ('MLEAF', '<f4', (3,)), ('MQ', '<f4'), ('MQRankSum', '<f4'), ('POS', '<i4'), ('QD', '<f4'), ('QUAL', '<f4'), ('RAW_MQandDP', '<i4', (2,)), ('REF', 'O'), ('ReadPosRankSum', '<f4'), ('SOR', '<f4'), ('altlen', '<i4', (3,)), ('is_snp', '?'), ('numalt', '<i4')])>
```

	AC	AF	ALT	AN	BaseQRankSum	CHROM	DP	END	Exc
0	[1 -1 -1]	[0.001276 nan nan]	[b'T' b'' b'']	780	-0.842	b'chr01'	5250	-1	
1	[2 -1 -1]	[0.002551 nan nan]	[b'C' b'' b'']	780	-0.842	b'chr01'	5199	-1	0
2	[1 -1 -1]	[0.001276 nan nan]	[b'G' b'' b'']	780	-0.116	b'chr01'	4431	-1	
...									
174357	[2 -1 -1]	[0.002551 nan nan]	[b'A' b'' b'']	780	nan	b'unanchored'	4	-1	
174358	[2 -1 -1]	[0.002551 nan nan]	[b'G' b'' b'']	780	nan	b'unanchored'	4	-1	
174359	[2 -1 -1]	[0.002551 nan nan]	[b'A' b'' b'']	780	nan	b'unanchored'	4	-1	

In [9]:

```python
notsnp = variants_np.query('(is_snp != True)')
notsnp
```

Out[9]: <VariantTable shape=(88508,) dtype=(numpy.record, [('AC', '<i4', (3,)), ('AF', '<f4', (3,)), ('ALT', 'O', (3,)), ('AN', '<i4'), ('BaseQRankSum', '<f4'), ('CHROM', 'O'), ('DP', '<i4'), ('END', '<i4'), ('ExcessHet', '<f4'), ('FILTER_LowQual', '?'), ('FILTER_PASS', '?'), ('FS', '<f4'), ('ID', 'O'), ('InbreedingCoeff', '<f4'), ('MLEAC', '<i4', (3,)), ('MLEAF', '<f4', (3,)), ('MQ', '<f4'), ('MQRankSum', '<f4'), ('POS', '<i4'), ('QD', '<f4'), ('QUAL', '<f4'), ('RAW_MQandDP', '<i4', (2,)), ('REF', 'O'), ('ReadPosRankSum', '<f4'), ('SOR', '<f4'), ('altlen', '<i4', (3,)), ('is_snp', '?'), ('numalt', '<i4')])>

	AC	AF	ALT	AN	BaseQRankSum	CHROM	DP	END	E
0	[6 -1 -1]	[0.007653 nan nan]	[b'*' b'' b'']	780	-2.539	b'chr01'	4987	-1	
1	[6 -1 -1]	[0.007653 nan nan]	[b'*' b'' b'']	780	-0.967	b'chr01'	4963	-1	
2	[680 1 6]	[0.872 0.001276 0.007653]	[b'T' b'G' b'*']	780	0.349	b'chr01'	4793	-1	
...									
88505	[2 -1 -1]	[0.002558 nan nan]	[b'*' b'' b'']	778	nan	b'unanchored'	253	-1	
88506	[127 2 -1]	[0.162 0.002558 nan]	[b'A' b'*' b'']	778	0.0	b'unanchored'	254	-1	
88507	[170 2 -1]	[0.218 0.002564 nan]	[b'T' b'*' b'']	776	nan	b'unanchored'	253	-1	

Plot function

In [10]:
```python
def plot_hist(f, dsubset='', bins=30, ):
    if dsubset == 'var':
        x = variants[f][:]
        l = 'Variant'
    elif dsubset == 'snp':
        x = rawsnps[f][:]
        l = 'Raw SNP'
    elif dsubset == 'notsnp':
        x = notsnp[f][:]
        l = 'Raw Not SNP'
    elif dsubset == 'biallelic':
        x = biallelic_np[f][:]
        l = 'Biallelic SNP'
    elif dsubset == 'varsel':
        x = var_selection[f][:]
        l = 'Filtered Variants'
    elif dsubset == 'snpsel':
        x = snp_selection[f][:]
        l = 'Filtered SNP'
    else:
        x = bi_selection[f][:]
        l = 'Biallelic SNP'
    fig, ax = plt.subplots(figsize=(10, 5))
    sns.despine(ax=ax, offset=10)
    ax.hist(x, bins=bins)
    ax.set_xlabel(f)
    ax.set_ylabel('No. variants')
    ax.set_title('%s %s distribution' % (l, f))
```

Find Biallelic SNPS

In [11]:
```
numalt = rawsnps['numalt']
np.max(numalt)
```

Out[11]:
```
3
```

In [12]:
```
count_numalt = np.bincount(numalt)
count_numalt
```

Out[12]:
```
array([     0, 169431,   4794,    135])
```

In [13]:
```
n_multiallelic = np.sum(count_numalt[2:])
n_multiallelic
```

Out[13]:
```
4929
```

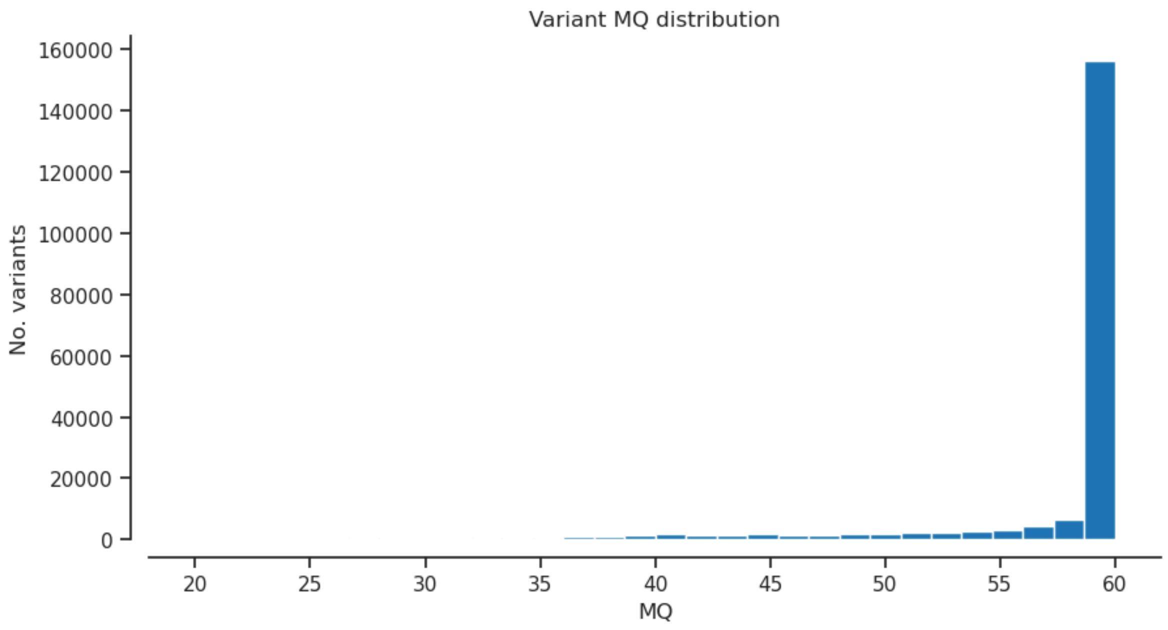
In [14]:
```
filter_expression = '(numalt == 1)'
biallelic_np = rawsnps.query(filter_expression)[:]
biallelic_np
```

Out[14]:
```
<VariantTable shape=(169431,) dtype=(numpy.record, [('AC', '<i4', (3,)), ('AF', '<f4', (3,)), ('ALT', 'O', (3,)), ('AN', '<i4'), ('BaseQRankSum', '<f4'), ('CHROM', 'O'), ('DP', '<i4'), ('END', '<i4'), ('ExcessHet', '<f4'), ('FILTER_LowQual', '?'), ('FILTER_PASS', '?'), ('FS', '<f4'), ('ID', 'O'), ('InbreedingCoeff', '<f4'), ('MLEAC', '<i4', (3,)), ('MLEAF', '<f4', (3,)), ('MQ', '<f4'), ('MQRankSum', '<f4'), ('POS', '<i4'), ('QD', '<f4'), ('QUAL', '<f4'), ('RAW_MQandDP', '<i4', (2,)), ('REF', 'O'), ('ReadPosRankSum', '<f4'), ('SOR', '<f4'), ('altlen', '<i4', (3,)), ('is_snp', '?'), ('numalt', '<i4')])>
```

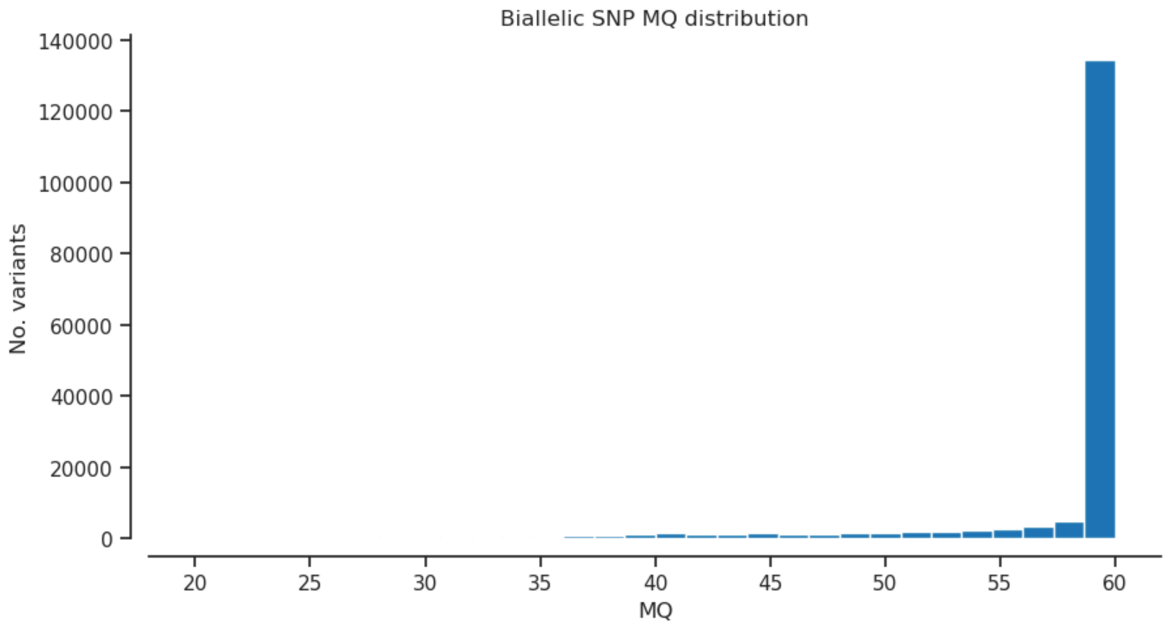
	AC	AF	ALT	AN	BaseQRankSum	CHROM	DP	END	Ex
0	[1 -1 -1]	[0.001276 nan nan]	[b'T' b'' b'']	780	-0.842	b'chr01'	5250	-1	
1	[2 -1 -1]	[0.002551 nan nan]	[b'C' b'' b'']	780	-0.842	b'chr01'	5199	-1	0
2	[1 -1 -1]	[0.001276 nan nan]	[b'G' b'' b'']	780	-0.116	b'chr01'	4431	-1	
...									
169428	[2 -1 -1]	[0.002551 nan nan]	[b'A' b'' b'']	780	nan	b'unanchored'	4	-1	
169429	[2 -1 -1]	[0.002551 nan nan]	[b'G' b'' b'']	780	nan	b'unanchored'	4	-1	
169430	[2 -1 -1]	[0.002551 nan nan]	[b'A' b'' b'']	780	nan	b'unanchored'	4	-1	

MQ - RMS mapping quality

In [15]:
```
plot_hist('MQ','var') # RMS mapping quality
```

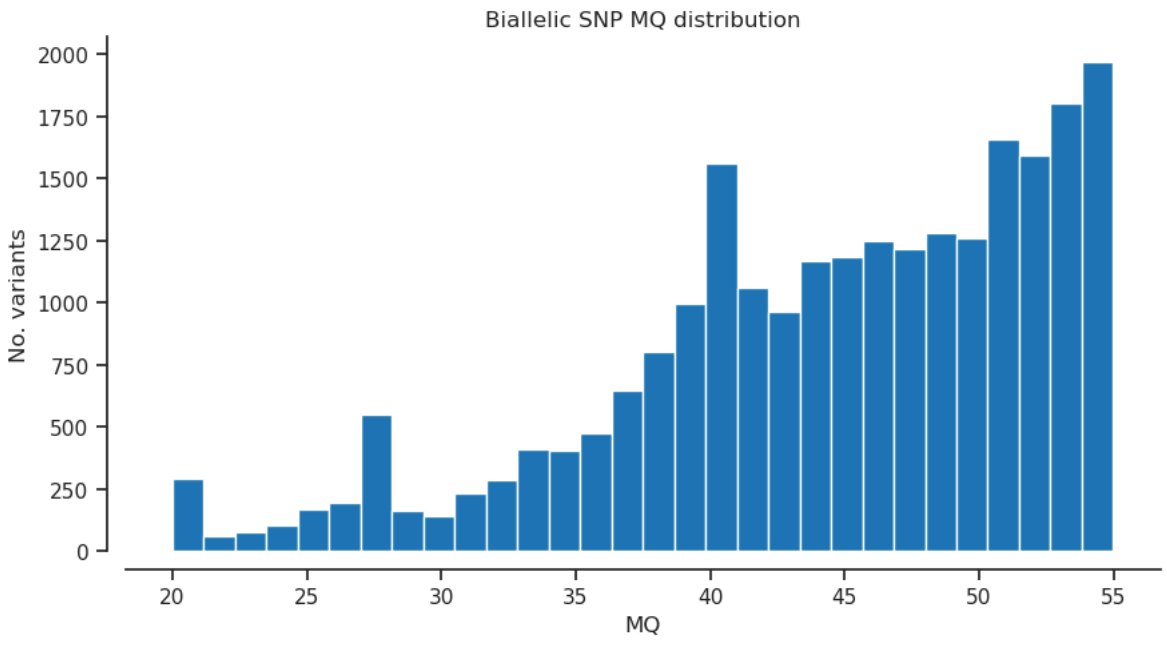


```
In [16]: plot_hist('MQ','biallelic') # RMS mapping quality
```



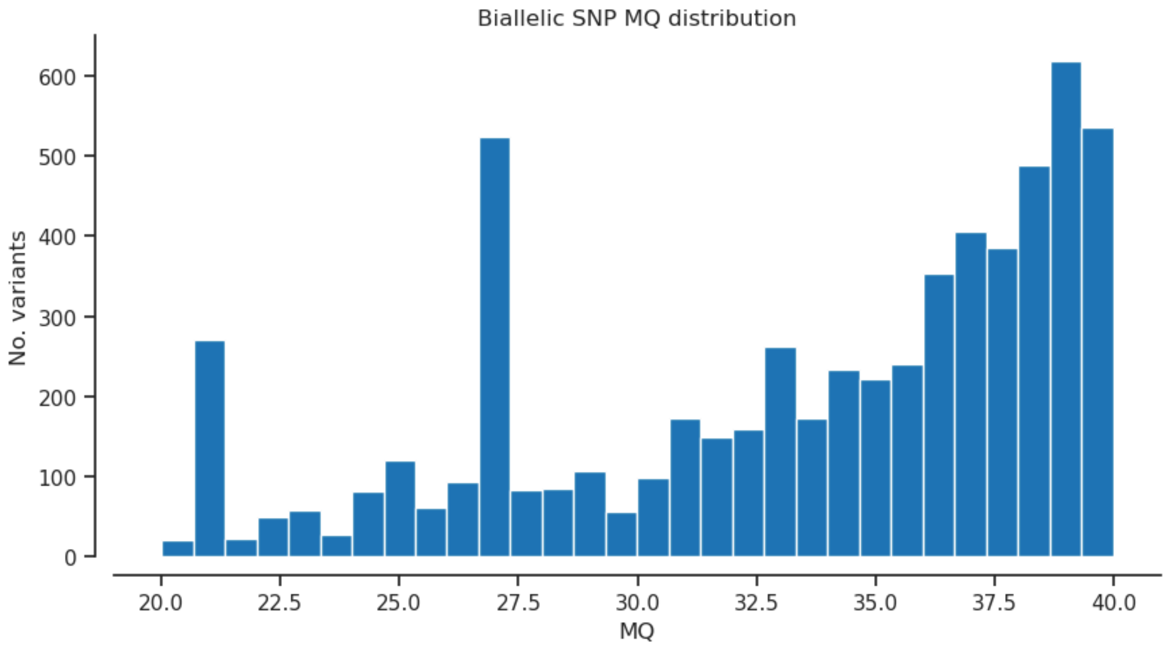
```
In [17]: filter_expression = '(MQ < 55)'
         bi_selection = biallelic_np.query(filter_expression)[:]
         #np.count_nonzero(var_selection)
```

```
In [18]: plot_hist('MQ')
```

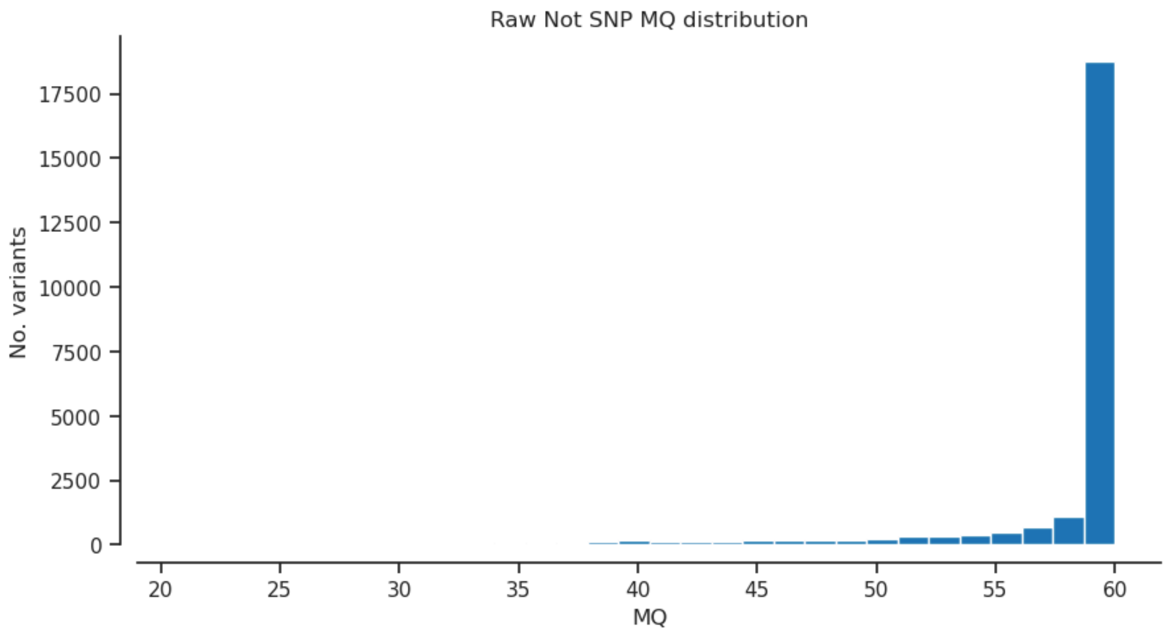


```
In [19]: filter_expression = '(MQ < 40)'
         bi_selection = biallelic_np.query(filter_expression)[:]
```

```
In [20]: plot_hist('MQ')
```

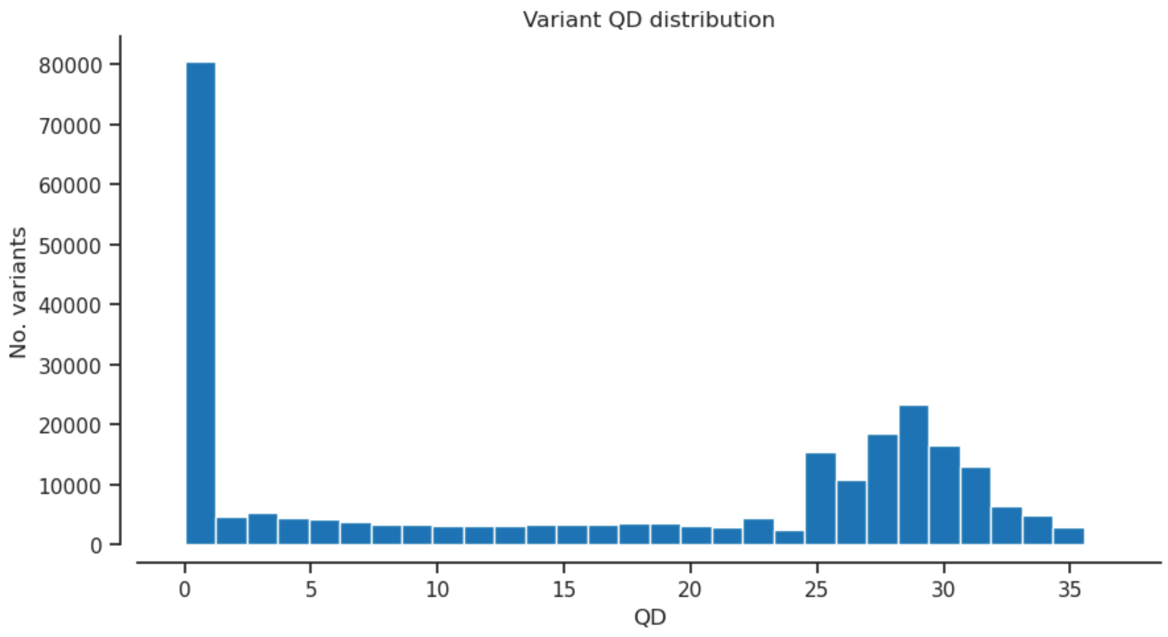


```
In [21]: plot_hist('MQ','notsnp')
```

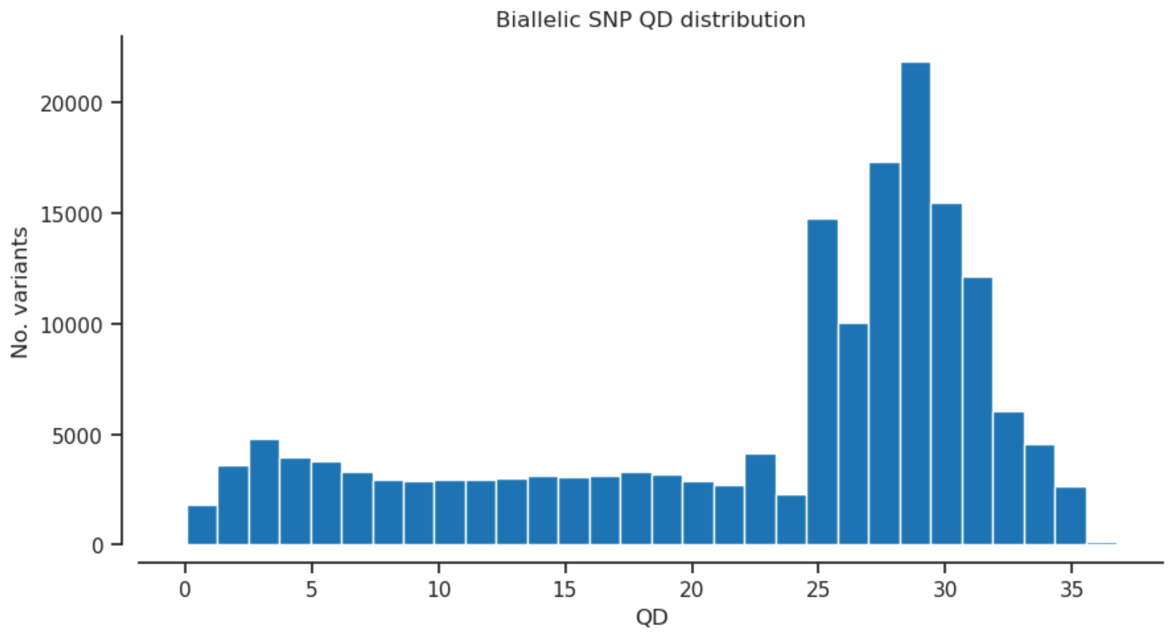


QD - Variant Confidence/Quality by Depth

```
In [22]: plot_hist('QD','var') # Variant Confidence/Quality by Depth
```

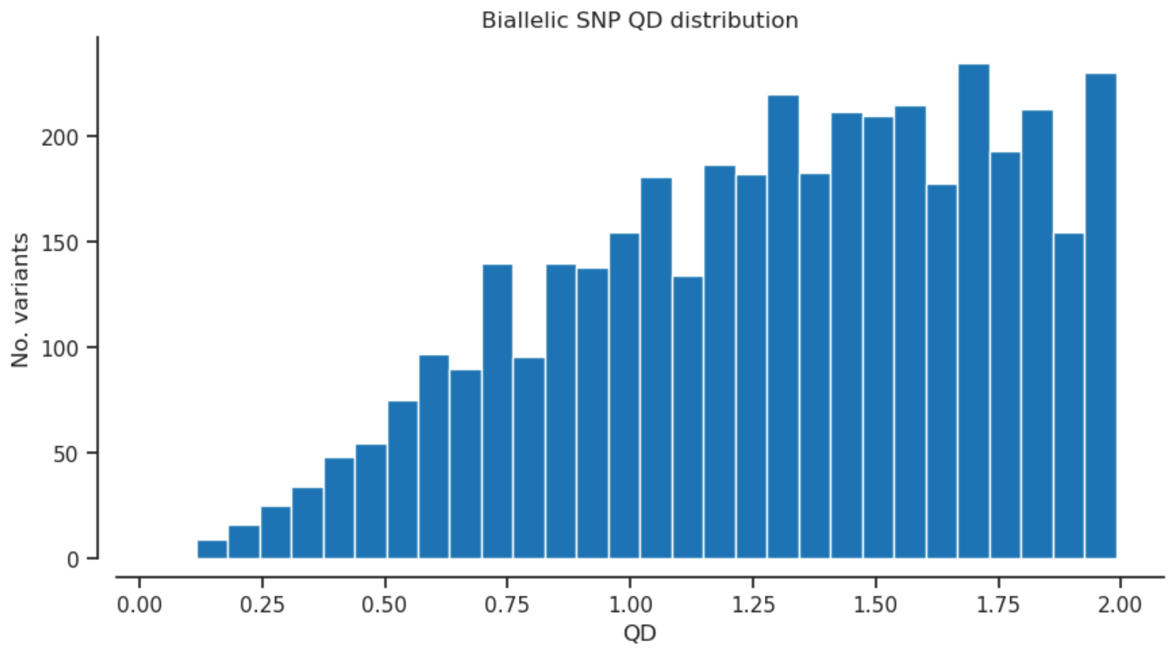


```
In [23]: plot_hist('QD','biallelic') # Variant Confidence/Quality by Depth
```

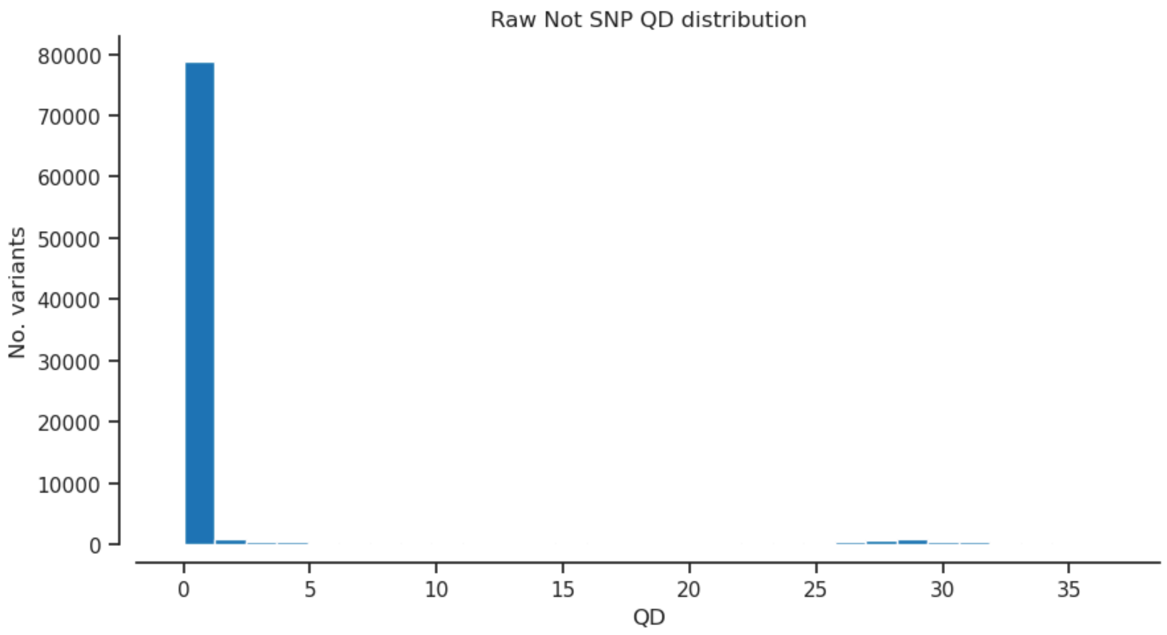


```
In [24]: filter_expression = '(QD < 2)'
         bi_selection = biallelic_np.query(filter_expression)[:]
```

```
In [25]: plot_hist('QD')
```

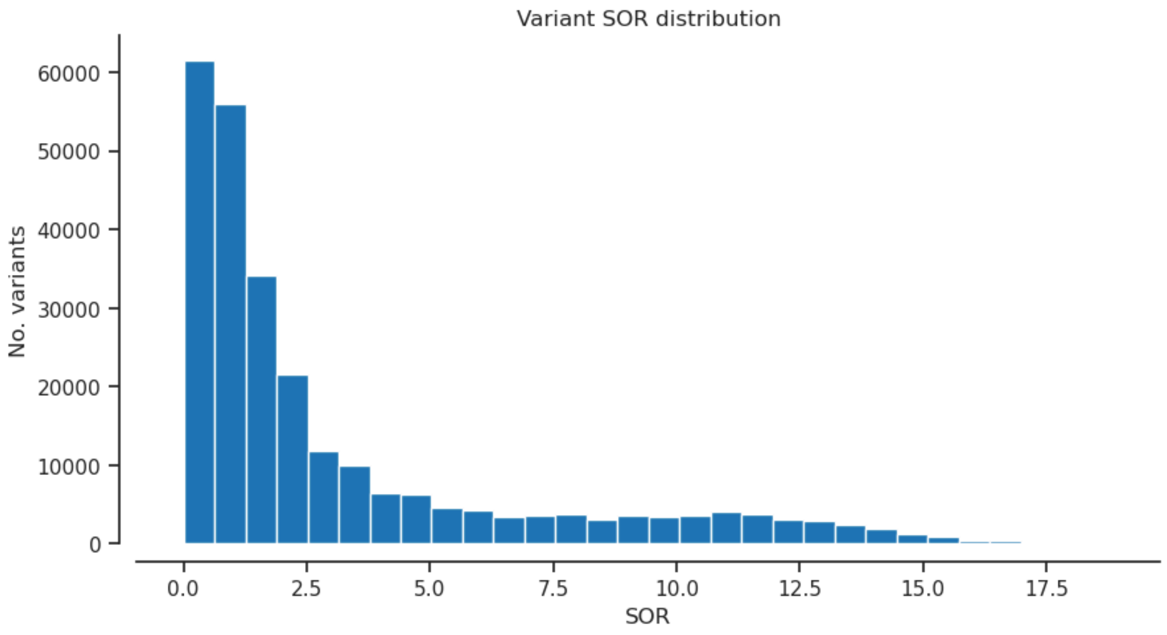


```
In [26]: plot_hist('QD','notsnp') # Variant Confidence/Quality by Depth
```

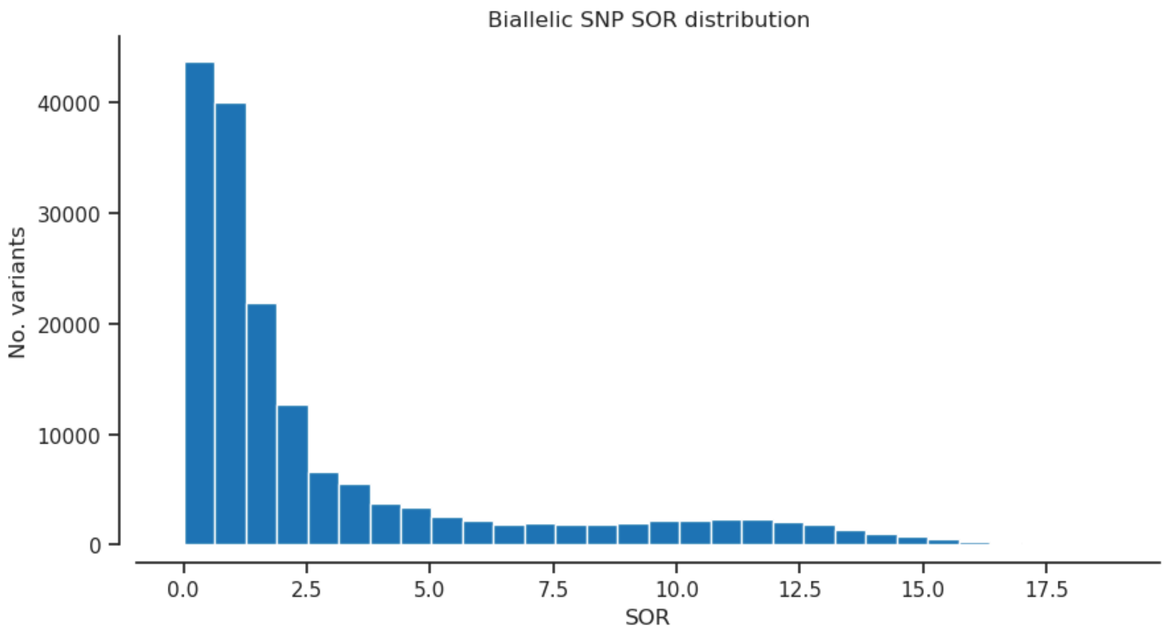


SOR - Symmetric Odds Ratio of 2x2 contingency table to detect strand bias

In [27]:
```
plot_hist('SOR', 'var') # Symmetric Odds Ratio of 2x2 contingency table t
```

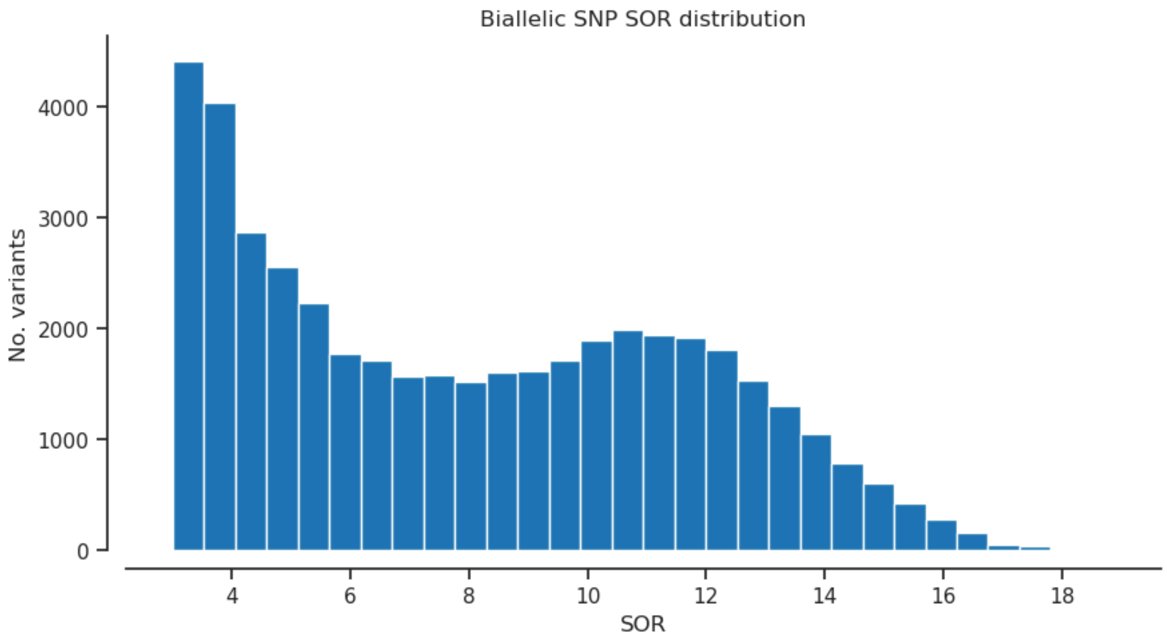


In [28]:
```
plot_hist('SOR','biallelic') # Symmetric Odds Ratio of 2x2 contingency ta
```

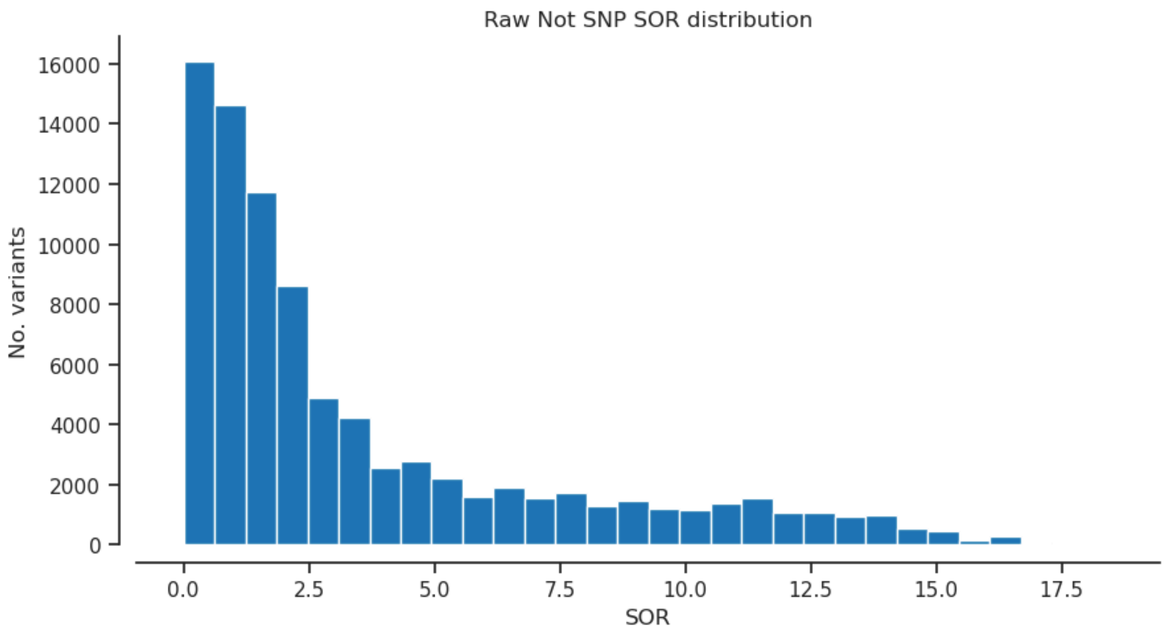


```
In [29]: filter_expression = '(SOR > 3)'
         bi_selection = biallelic_np.query(filter_expression)[:]
```

```
In [30]: plot_hist('SOR') # Symmetric Odds Ratio of 2x2 contingency table to detec
```

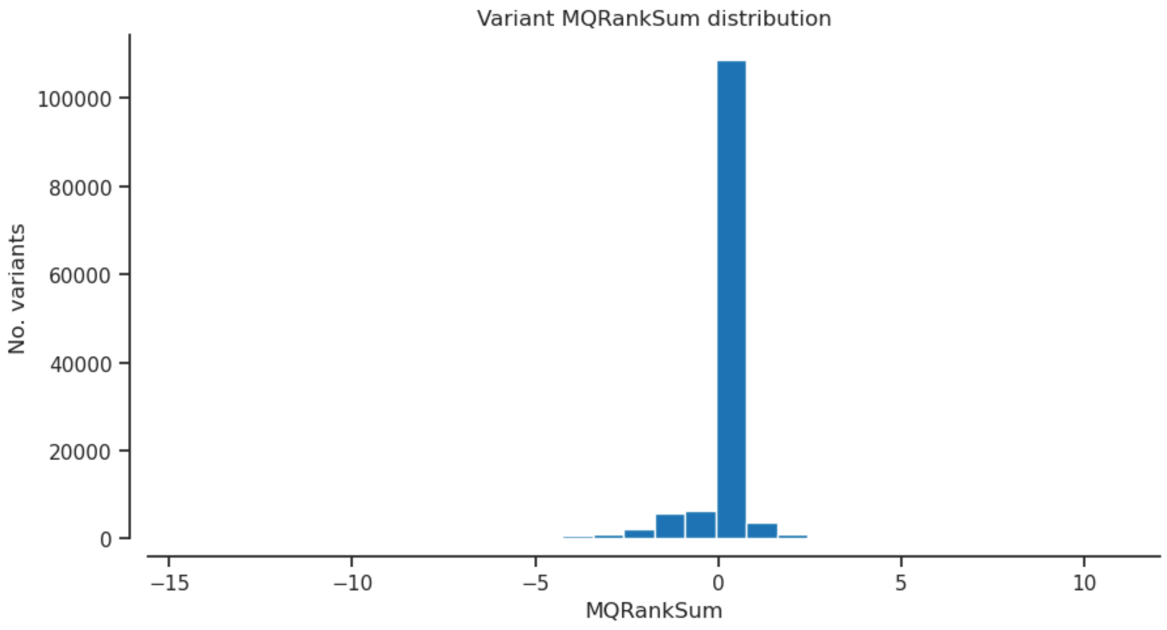


```
In [31]: plot_hist('SOR','notsnp') # Symmetric Odds Ratio of 2x2 contingency table
```

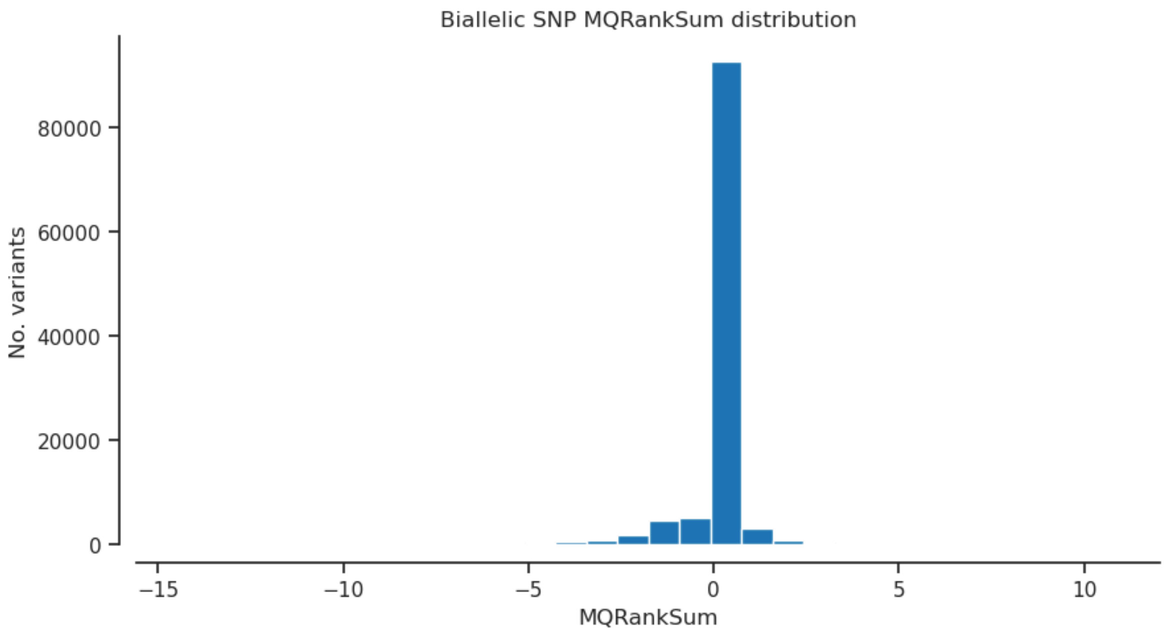


MQRankSum - Z-score From Wilcoxon rank sum test of Alt vs. Ref read mapping qualities

In [32]:
```
plot_hist('MQRankSum','var') # Z-score From Wilcoxon rank sum test of Alt
```

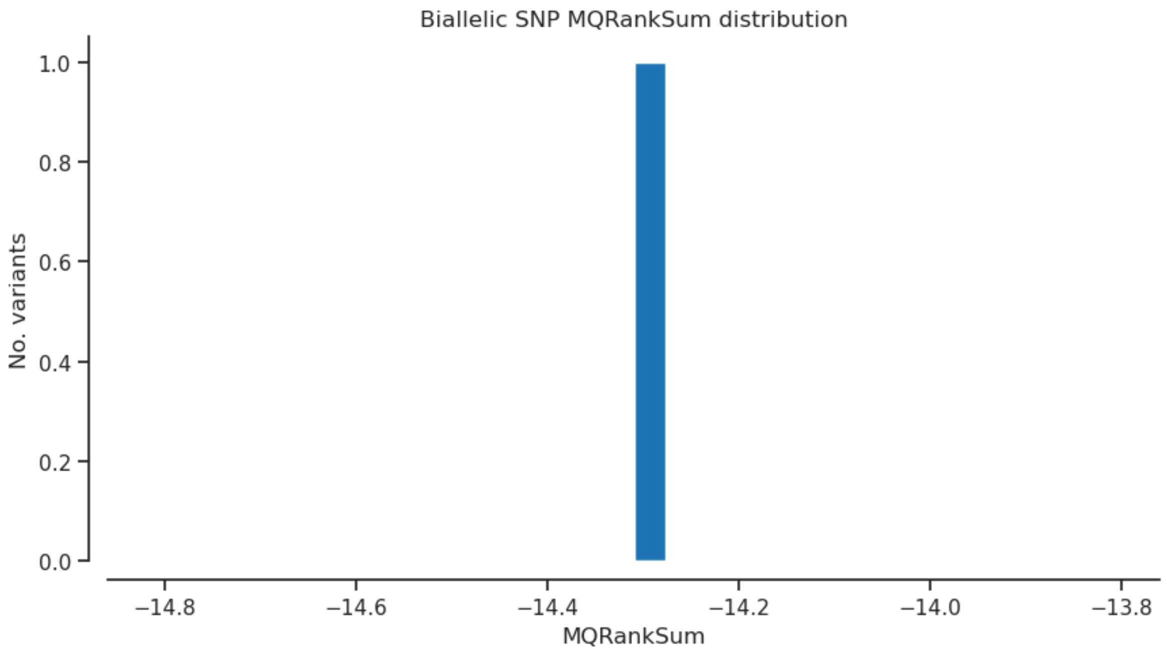


In [33]:
```
plot_hist('MQRankSum','biallelic') # Z-score From Wilcoxon rank sum test
```

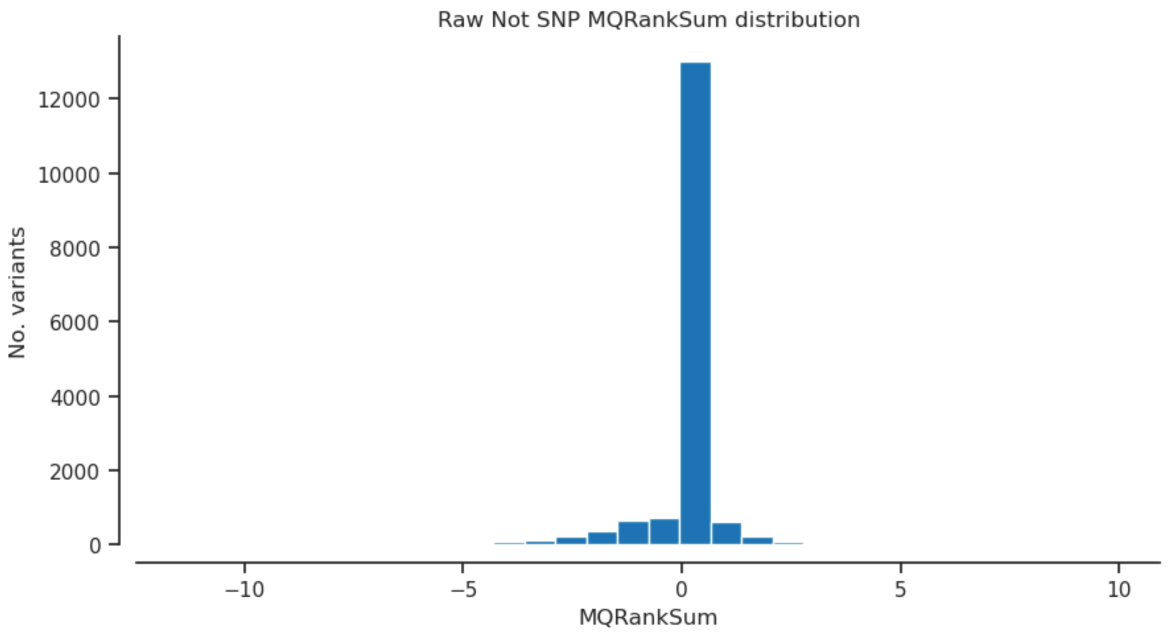


In [34]:
```
filter_expression = '(MQRankSum < -12.5)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

In [35]:
```
plot_hist('MQRankSum') # Z-score From Wilcoxon rank sum test of Alt vs. R
```

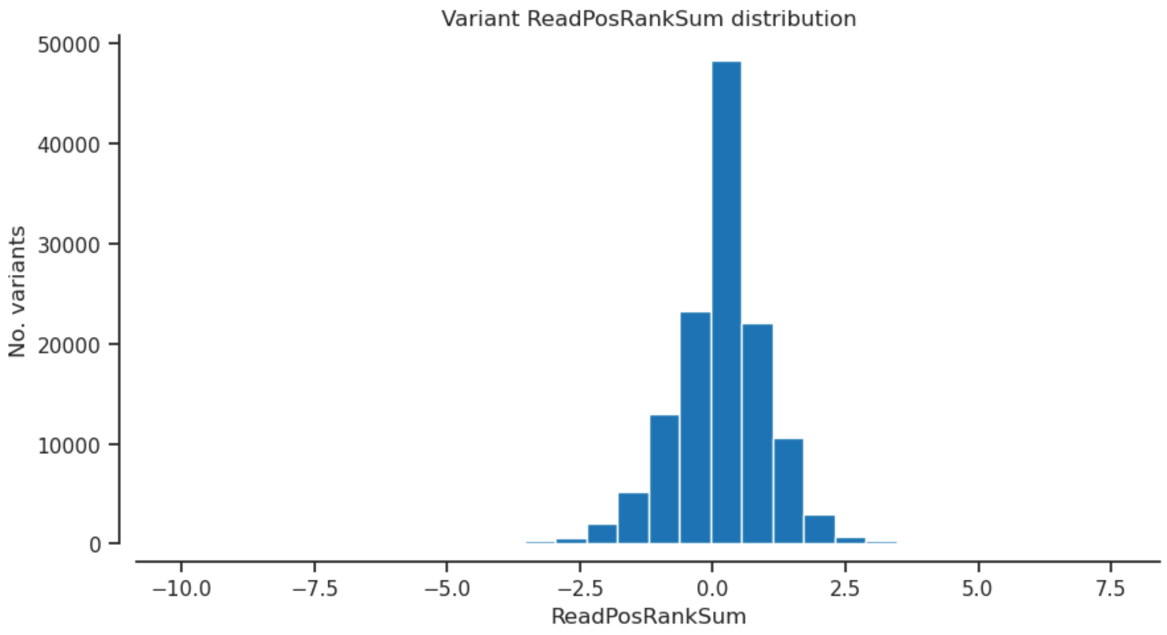


In [36]:
```
plot_hist('MQRankSum','notsnp') # Z-score From Wilcoxon rank sum test of
```

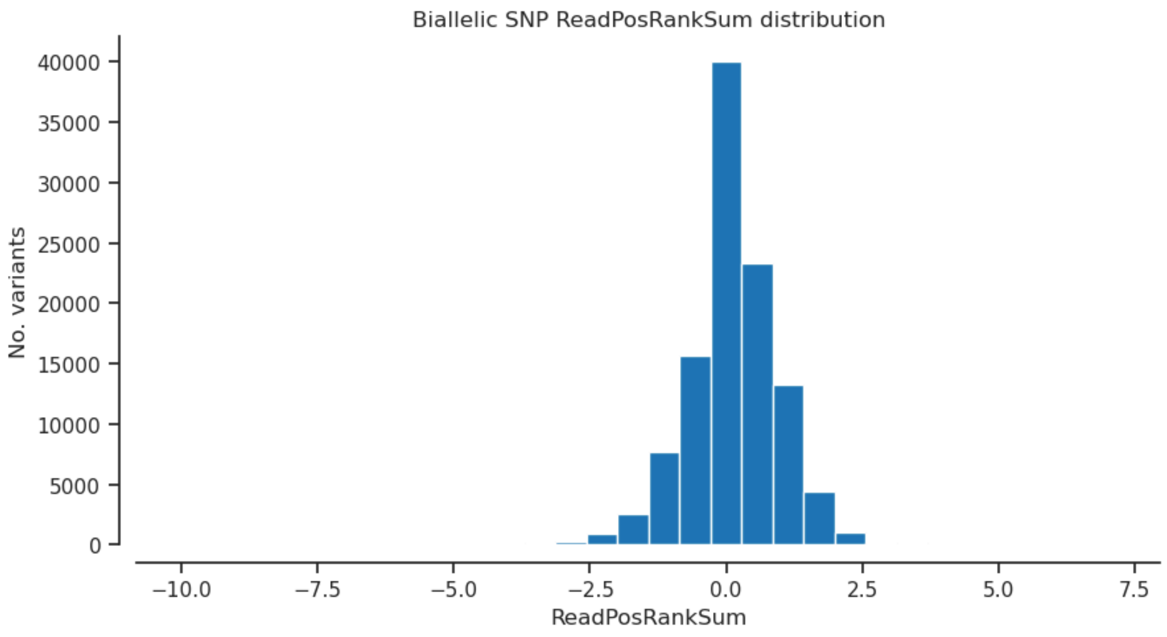


ReadPosRankSum - Z-score from Wilcoxon rank sum test of Alt vs. Ref read position bias

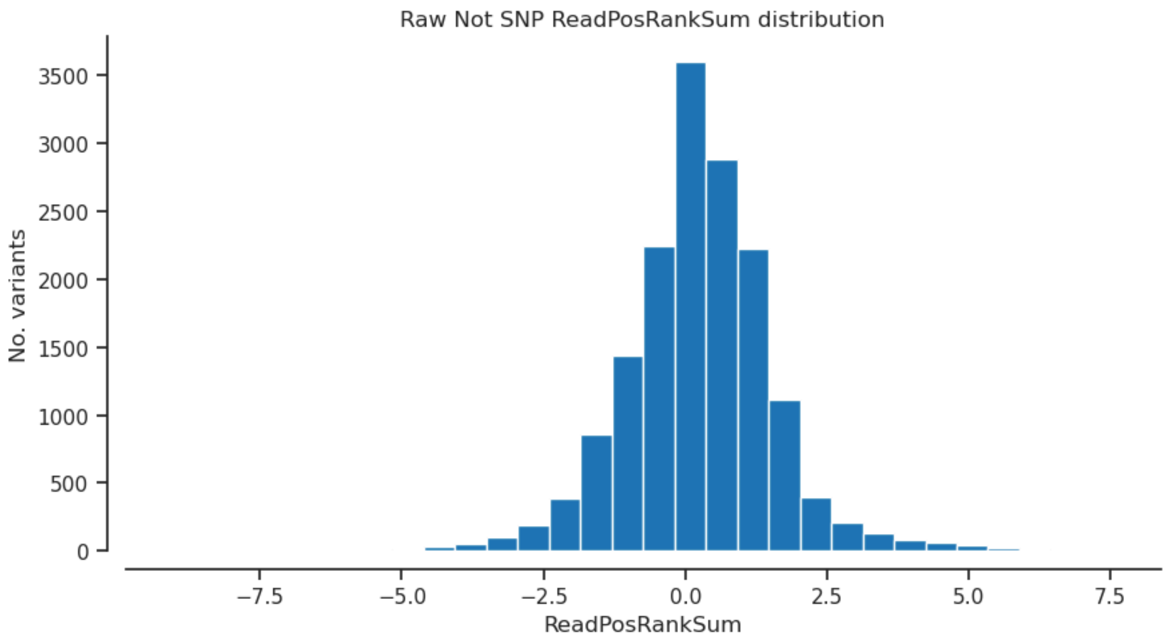
In [37]:
```
plot_hist('ReadPosRankSum','var') # Z-score from Wilcoxon rank sum test o
```



In [38]:
```
plot_hist('ReadPosRankSum','biallelic') # Z-score from Wilcoxon rank sum
```

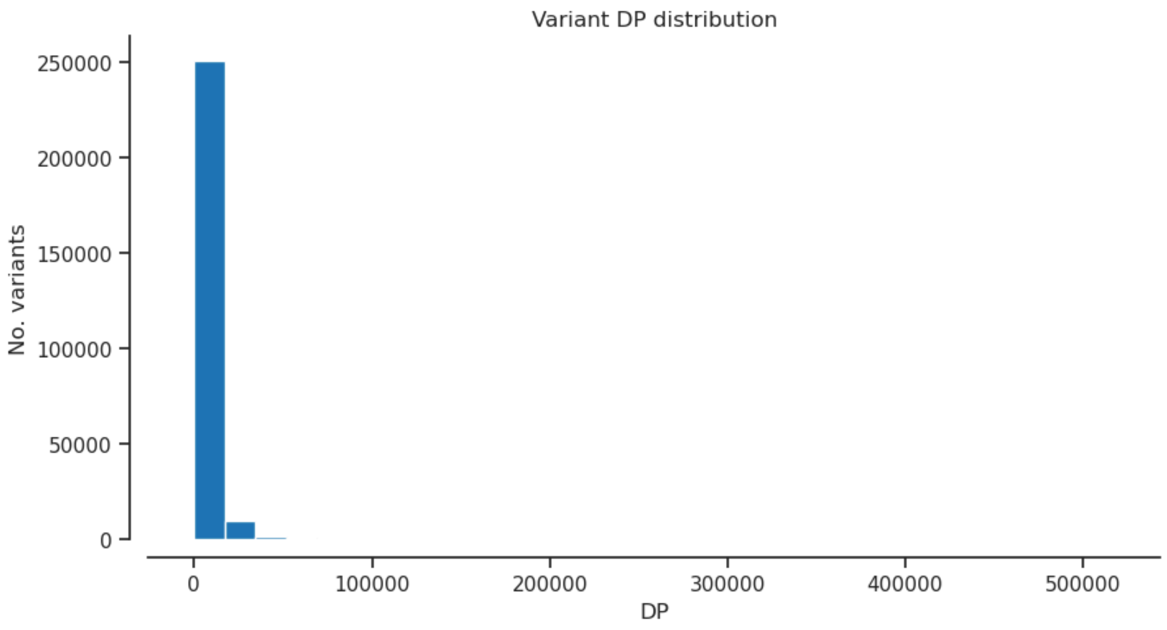


```
In [39]: plot_hist('ReadPosRankSum','notsnp') # Z-score from Wilcoxon rank sum tes
```

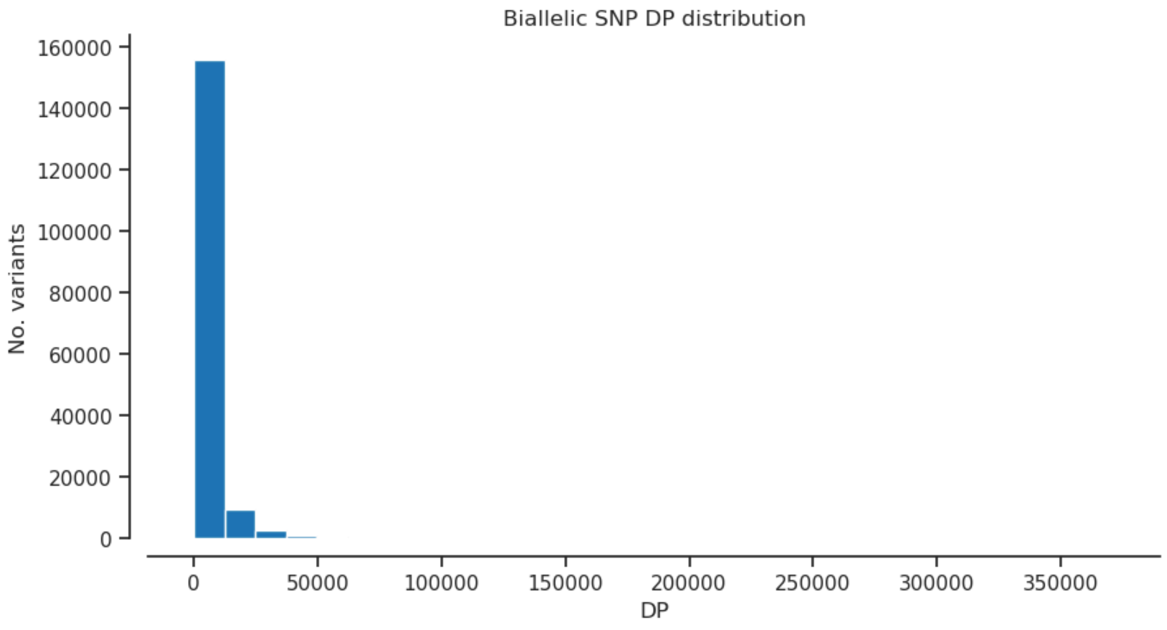


DP - Approximate read depth

```
In [40]: plot_hist('DP','var')
```

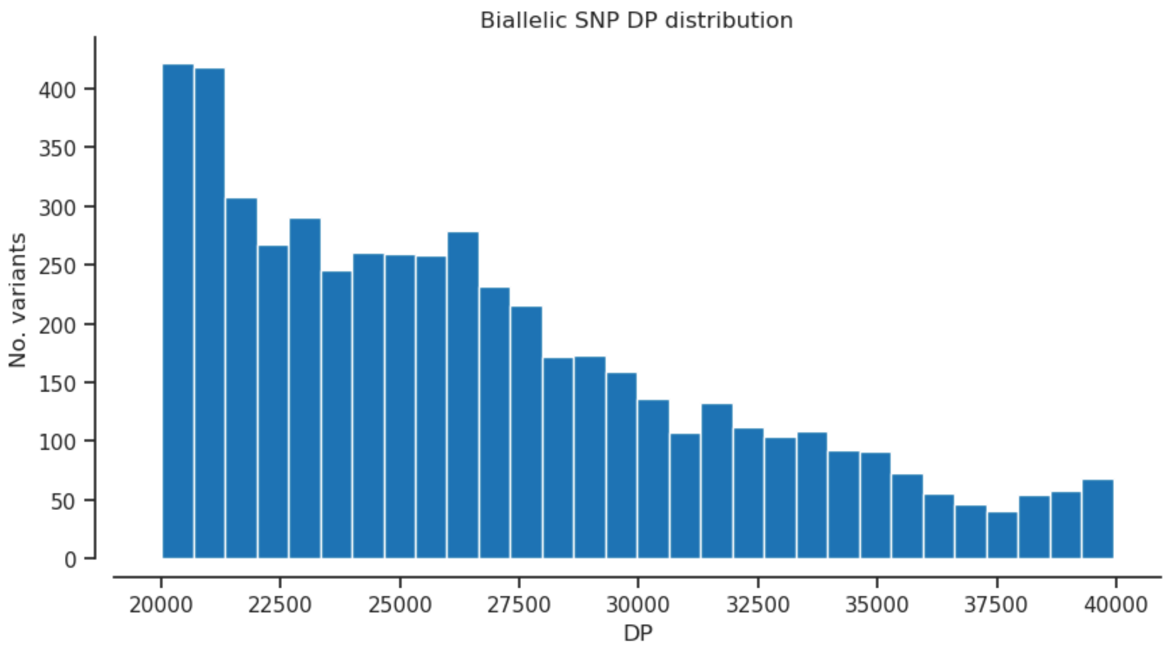


```
In [41]: plot_hist('DP','biallelic')
```

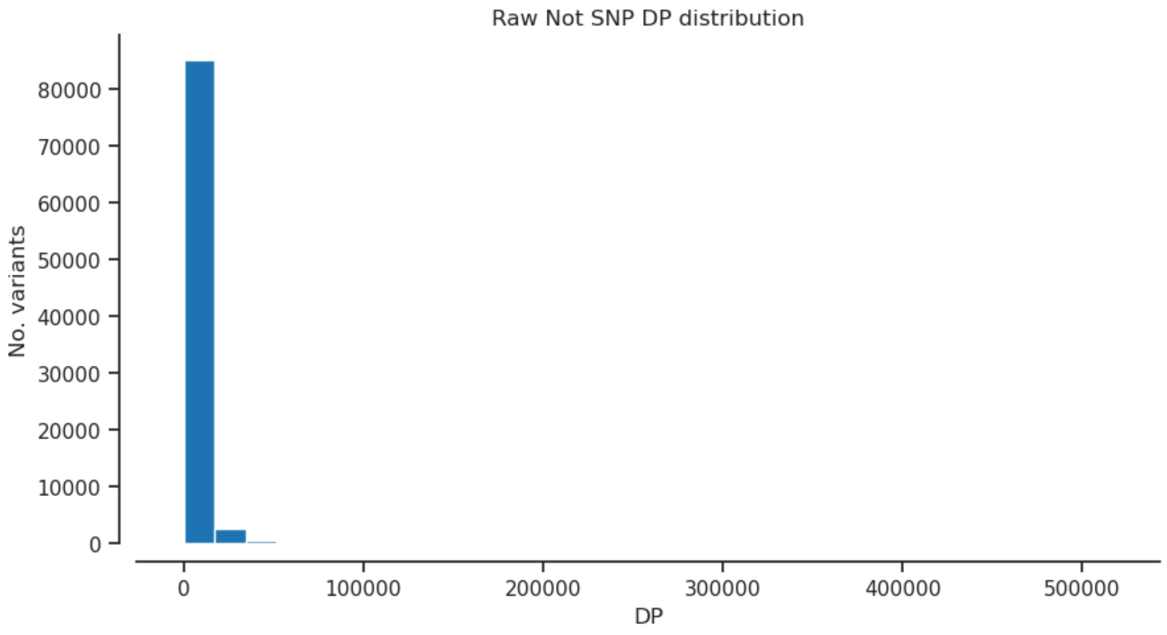


```
In [42]: filter_expression = '(DP > 20000) & (DP < 40000)'
         bi_selection = biallelic_np.query(filter_expression)[:]
```

```
In [43]: plot_hist('DP')
```

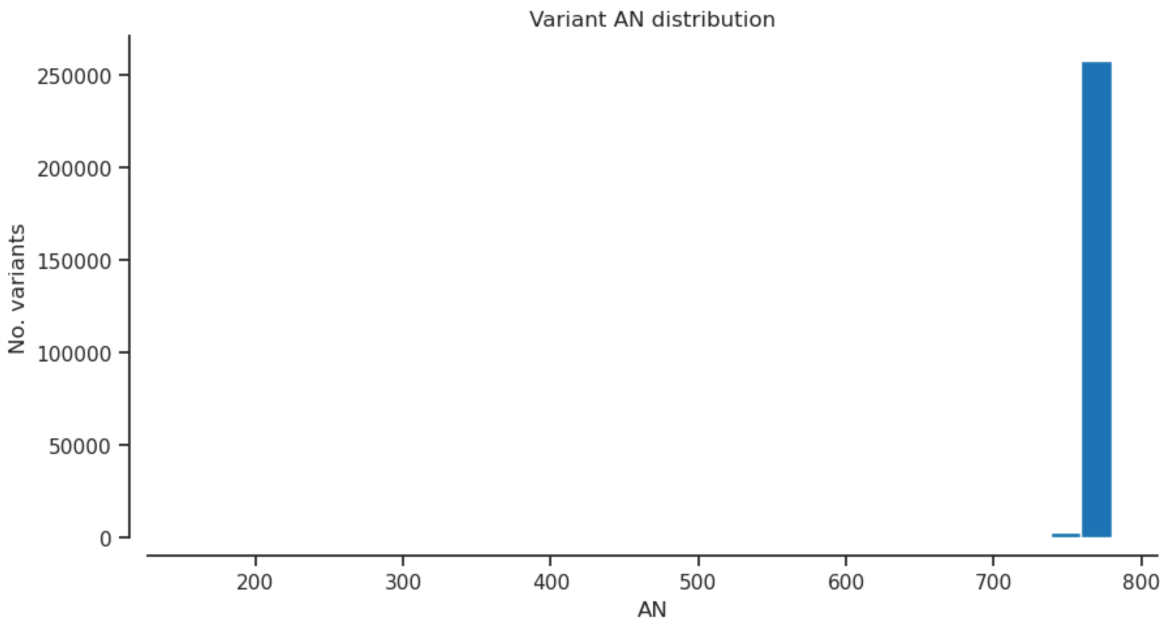


```
In [44]: plot_hist('DP','notsnp')
```

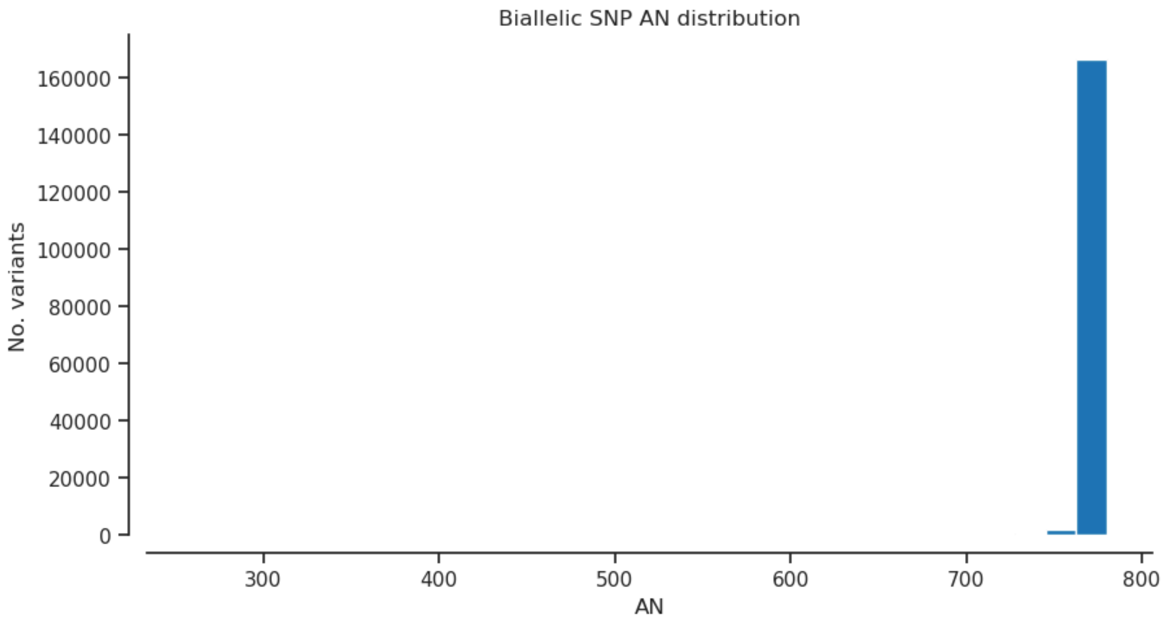


AN - Total number of alleles in called genotypes

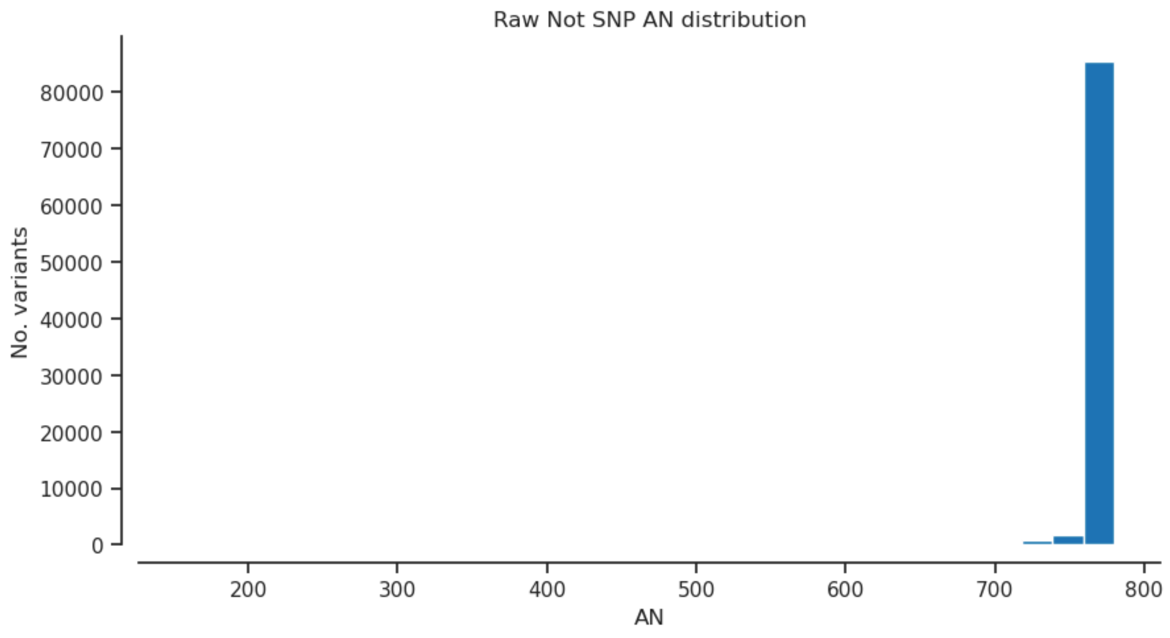
```
In [45]: plot_hist('AN','var') # Total number of alleles in called genotypes
```



```
In [46]: plot_hist('AN','biallelic') # Total number of alleles in called genotypes
```



```
In [47]: plot_hist('AN','notsnp') # Total number of alleles in called genotypes
```



Selected filter

```
In [48]: # QD: Variant Confidence/Quality by Depth
         # AN: Total number of alleles in called genotypes
         filter_expression = '(QD >= 2) & (MQ >= 40) & (MQRankSum >= -12.5) & (is_
         variant_selection = variants_np.eval(filter_expression)[:]
         np.count_nonzero(variant_selection)
```

```
Out[48]: 107664
```

Genotype

```
In [49]: calldata_var = callset_var['calldata']
         list(calldata_var)
```

```
Out[49]: ['AD', 'DP', 'GQ', 'GT', 'MIN_DP', 'PGT', 'PID', 'PL', 'PS', 'RGQ', 'S
         B']
```

```
In [50]: genotypes_var = allel.GenotypeChunkedArray(calldata_var['GT'])
         genotypes_var
```

Out[50]: <GenotypeChunkedArray shape=(262868, 390, 2) dtype=int8 chunks=(65536, 64, 2) nbytes=195.5M cbytes=11.3M cratio=17.3 compression=gzip compression_opts=1 values=h5py._hl.dataset.Dataset>

	0	1	2	3	4	...	385	386	387	388	389
0	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
1	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
2	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
...						...					
262865	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
262866	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
262867	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0

In [51]:
```python
# using the selected filters set above
gt_filtered_snps = genotypes_var.subset(variant_selection)
gt_filtered_snps
```

Out[51]: <GenotypeChunkedArray shape=(107664, 390, 2) dtype=int8 chunks=(1683, 390, 2) nbytes=80.1M cbytes=10.1M cratio=7.9 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3	4	...	385	386	387	388	389
0	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
1	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
2	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/1	0/0
...						...					
107661	0/0	0/0	0/0	1/2	0/0	...	0/0	0/0	0/0	0/0	0/0
107662	0/0	0/0	0/0	0/1	0/0	...	0/0	0/0	0/0	0/0	0/0
107663	0/0	0/0	0/0	0/1	0/0	...	0/0	0/0	0/0	0/0	0/0

In [52]:
```python
# grab the allele counts for the populations
ac = gt_filtered_snps.count_alleles()
ac
```

Out[52]: <AlleleCountsChunkedArray shape=(107664, 4) dtype=int32 chunks=(26916, 4) nbytes=1.6M cbytes=401.5K cratio=4.2 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3
0	779	1	0	0
1	778	2	0	0
2	779	1	0	0
...	...			
107661	775	1	2	0
107662	779	1	0	0
107663	779	1	0	0

In [53]:
```
ac[:]
```

Out[53]: <AlleleCountsArray shape=(107664, 4) dtype=int32>

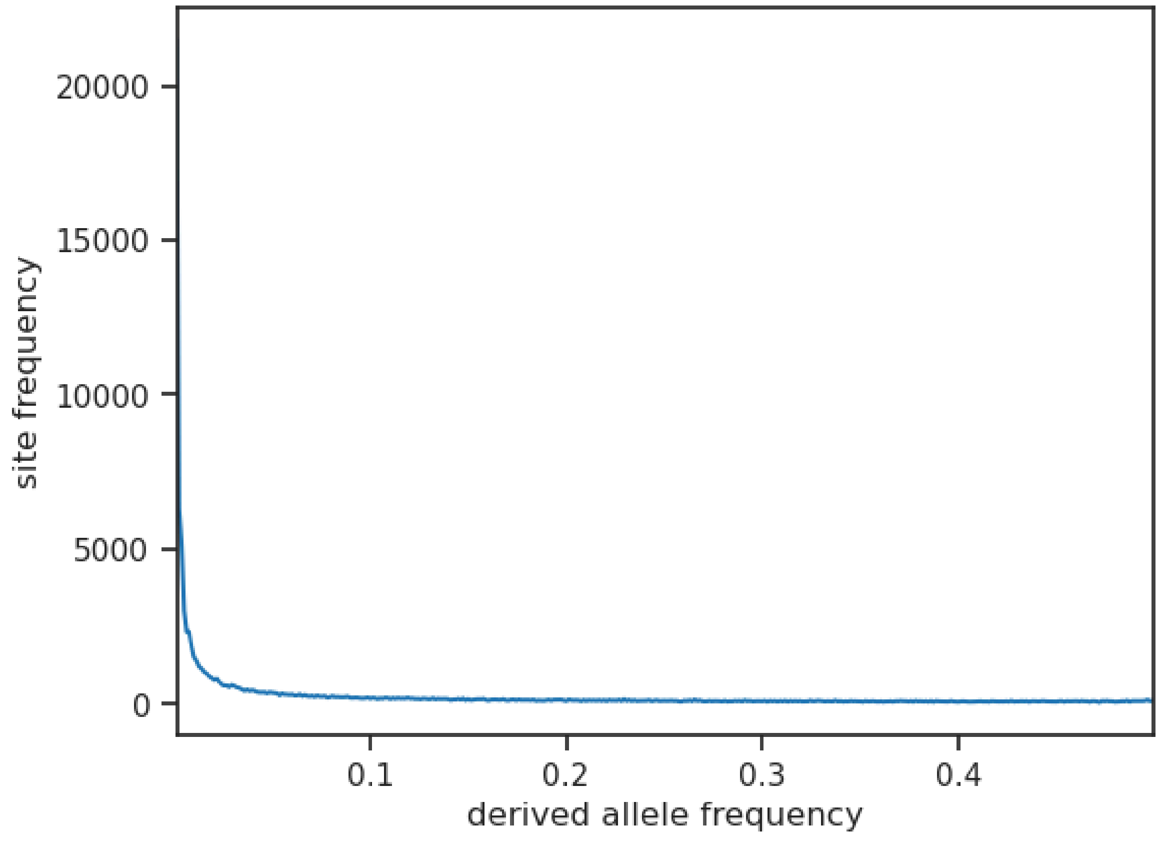
	0	1	2	3
0	779	1	0	0
1	778	2	0	0
2	779	1	0	0
...	...			
107661	775	1	2	0
107662	779	1	0	0
107663	779	1	0	0

In [54]:
```
# Which ones are biallelic?
is_biallelic_01 = ac.is_biallelic_01()[:]
ac1 = ac.compress(is_biallelic_01, axis=0)[:, :2]
ac1
##this part of the code is only for graphing the SFS, is not useful for f
```

Out[54]:
```
array([[779,   1],
       [778,   2],
       [779,   1],
       ...,
       [767,  11],
       [779,   1],
       [779,   1]], dtype=int32)
```

In [55]:
```
# plot the sfs of the derived allele
s = allel.sfs_folded(ac1)
allel.plot_sfs(s, yscale="linear", n=ac1.sum(axis=1).max())
```

Out[55]:
```
<Axes: xlabel='derived allele frequency', ylabel='site frequency'>
```



In [56]:
```
biallelic = (ac.max_allele() == 1)
###This is the filter expression for biallelic sites
biallelic
```

Out[56]:
```
<ChunkedArrayWrapper shape=(107664,) dtype=bool chunks=(107664,)
   nbytes=105.1K cbytes=15.1K cratio=7.0
   compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shu
ffle': 1, 'blocksize': 0}
   values=zarr.core.Array>
```

In [57]:
```
# select only the biallelic variants
gt_biallelic = gt_filtered_snps.compress(biallelic)
gt_biallelic
```

Out[57]: <GenotypeChunkedArray shape=(103830, 390, 2) dtype=int8 chunks=(1623, 390, 2) nbytes=77.2M cbytes=9.6M cratio=8.1 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3	4	...	385	386	387	388	389
0	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
1	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
2	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/1	0/0
...						...					
103827	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
103828	0/0	0/0	0/0	0/1	0/0	...	0/0	0/0	0/0	0/0	0/0
103829	0/0	0/0	0/0	0/1	0/0	...	0/0	0/0	0/0	0/0	0/0

In [58]:
```
n_variants = len(gt_biallelic)
n_variants
```

Out[58]:
```
103830
```

In [59]:
```
pc_missing = gt_biallelic.count_missing(axis=0)[:] * 100 / n_variants
pc_het = gt_biallelic.count_het(axis=0)[:] * 100 / n_variants
```

Samples

In [60]:
```
samples_var = callset_var['samples']
samples_var = list(samples_var)
samples_var
```

```
Out[60]: [b'ESP00046-001',
          b'ESP00046-002',
          b'ESP00046-003',
          b'ESP00046-004',
          b'ESP00046-005',
          b'ESP00046-006',
          b'ESP00046-007',
          b'ESP00046-008',
          b'ESP00046-009',
          b'ESP00046-010',
          b'ESP00046-011',
          b'ESP00046-012',
          b'ESP00046-013',
          b'ESP00046-014',
          b'ESP00046-015',
          b'ESP00046-016',
          b'ESP00046-017',
          b'ESP00046-018',
          b'ESP00046-019',
          b'ESP00046-020',
          b'ESP00046-021',
          b'ESP00046-022',
          b'ESP00046-023',
          b'ESP00046-024',
          b'ESP00046-025',
          b'ESP00059-001',
          b'ESP00059-002',
          b'ESP00059-003',
          b'ESP00059-004',
          b'ESP00059-005',
          b'ESP00059-006',
          b'ESP00059-007',
          b'ESP00059-008',
          b'ESP00059-009',
          b'ESP00059-010',
          b'ESP00059-011',
          b'ESP00059-012',
          b'ESP00059-013',
          b'ESP00059-014',
          b'ESP00059-015',
          b'ESP00059-016',
          b'ESP00059-017',
          b'ESP00059-018',
          b'ESP00059-019',
          b'ESP00059-020',
          b'ESP00059-021',
          b'ESP00059-022',
          b'ESP00059-023',
          b'ESP00059-024',
          b'ESP00059-025',
          b'ESP00080-001',
          b'ESP00080-002',
          b'ESP00080-003',
          b'ESP00080-004',
          b'ESP00080-005',
          b'ESP00080-006',
          b'ESP00080-007',
          b'ESP00080-008',
          b'ESP00080-009',
          b'ESP00080-010',
          b'ESP00080-011',
          b'ESP00080-012',
          b'ESP00080-013',
          b'ESP00080-014',
          b'ESP00080-015',
          b'ESP00080-016',
          b'ESP00080-017',
          b'ESP00080-018',
          b'ESP00080-019',
          b'ESP00080-020',
          b'ESP00080-021',
          b'ESP00080-022',
          b'ESP00080-023',
          b'ESP00080-024',
          b'ESP00080-025',
          b'ESP00084-001',
          b'ESP00084-002',
          b'ESP00084-003',
          b'ESP00084-004',
          b'ESP00084-005',
          b'ESP00084-006',
          b'ESP00084-007',
          b'ESP00084-008',
          b'ESP00084-009',
          b'ESP00084-010',
          b'ESP00084-011',
          b'ESP00084-012',
          b'ESP00084-013',
          b'ESP00084-014',
          b'ESP00084-015',
          b'ESP00084-016',
          b'ESP00084-017',
          b'ESP00084-018',
          b'ESP00084-019',
          b'ESP00084-020',
          b'ESP00084-021',
          b'ESP00084-022',
          b'ESP00084-023',
          b'ESP00084-024',
          b'ESP00084-025',
          b'ESP00096-001',
          b'ESP00096-002',
          b'ESP00096-003',
          b'ESP00096-004',
          b'ESP00096-005',
          b'ESP00096-006',
          b'ESP00096-007',
          b'ESP00096-008',
          b'ESP00096-009',
          b'ESP00096-010',
          b'ESP00096-011',
          b'ESP00096-012',
          b'ESP00096-013',
          b'ESP00096-014',
          b'ESP00096-015',
          b'ESP00096-016',
          b'ESP00096-017',
          b'ESP00096-018',
          b'ESP00096-019',
          b'ESP00096-020',
          b'ESP00096-021',
          b'ESP00096-022',
          b'ESP00096-023',
          b'ESP00096-024',
          b'ESP00096-025',
          b'ESP00099-001',
          b'ESP00099-002',
          b'ESP00099-003',
          b'ESP00099-004',
          b'ESP00099-005',
          b'ESP00099-006',
          b'ESP00099-007',
          b'ESP00099-008',
          b'ESP00099-009',
          b'ESP00099-010',
          b'ESP00099-011',
          b'ESP00099-012',
          b'ESP00099-013',
          b'ESP00099-014',
          b'ESP00099-015',
          b'ESP00099-016',
          b'ESP00099-017',
          b'ESP00099-018',
          b'ESP00099-019',
          b'ESP00099-020',
          b'ESP00099-021',
          b'ESP00099-022',
          b'ESP00099-023',
          b'ESP00099-024',
          b'ESP00099-025',
          b'ESP00163-001',
          b'ESP00163-002',
          b'ESP00163-003',
          b'ESP00163-004',
          b'ESP00163-005',
          b'ESP00163-006',
          b'ESP00163-007',
          b'ESP00163-008',
          b'ESP00163-009',
          b'ESP00163-010',
          b'ESP00163-011',
          b'ESP00163-012',
          b'ESP00163-013',
          b'ESP00163-014',
          b'ESP00163-015',
          b'ESP00163-016',
          b'ESP00163-017',
          b'ESP00163-018',
          b'ESP00163-019',
          b'ESP00163-020',
          b'ESP00163-021',
          b'ESP00163-022',
          b'ESP00163-023',
          b'ESP00163-024',
          b'ESP00163-025',
          b'ESP00183-001',
          b'ESP00183-002',
          b'ESP00183-003',
          b'ESP00183-004',
          b'ESP00183-005',
          b'ESP00183-006',
          b'ESP00183-007',
          b'ESP00183-008',
          b'ESP00183-009',
          b'ESP00183-010',
          b'ESP00183-011',
          b'ESP00183-012',
          b'ESP00183-013',
          b'ESP00183-014',
          b'ESP00183-015',
          b'ESP00183-016',
          b'ESP00183-017',
          b'ESP00183-018',
          b'ESP00183-019',
          b'ESP00183-020',
          b'ESP00183-021',
          b'ESP00183-022',
          b'ESP00183-023',
          b'ESP00183-024',
          b'ESP00183-025',
          b'ESP00218-001',
          b'ESP00218-002',
          b'ESP00218-003',
          b'ESP00218-004',
          b'ESP00218-005',
          b'ESP00218-006',
          b'ESP00218-007',
          b'ESP00218-008',
          b'ESP00218-009',
          b'ESP00218-010',
          b'ESP00218-011',
          b'ESP00218-012',
          b'ESP00218-013',
          b'ESP00218-014',
          b'ESP00218-015',
          b'ESP00218-016',
          b'ESP00218-017',
          b'ESP00218-018',
          b'ESP00218-019',
          b'ESP00218-020',
          b'ESP00218-021',
          b'ESP00218-022',
          b'ESP00218-023',
          b'ESP00218-024',
          b'ESP00218-025',
          b'ESP00270-001',
          b'ESP00270-002',
          b'ESP00270-003',
          b'ESP00270-004',
          b'ESP00270-005',
          b'ESP00270-006',
          b'ESP00270-007',
          b'ESP00270-008',
          b'ESP00270-009',
          b'ESP00270-010',
          b'ESP00270-011',
          b'ESP00270-012',
          b'ESP00270-013',
          b'ESP00270-014',
          b'ESP00270-015',
          b'ESP00270-016',
          b'ESP00270-017',
          b'ESP00270-018',
          b'ESP00270-019',
          b'ESP00270-020',
          b'ESP00270-021',
          b'ESP00270-022',
          b'ESP00270-023',
          b'ESP00270-024',
          b'ESP00270-025',
          b'ESP00288-001',
          b'ESP00288-002',
          b'ESP00288-003',
          b'ESP00288-004',
          b'ESP00288-005',
          b'ESP00288-006',
          b'ESP00288-007',
          b'ESP00288-008',
          b'ESP00288-009',
          b'ESP00288-010',
          b'ESP00288-011',
          b'ESP00288-012',
          b'ESP00288-013',
          b'ESP00288-014',
          b'ESP00288-015',
          b'ESP00288-016',
          b'ESP00288-017',
          b'ESP00288-018',
          b'ESP00288-019',
          b'ESP00288-020',
          b'ESP00288-021',
          b'ESP00288-022',
          b'ESP00288-023',
          b'ESP00288-024',
          b'ESP00288-025',
          b'ESP00358-001',
          b'ESP00358-002',
          b'ESP00358-003',
          b'ESP00358-004',
          b'ESP00358-005',
          b'ESP00358-006',
          b'ESP00358-007',
          b'ESP00358-008',
          b'ESP00358-009',
          b'ESP00358-010',
          b'ESP00358-011',
          b'ESP00358-012',
          b'ESP00358-013',
          b'ESP00358-014',
          b'ESP00358-015',
          b'ESP00358-016',
          b'ESP00358-017',
          b'ESP00358-018',
          b'ESP00358-019',
          b'ESP00358-020',
          b'ESP00358-021',
          b'ESP00358-022',
          b'ESP00358-023',
          b'ESP00358-024',
          b'ESP00358-025',
          b'ESP00372-001',
          b'ESP00372-002',
          b'ESP00372-003',
          b'ESP00372-004',
          b'ESP00372-005',
          b'ESP00372-006',
          b'ESP00372-007',
          b'ESP00372-008',
          b'ESP00372-009',
          b'ESP00372-010',
          b'ESP00372-011',
          b'ESP00372-012',
          b'ESP00372-013',
          b'ESP00372-014',
          b'ESP00372-015',
          b'ESP00382-001',
          b'ESP00382-002',
          b'ESP00382-003',
          b'ESP00382-004',
          b'ESP00382-005',
          b'ESP00382-006',
          b'ESP00382-007',
          b'ESP00382-008',
          b'ESP00382-009',
          b'ESP00382-010',
          b'ESP00382-011',
          b'ESP00382-012',
          b'ESP00382-013',
          b'ESP00382-014',
          b'ESP00382-015',
          b'ESP00382-016',
          b'ESP00382-017',
          b'ESP00382-018',
          b'ESP00382-019',
          b'ESP00382-020',
          b'ESP00382-021',
          b'ESP00382-022',
          b'ESP00382-023',
          b'ESP00382-024',
          b'ESP00382-025',
          b'ESP00394-001',
          b'ESP00394-002',
          b'ESP00394-003',
          b'ESP00394-004',
          b'ESP00394-005',
          b'ESP00394-006',
          b'ESP00394-007',
          b'ESP00394-008',
          b'ESP00394-009',
          b'ESP00394-010',
          b'ESP00394-011',
          b'ESP00394-012',
          b'ESP00394-013',
          b'ESP00394-014',
          b'ESP00394-015',
          b'ESP00394-016',
          b'ESP00394-017',
          b'ESP00394-018',
          b'ESP00394-019',
          b'ESP00394-020',
          b'ESP00394-021',
          b'ESP00394-022',
          b'ESP00394-023',
          b'ESP00394-024',
          b'ESP00394-025',
          b'ITA00268-001',
          b'ITA00268-002',
          b'ITA00268-003',
          b'ITA00268-004',
          b'ITA00268-005',
          b'ITA00268-006',
          b'ITA00268-007',
          b'ITA00268-008',
          b'ITA00268-009',
          b'ITA00268-010',
          b'ITA00268-011',
          b'ITA00268-012',
          b'ITA00268-013',
          b'ITA00268-014',
          b'ITA00268-015',
          b'ITA00268-016',
          b'ITA00268-017',
          b'ITA00268-018',
          b'ITA00268-019',
          b'ITA00268-020',
          b'ITA00268-021',
          b'ITA00268-022',
          b'ITA00268-023',
          b'ITA00268-024',
          b'ITA00268-025']
```

In [61]:
```
samples_fn = '~/scratch/data/Qsuber/Quercus_suber_sample_list_scikit-alle
samples = pandas.read_csv(samples_fn, sep='\t')
samples
```

Out[61]:

	ID	Population
0	ESP00046-001	ESP00046
1	ESP00046-002	ESP00046
2	ESP00046-003	ESP00046
3	ESP00046-004	ESP00046
4	ESP00046-005	ESP00046
...	...	...
385	ITA00268-021	ITA00268
386	ITA00268-022	ITA00268
387	ITA00268-023	ITA00268
388	ITA00268-024	ITA00268
389	ITA00268-025	ITA00268

390 rows × 2 columns

In [62]:
```
samples.Population.value_counts()
```

Out[62]:
```
Population
ESP00046    25
ESP00059    25
ESP00080    25
ESP00084    25
ESP00096    25
ESP00099    25
ESP00163    25
ESP00183    25
ESP00218    25
ESP00270    25
ESP00288    25
ESP00358    25
ESP00382    25
ESP00394    25
ITA00268    25
ESP00372    15
Name: count, dtype: int64
```

In [63]:
```
populations = samples.Population.unique()
populations
###This identifiers come from the metadata file
```

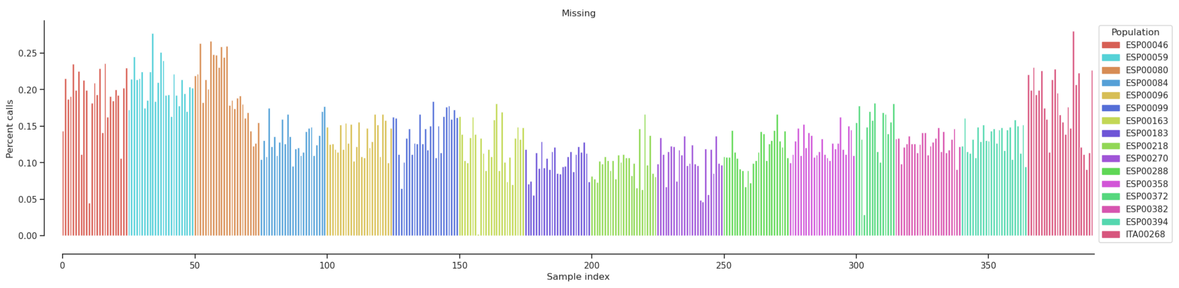
Out[63]:
```
array(['ESP00046', 'ESP00059', 'ESP00080', 'ESP00084', 'ESP00096',
       'ESP00099', 'ESP00163', 'ESP00183', 'ESP00218', 'ESP00270',
       'ESP00288', 'ESP00358', 'ESP00372', 'ESP00382', 'ESP00394',
       'ITA00268'], dtype=object)
```

Gt frequency function

In [64]:
```
def plot_genotype_frequency(pc, title):
    fig, ax = plt.subplots(figsize=(24, 5))
    sns.despine(ax=ax, offset=24)
    left = np.arange(len(pc))
    palette = sns.color_palette("hls", 16)
    pop2color = {'ESP00046': palette[0],
                 'ESP00059': palette[8],
                 'ESP00080': palette[1],
                 'ESP00084': palette[9],
                 'ESP00096': palette[2],
                 'ESP00099': palette[10],
                 'ESP00163': palette[3],
                 'ESP00183': palette[11],
                 'ESP00218': palette[4],
                 'ESP00270': palette[12],
                 'ESP00288': palette[5],
                 'ESP00358': palette[13],
                 'ESP00372': palette[6],
                 'ESP00382': palette[14],
                'ESP00394': palette[7],
                'ITA00268': palette[15]}
    colors = [pop2color[p] for p in samples.Population]
    ax.bar(left, pc, color=colors)
    ax.set_xlim(0, len(pc))
    ax.set_xlabel('Sample index')
    ax.set_ylabel('Percent calls')
    ax.set_title(title)
    handles = [mpl.patches.Patch(color=palette[0]),
               mpl.patches.Patch(color=palette[8]),
               mpl.patches.Patch(color=palette[1]),
               mpl.patches.Patch(color=palette[9]),
               mpl.patches.Patch(color=palette[2]),
               mpl.patches.Patch(color=palette[10]),
               mpl.patches.Patch(color=palette[3]),
               mpl.patches.Patch(color=palette[11]),
               mpl.patches.Patch(color=palette[4]),
               mpl.patches.Patch(color=palette[12]),
               mpl.patches.Patch(color=palette[5]),
               mpl.patches.Patch(color=palette[13]),
               mpl.patches.Patch(color=palette[6]),
              mpl.patches.Patch(color=palette[14]),
              mpl.patches.Patch(color=palette[7]),
              mpl.patches.Patch(color=palette[15])]
    ax.legend(handles=handles, labels=['ESP00046', 'ESP00059', 'ESP00080'
       'ESP00099', 'ESP00163', 'ESP00183', 'ESP00218', 'ESP00270',
       'ESP00288', 'ESP00358', 'ESP00372', 'ESP00382', 'ESP00394', 'ITA00
              bbox_to_anchor=(1, 1), loc='upper left')
```

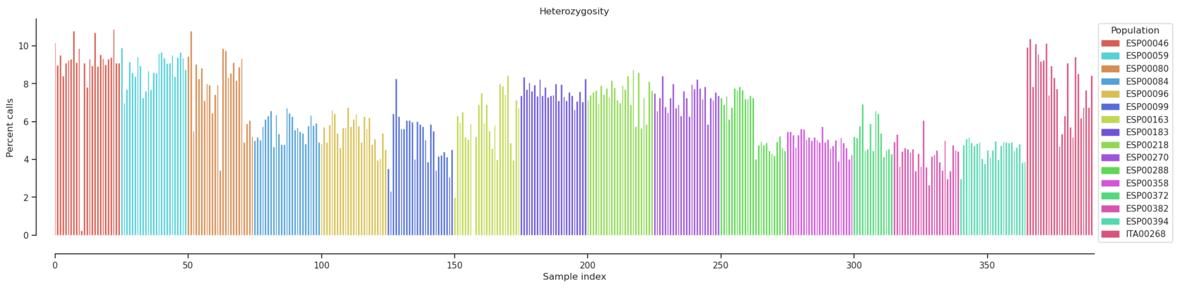
Plot missing

In [65]:
```
plot_genotype_frequency(pc_missing, 'Missing')
```



Plot heterozygosity

In [66]:
```
plot_genotype_frequency(pc_het, 'Heterozygosity')
```



PCA

In [67]:
```
palette = sns.color_palette("hls",16)
pop_colours = {
                'ESP00046': palette[0],
                 'ESP00059': palette[8],
                 'ESP00080': palette[1],
                 'ESP00084': palette[9],
                 'ESP00096': palette[2],
                 'ESP00099': palette[10],
                 'ESP00163': palette[3],
                 'ESP00183': palette[11],
                 'ESP00218': palette[4],
                 'ESP00270': palette[12],
                 'ESP00288': palette[5],
                 'ESP00358': palette[13],
                 'ESP00372': palette[6],
                 'ESP00382': palette[14],
                'ESP00394': palette[7],
                'ITA00268': palette[15]
}
```

In [68]:
```
def plot_pca_coords(coords, model, pc1, pc2, ax, sample_population):
    sns.despine(ax=ax, offset=5)
    x = coords[:, pc1]
    y = coords[:, pc2]
    for pop in populations:
        flt = (sample_population == pop)
        ax.plot(x[flt], y[flt], marker='o', linestyle=' ', color=pop_colo
                label=pop, markersize=6, mec='k', mew=.5)
    ax.set_xlabel('PC%s (%.1f%%)' % (pc1+1, model.explained_variance_rati
    ax.set_ylabel('PC%s (%.1f%%)' % (pc2+1, model.explained_variance_rati


def fig_pca(coords, model, title, sample_population=None):
    if sample_population is None:
        sample_population = samples.Population
    # plot coords for PCs 1 vs 2, 3 vs 4
    fig = plt.figure(figsize=(10, 5))
    ax = fig.add_subplot(1, 2, 1)
    plot_pca_coords(coords, model, 0, 1, ax, sample_population)
    ax = fig.add_subplot(1, 2, 2)
    plot_pca_coords(coords, model, 2, 3, ax, sample_population)
    ax.legend(bbox_to_anchor=(1, 1), loc='upper left')
    fig.suptitle(title, y=1.02)
    fig.tight_layout()
```

In [69]:
```
ac2 = gt_biallelic.count_alleles()
ac2
```

Out[69]:
```
<AlleleCountsChunkedArray shape=(103830, 2) dtype=int32 chunks=(25958, 2)
nbytes=811.2K cbytes=304.9K cratio=2.7 compression=blosc compression_opts=
{'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>
```

	0	1
0	779	1
1	778	2
2	779	1
...	...	
103827	767	11
103828	779	1
103829	779	1

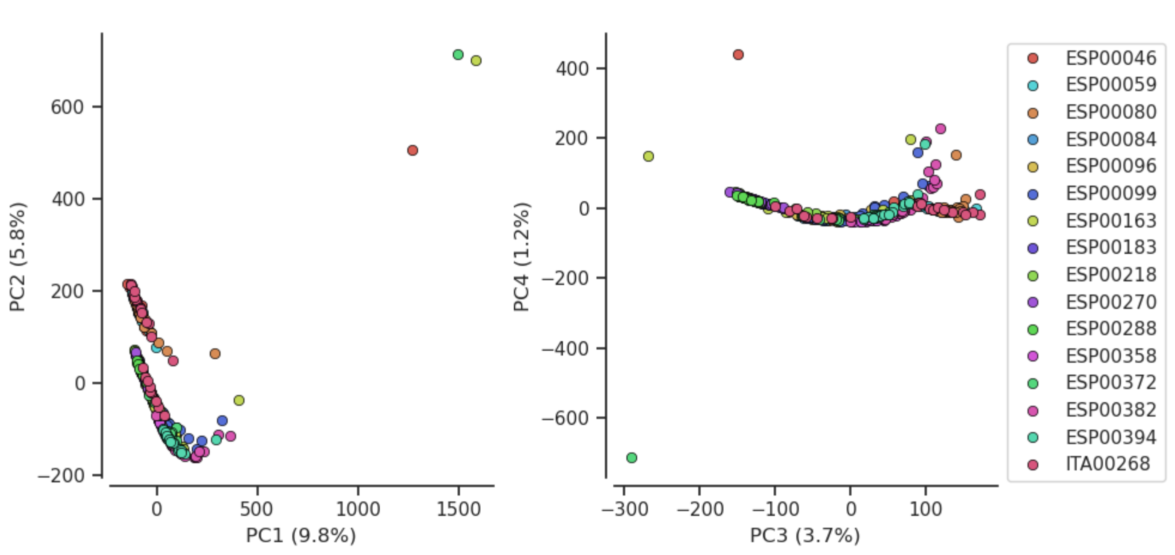
In [70]:
```
flt = (ac2[:, :2].min(axis=1) > 1)
gf = gt_biallelic.compress(flt, axis=0)
gn = gf.to_n_alt()
gn
```

Out[70]:
```
<ChunkedArrayWrapper shape=(82346, 390) dtype=int8 chunks=(2574, 390)
   nbytes=30.6M cbytes=6.2M cratio=4.9
   compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shu
ffle': 1, 'blocksize': 0}
   values=zarr.core.Array>
```

In [71]:
```
coords1, model1 = allel.pca(gn, n_components=10, scaler='patterson')
```

In [72]:
```
fig_pca(coords1, model1, 'Figure 1. Conventional PCA.')
```

Figure 1. Conventional PCA.



In [73]:
```
outliers = coords1[:,0]>1000
samples[outliers]
```

Out[73]:

	ID	Population
10	ESP00046-011	ESP00046
157	ESP00163-008	ESP00163
303	ESP00372-004	ESP00372

In [76]:
```
outliers2 = coords1[:,2]<-300
samples[outliers2]
```

Out[76]:

	ID	Population
157	ESP00163-008	ESP00163

In [74]:
```
pc_het[outliers]
```

Out[74]:
```
array([0.28154934, 0.01122117])
```

In [75]:
```
pc_missing[outliers]
```

Out[75]:
```
array([0.0489651 , 0.00204021])
```

In [ ]:
```
```