In [ ]:
```
import sys
!{sys.executable} -m pip install --user scikit-allel
```

In [1]:
```
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 [2]:
```
allel.vcf_to_hdf5('/users/mcevoysu/scratch/output/Bpendula/vcf_filtering/
```

Get data

In [3]:
```
callset_var_fn = '/users/mcevoysu/scratch/output/Bpendula/scikit-allel/ra
callset_var = h5py.File(callset_var_fn, mode='r')
```

In [4]:
```
calldata_var = callset_var['calldata']
list(calldata_var)
```

Out[4]:
```
['AD', 'DP', 'GQ', 'GT', 'MIN_DP', 'PGT', 'PID', 'PL', 'PS', 'RGQ', 'S
B']
```

In [5]:
```
list(callset_var['variants'])
```

Out[5]:
```
['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 [6]:
```
variants = allel.VariantChunkedTable(callset_var['variants'])
variants
```

Out[6]: <VariantChunkedTable shape=(289868,) 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=49.5M cbytes=11.0M cratio=4.5 values=h5py._hl.group.Group>

	AC	AF	ALT	AN	BaseQRankSum	CHROM	DP	END	Ex
0	[23 9 -1]	[0.031 0.012 nan]	[b'T' b'A' b'']	728	0.175	b'Contig0'	16605	-1	
1	[1 -1 -1]	[0.001351 nan nan]	[b'*' b'' b'']	728	-0.979	b'Contig0'	16619	-1	
2	[1 -1 -1]	[0.001351 nan nan]	[b'C' b'' b'']	728	0.962	b'Contig0'	16693	-1	
...									
289865	[2 -1 -1]	[0.002703 nan nan]	[b'A' b'' b'']	728	nan	b'Contig999'	65	-1	
289866	[4 -1 -1]	[0.005405 nan nan]	[b'G' b'' b'']	728	nan	b'Contig999'	64	-1	
289867	[2 -1 -1]	[0.002703 nan nan]	[b'T' b'' b'']	728	nan	b'Contig999'	63	-1	

In [7]:
```
variants_np = variants[:]
rawsnps = variants_np.query('(is_snp == True)')
rawsnps
```

Out[7]: <VariantTable shape=(178476,) 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	[23 9 -1]	[0.031 0.012 nan]	[b'T' b'A' b'']	728	0.175	b'Contig0'	16605	-1	
1	[1 -1 -1]	[0.001351 nan nan]	[b'C' b'' b'']	728	0.962	b'Contig0'	16693	-1	
2	[478 -1 -1]	[0.654 nan nan]	[b'G' b'' b'']	728	-0.043	b'Contig0'	17345	-1	
...									
178473	[2 -1 -1]	[0.002703 nan nan]	[b'A' b'' b'']	728	nan	b'Contig999'	65	-1	
178474	[4 -1 -1]	[0.005405 nan nan]	[b'G' b'' b'']	728	nan	b'Contig999'	64	-1	
178475	[2 -1 -1]	[0.002703 nan nan]	[b'T' b'' b'']	728	nan	b'Contig999'	63	-1	

In [8]:
```
notsnp = variants_np.query('(is_snp != True)')
notsnp
```

Out[8]: <VariantTable shape=(111392,) 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]	[0.001351 nan nan]	[b'*' b'' b'']	728	-0.979	b'Contig0'	16619	-1	
1	[3 -1 -1]	[0.004054 nan nan]	[b'*' b'' b'']	728	nan	b'Contig0'	16752	-1	0
2	[7 3 -1]	[0.009459 0.004054 nan]	[b'G' b'*' b'']	728	-0.873	b'Contig0'	16787	-1	0.
...									
111389	[2 -1 -1]	[0.002703 nan nan]	[b'*' b'' b'']	728	nan	b'Contig999'	67	-1	
111390	[2 -1 -1]	[0.002703 nan nan]	[b'*' b'' b'']	728	nan	b'Contig999'	66	-1	
111391	[2 -1 -1]	[0.002703 nan nan]	[b'*' b'' b'']	728	nan	b'Contig999'	66	-1	

Plot function

In [9]:
```
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 [10]:
```
numalt = rawsnps['numalt']
np.max(numalt)
```

Out[10]:
```
3
```

In [11]:
```
count_numalt = np.bincount(numalt)
count_numalt
```

Out[11]:
```
array([     0, 167993,  10219,    264])
```

In [12]:
```
n_multiallelic = np.sum(count_numalt[2:])
n_multiallelic
```

Out[12]:
```
10483
```

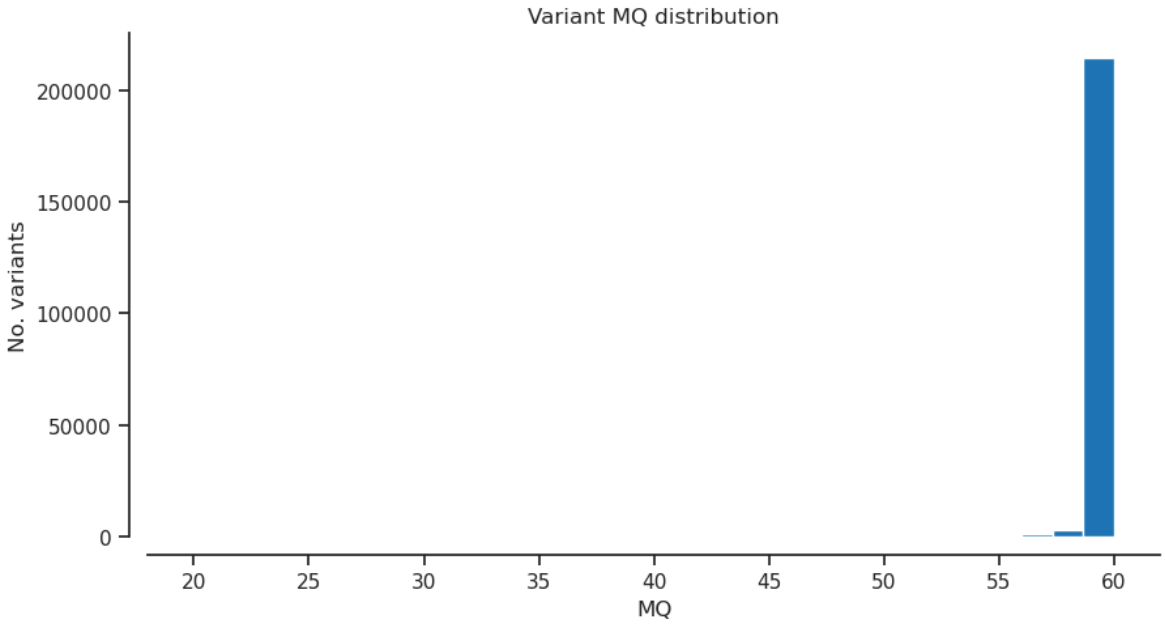
In [13]:
```
filter_expression = '(numalt == 1)'
biallelic_np = rawsnps.query(filter_expression)[:]
biallelic_np
```

Out[13]: <VariantTable shape=(167993,) 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')])>

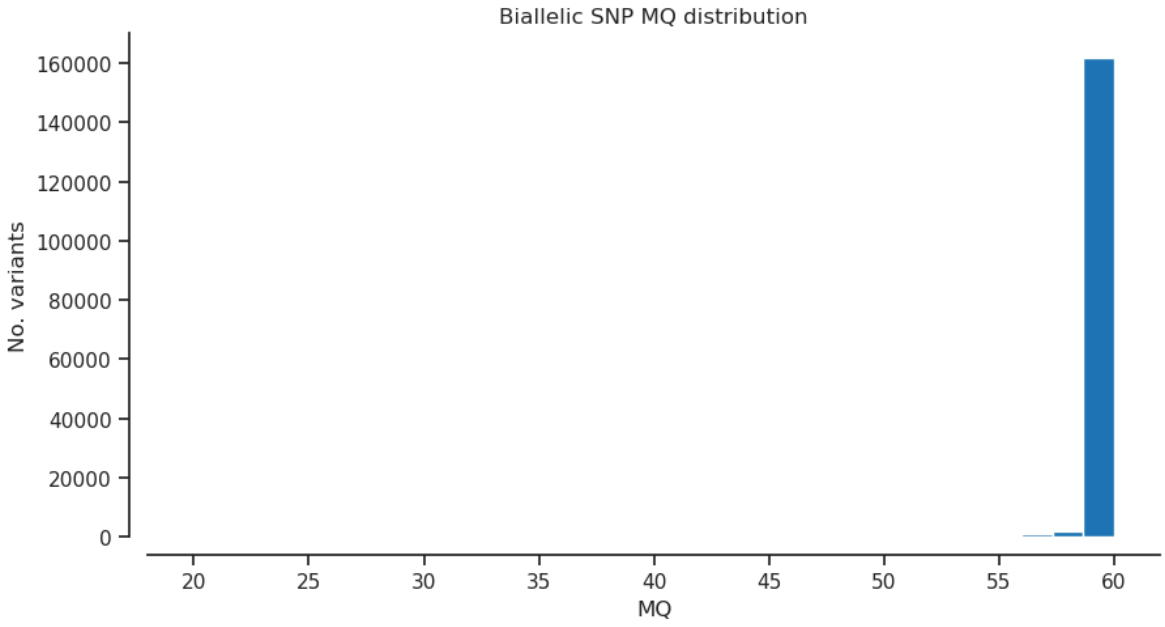
	AC	AF	ALT	AN	BaseQRankSum	CHROM	DP	END	E
0	[1 -1 -1]	[0.001351 nan nan]	[b'C' b'' b'']	728	0.962	b'Contig0'	16693	-1	
1	[478 -1 -1]	[0.654 nan nan]	[b'G' b'' b'']	728	-0.043	b'Contig0'	17345	-1	
2	[23 -1 -1]	[0.031 nan nan]	[b'T' b'' b'']	728	0.66	b'Contig0'	17175	-1	
...									
167990	[2 -1 -1]	[0.002703 nan nan]	[b'A' b'' b'']	728	nan	b'Contig999'	65	-1	
167991	[4 -1 -1]	[0.005405 nan nan]	[b'G' b'' b'']	728	nan	b'Contig999'	64	-1	
167992	[2 -1 -1]	[0.002703 nan nan]	[b'T' b'' b'']	728	nan	b'Contig999'	63	-1	

MQ - RMS mapping quality

In [14]:
```
plot_hist('MQ','var') # RMS mapping quality
```

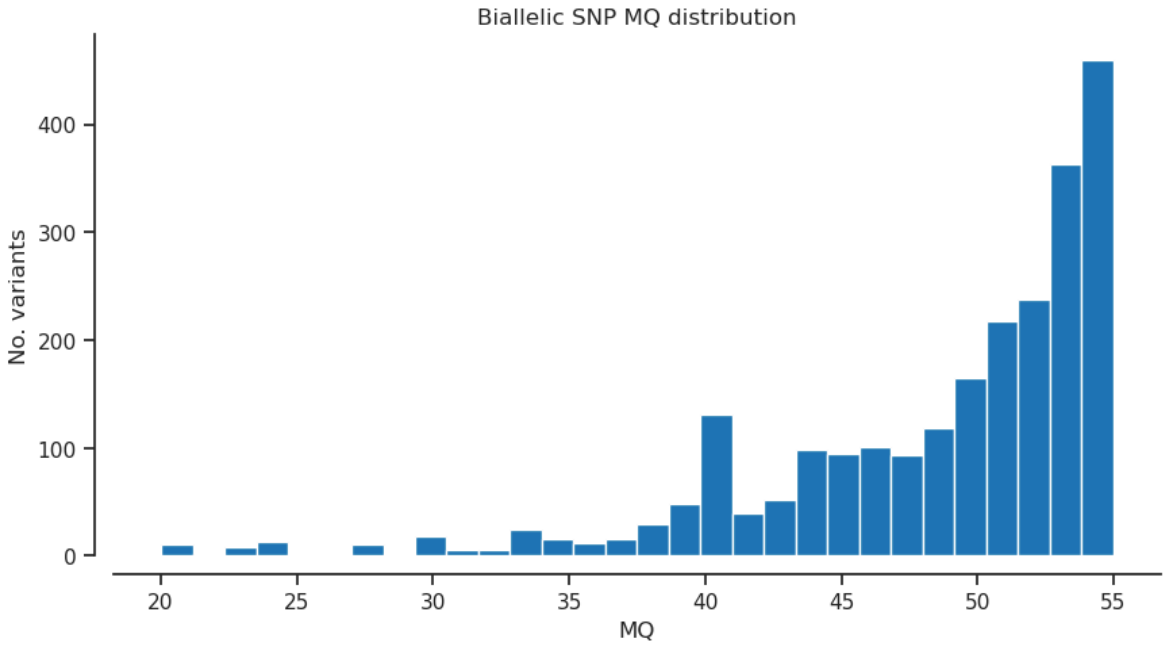


In [15]:
```
plot_hist('MQ','biallelic') # RMS mapping quality
```



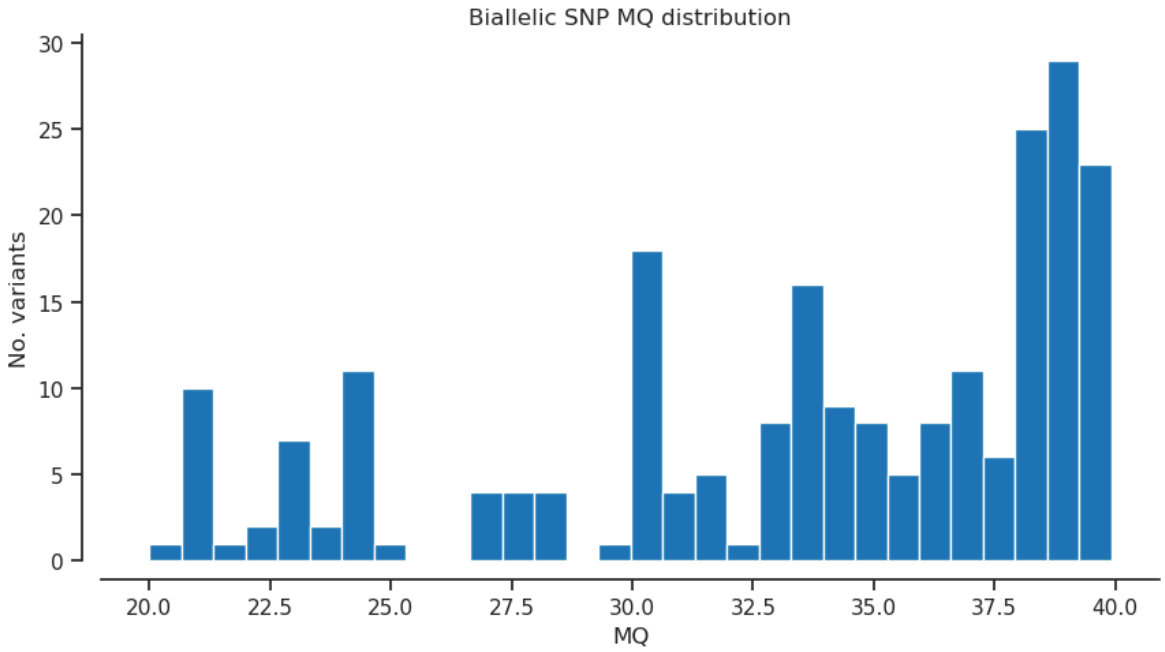
In [16]:
```
filter_expression = '(MQ < 55)'
bi_selection = biallelic_np.query(filter_expression)[:]
#np.count_nonzero(var_selection)
```

In [17]:
```
plot_hist('MQ')
```

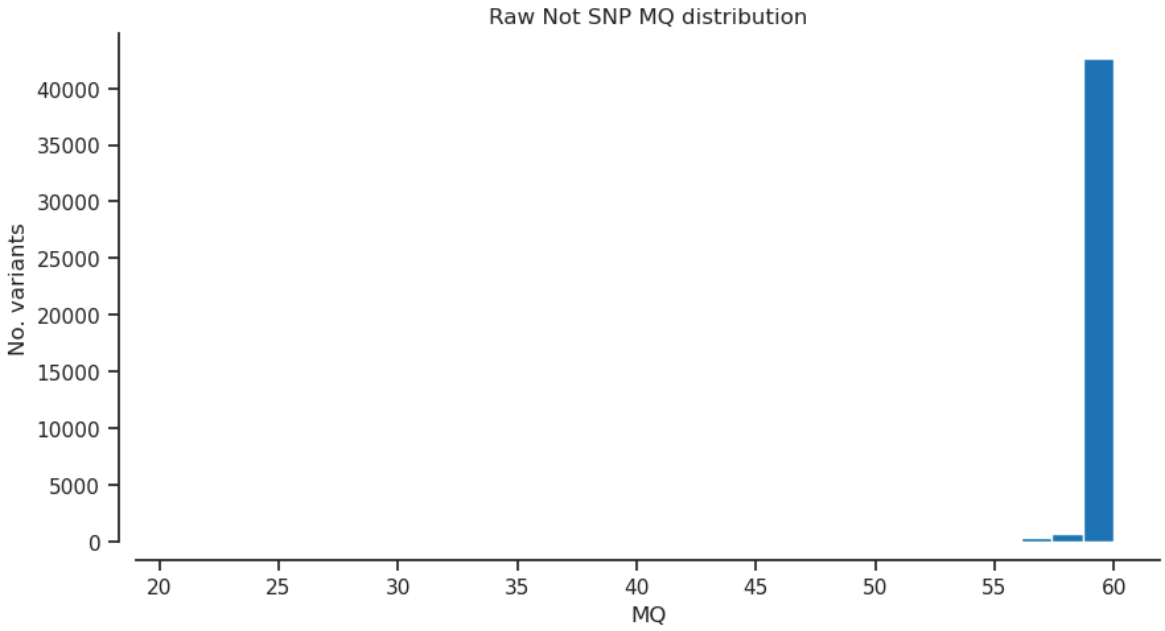


In [18]:
```
filter_expression = '(MQ < 40)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

In [19]:
```
plot_hist('MQ')
```

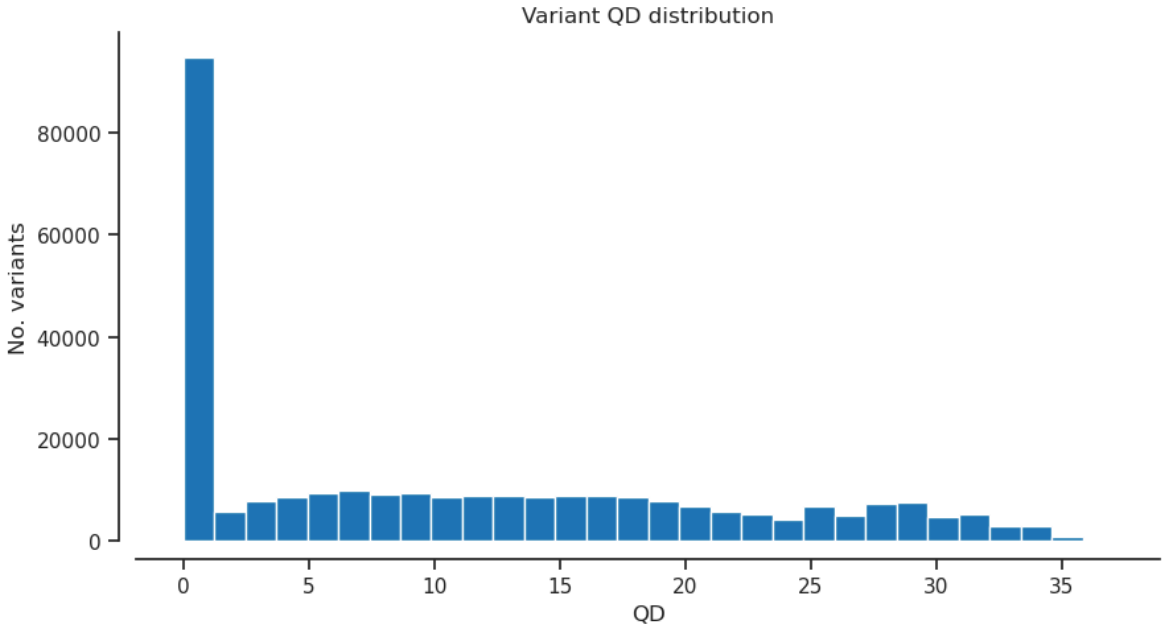


In [20]:
```
plot_hist('MQ','notsnp')
```

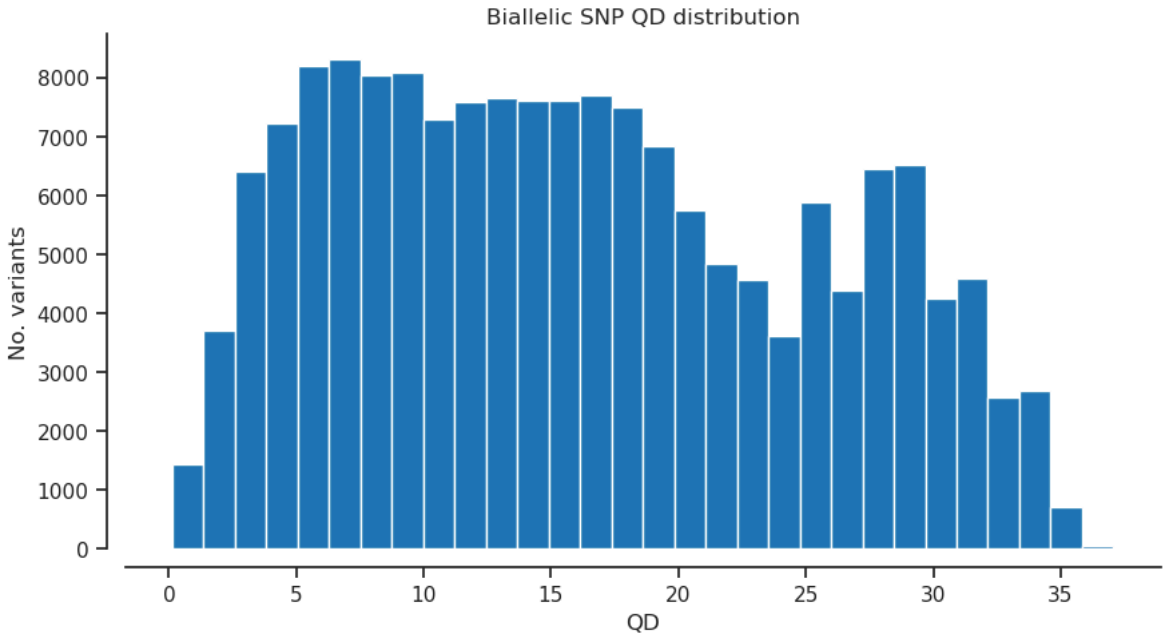


QD - Variant Confidence/Quality by Depth

In [21]:
```
plot_hist('QD','var') # Variant Confidence/Quality by Depth
```

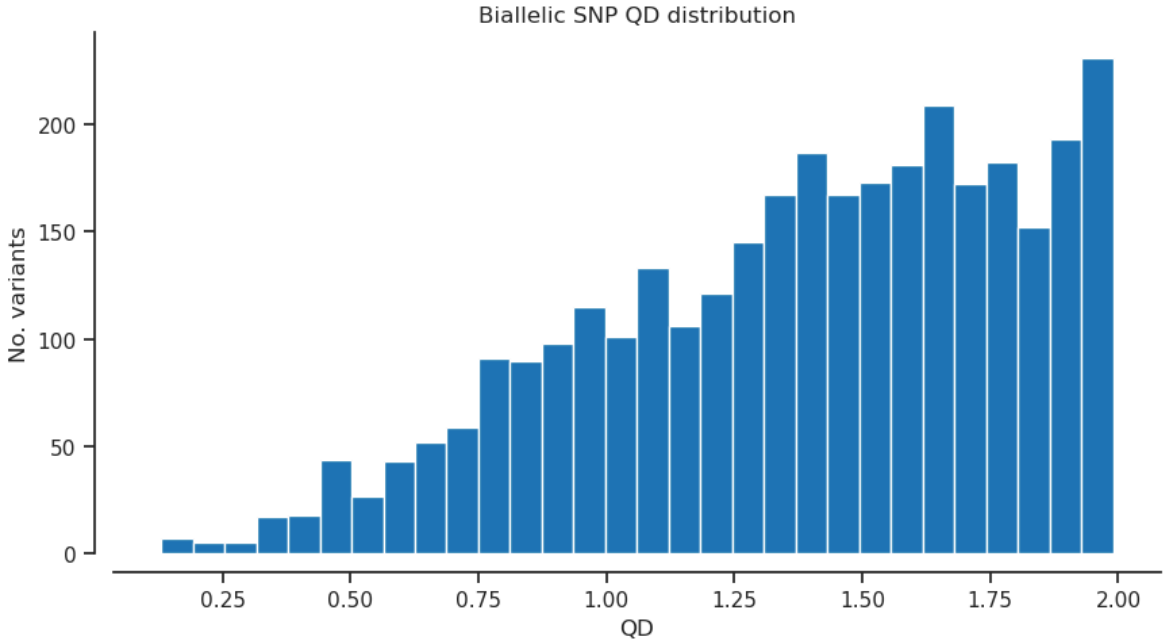


In [22]:
```
plot_hist('QD','biallelic') # Variant Confidence/Quality by Depth
```

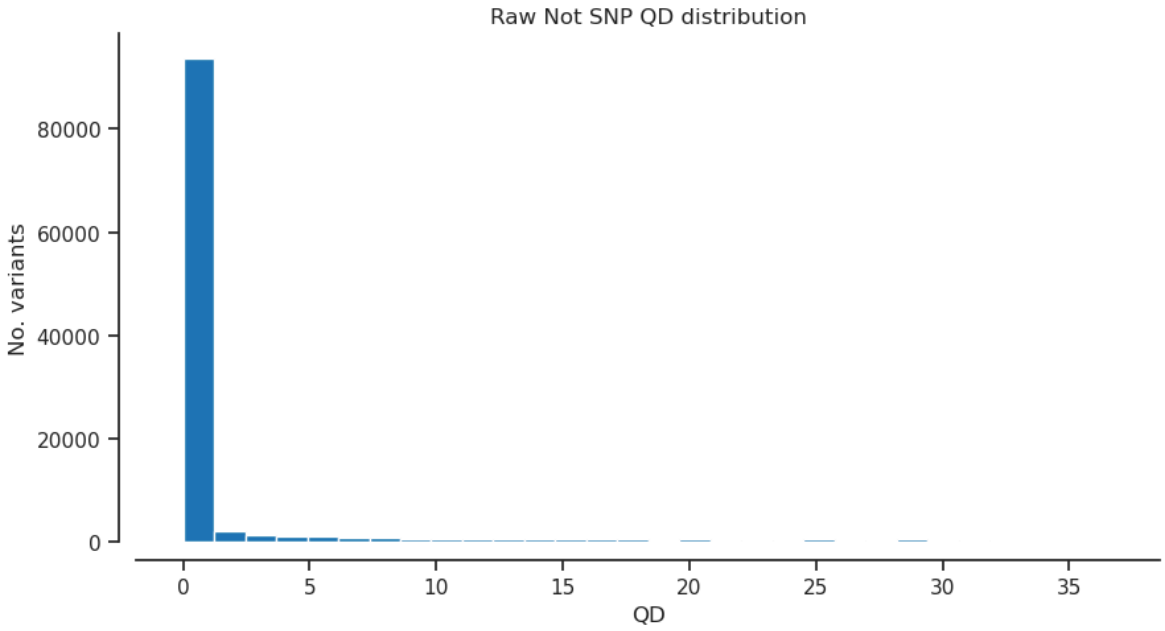


In [23]:
```
filter_expression = '(QD < 2)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

In [24]:
```
plot_hist('QD')
```

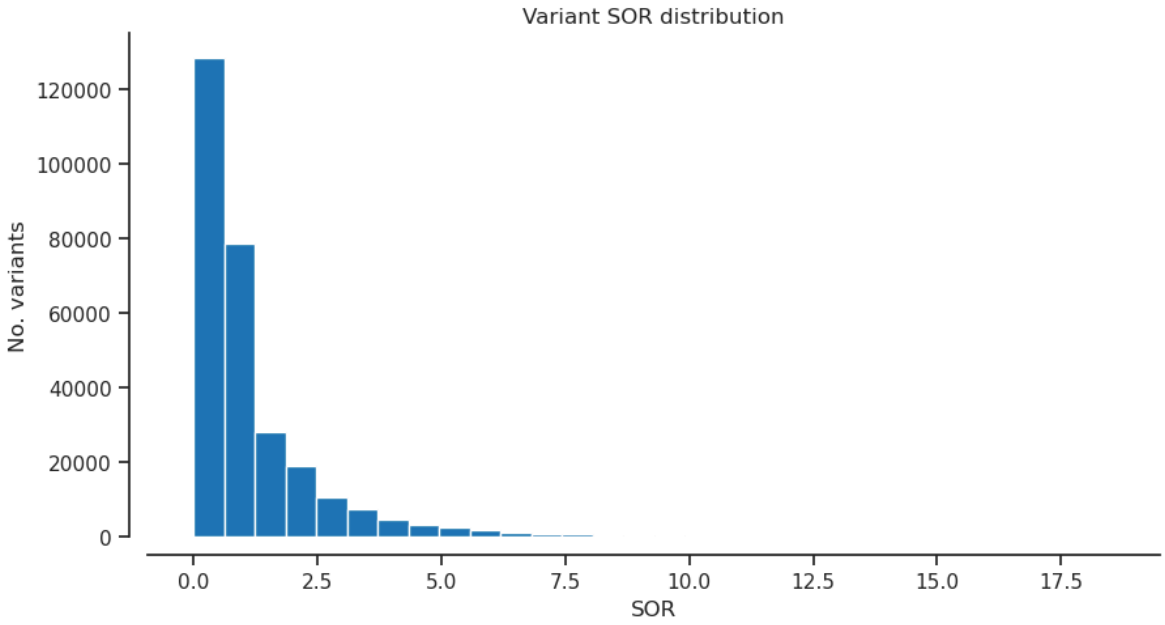


In [25]:
```
plot_hist('QD','notsnp') # Variant Confidence/Quality by Depth
```

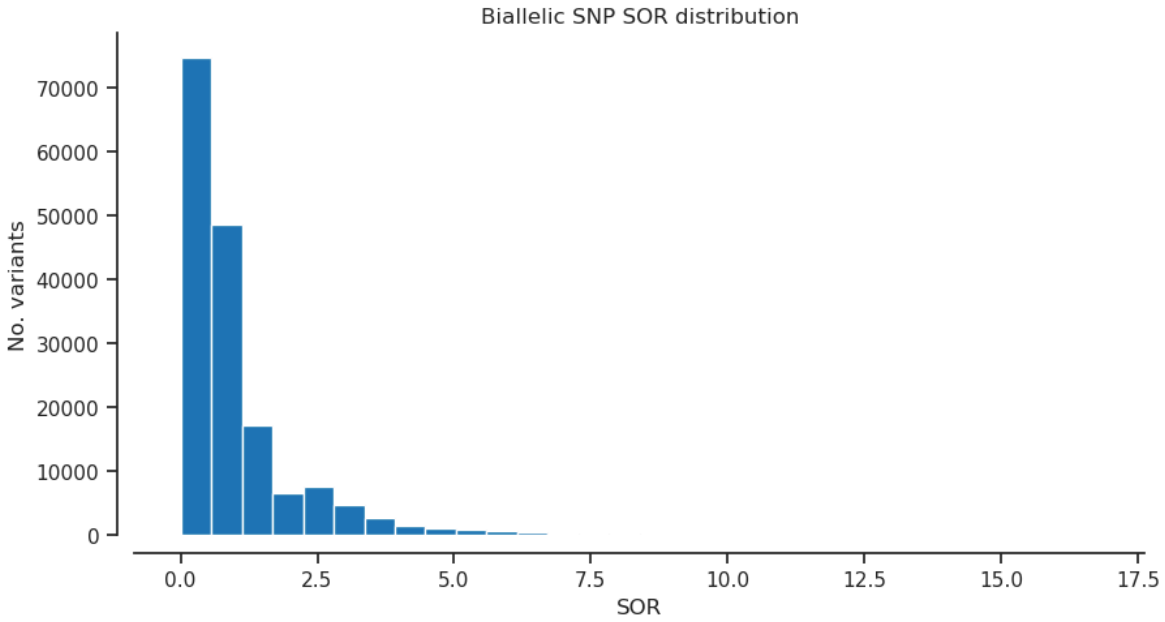


SOR - Symmetric Odds Ratio of 2x2 contingency table to detect strand bias

In [26]:
```
plot_hist('SOR', 'var') # Symmetric Odds Ratio of 2x2 contingency table t
```

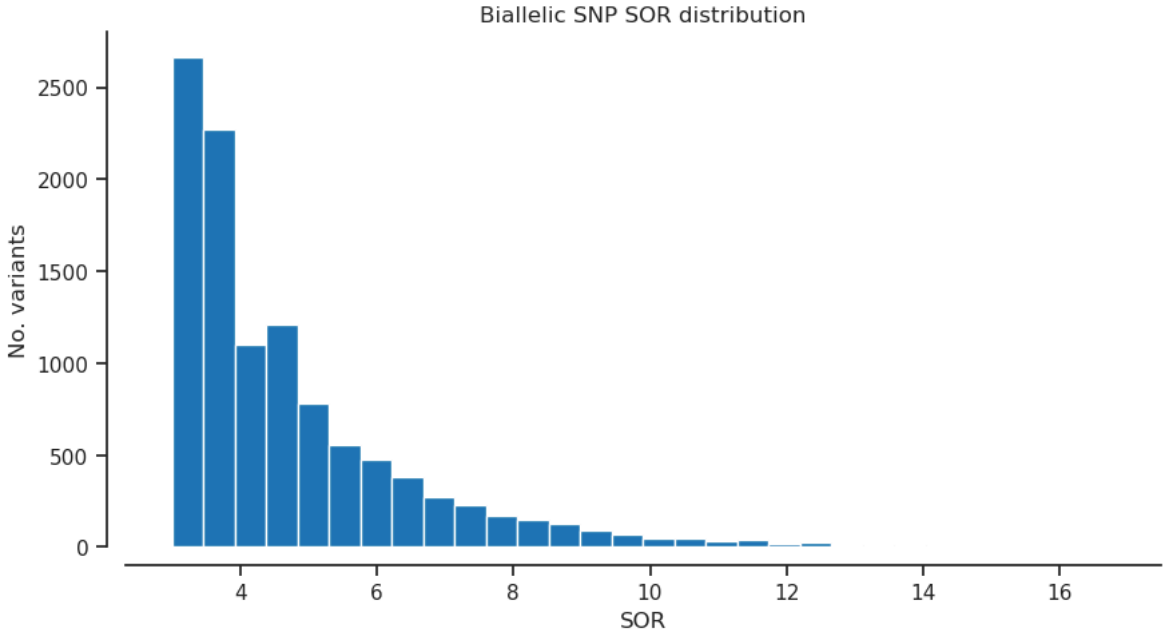


In [27]:
```
plot_hist('SOR','biallelic') # Symmetric Odds Ratio of 2x2 contingency ta
```

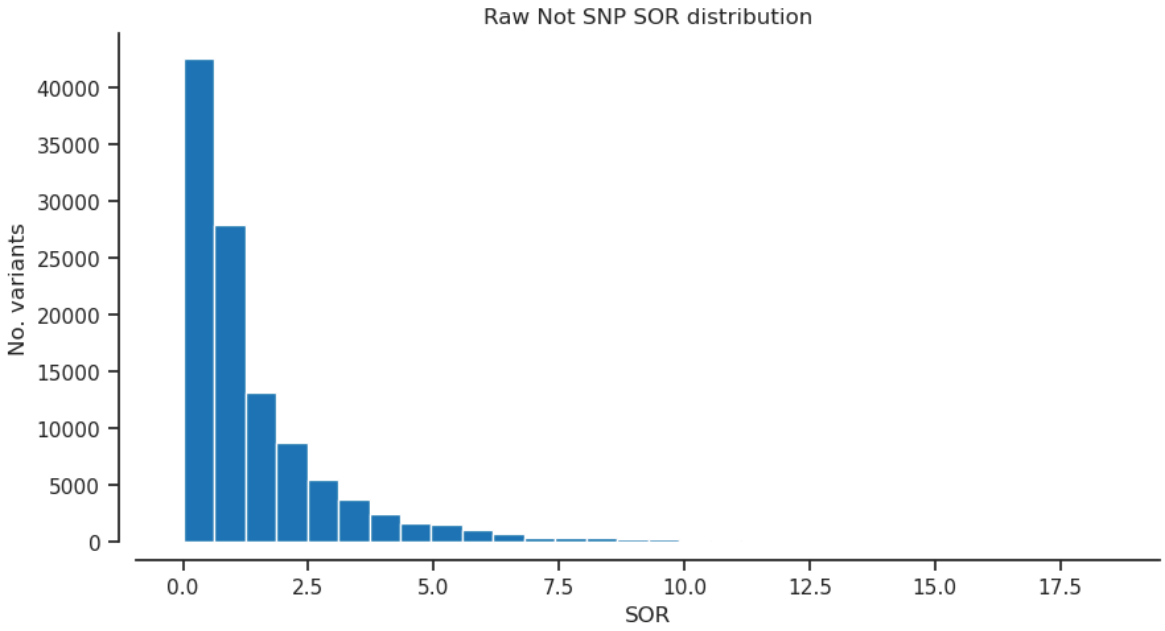


In [28]:
```
filter_expression = '(SOR > 3)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

In [29]:
```
plot_hist('SOR') # Symmetric Odds Ratio of 2x2 contingency table to detec
```

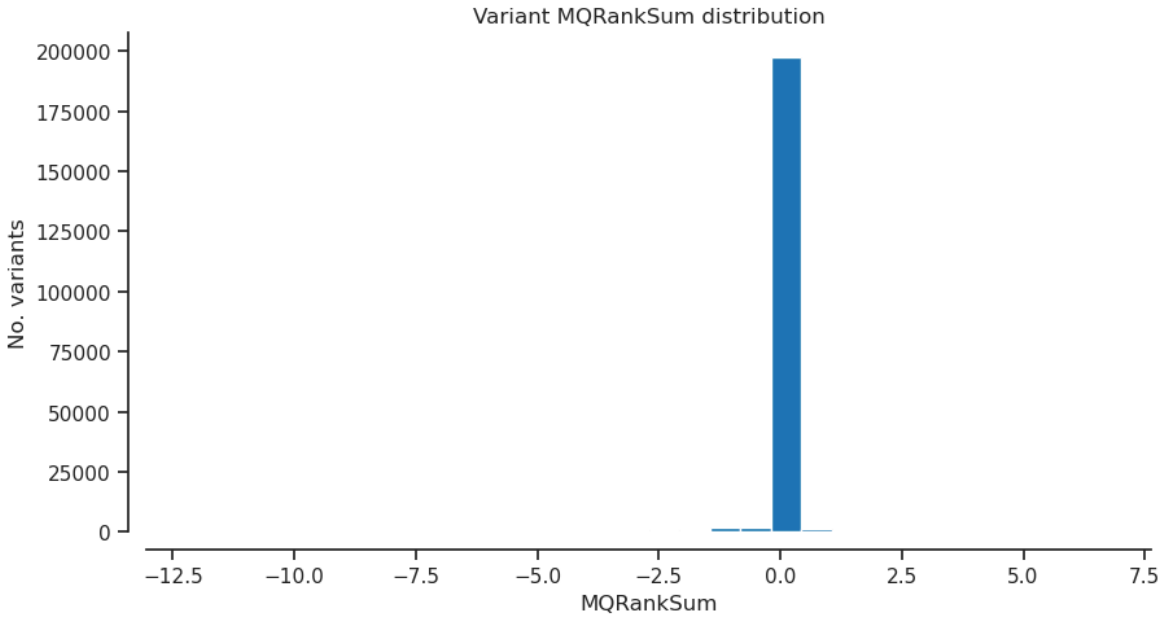


In [30]:
```
plot_hist('SOR','notsnp') # Symmetric Odds Ratio of 2x2 contingency table
```

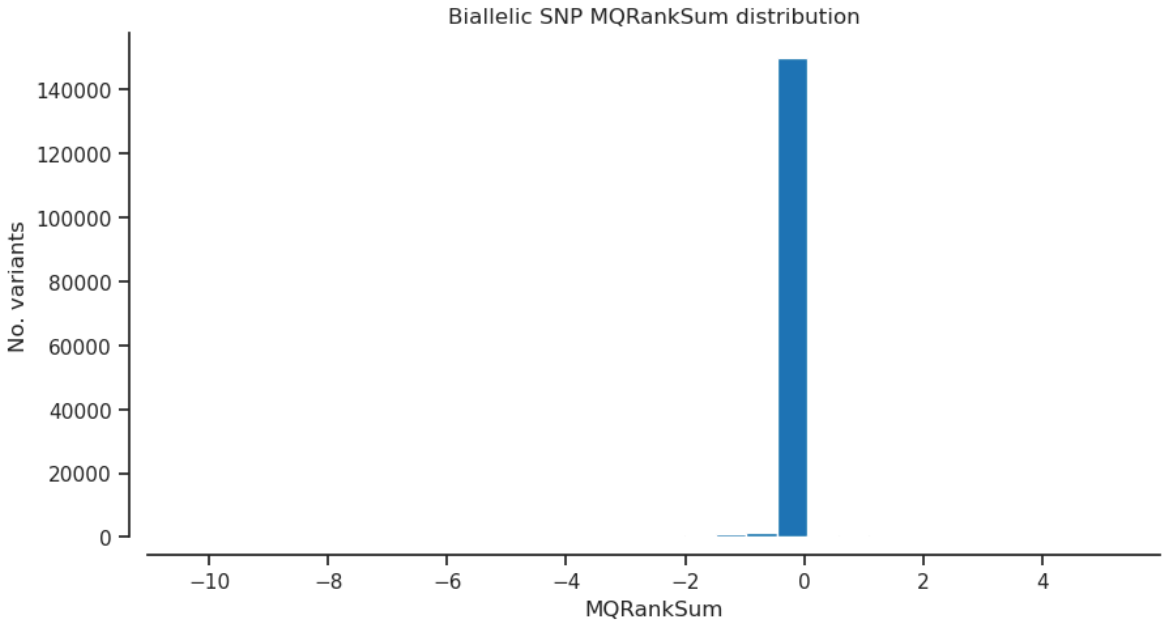


MQRankSum - Z-score From Wilcoxon rank sum test of Alt vs. Ref read mapping qualities

In [31]:
```
plot_hist('MQRankSum','var') # Z-score From Wilcoxon rank sum test of Alt
```

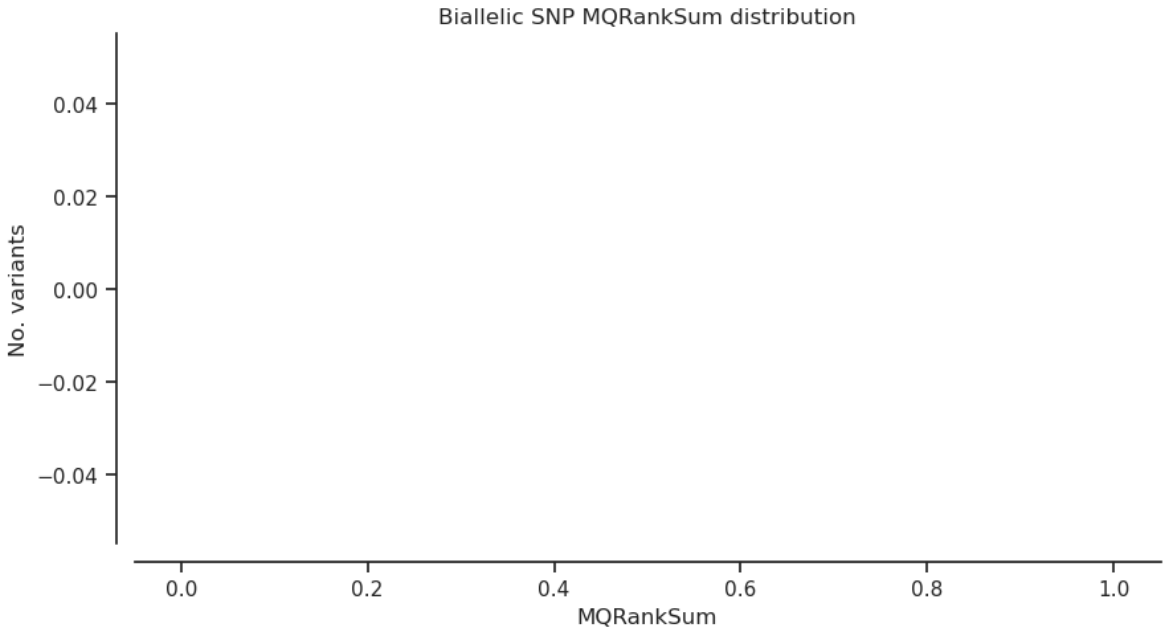


In [32]:
```
plot_hist('MQRankSum','biallelic') # Z-score From Wilcoxon rank sum test
```

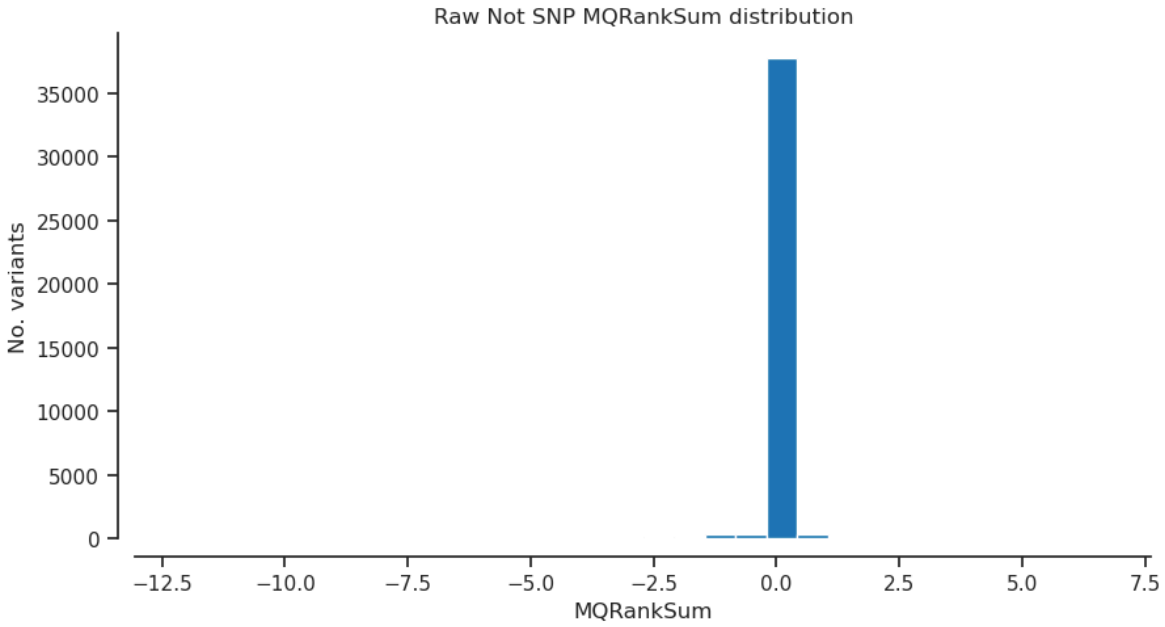


In [33]:
```
filter_expression = '(MQRankSum < -12.5)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

In [34]:
```
plot_hist('MQRankSum') # Z-score From Wilcoxon rank sum test of Alt vs. R
```

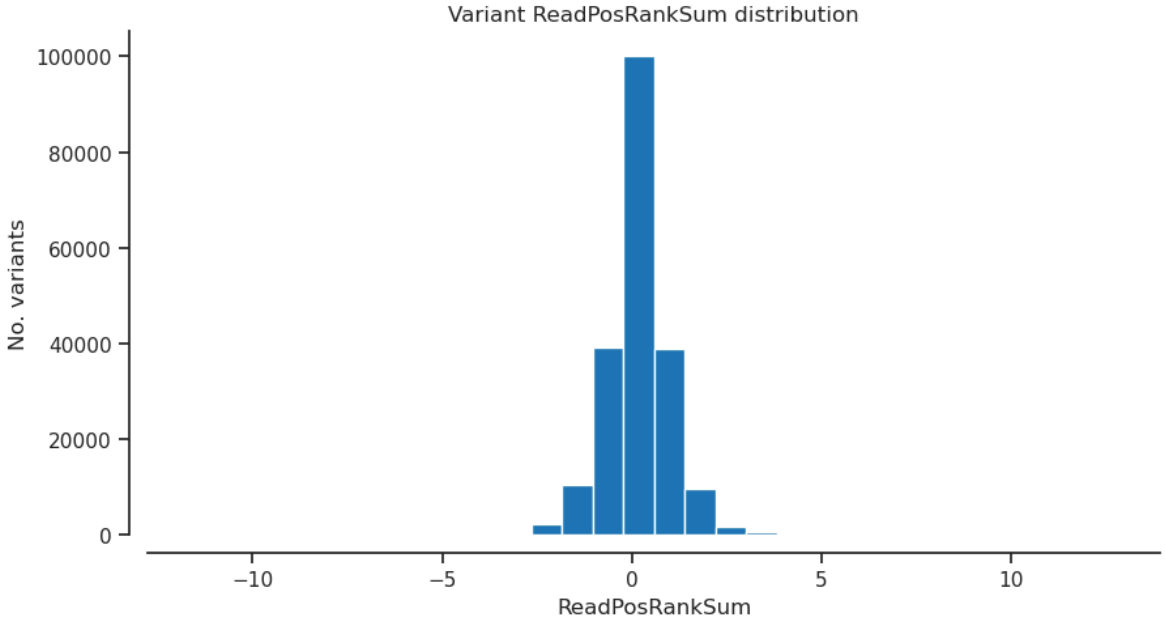


In [35]:
```
plot_hist('MQRankSum','notsnp') # Z-score From Wilcoxon rank sum test of
```

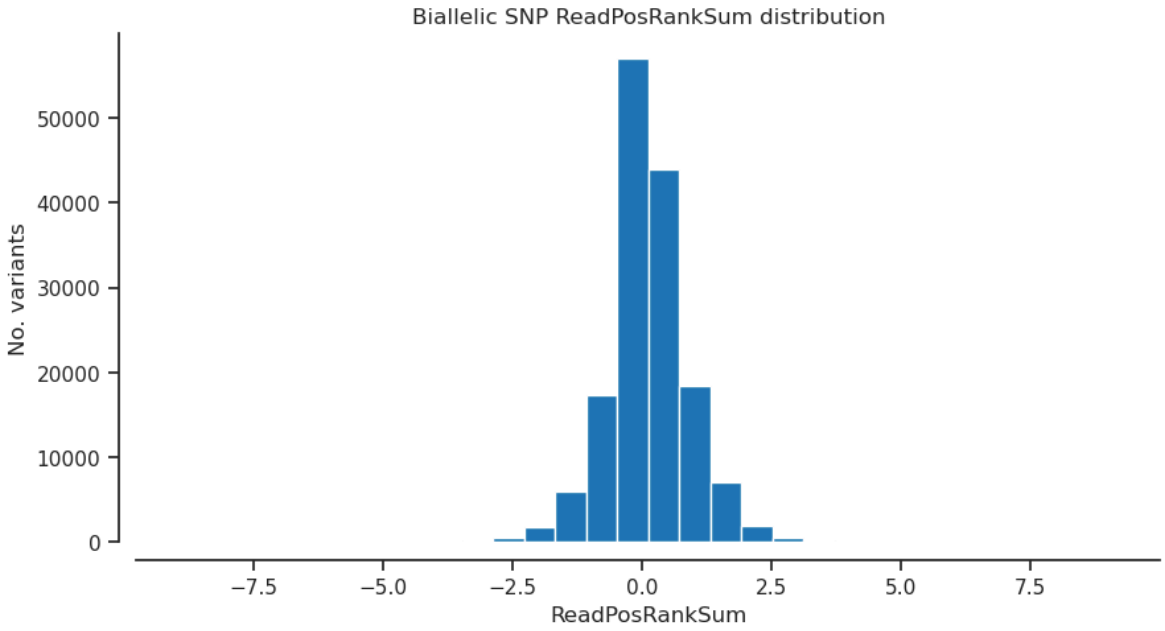


ReadPosRankSum - Z-score from Wilcoxon rank sum test of Alt vs. Ref read position bias

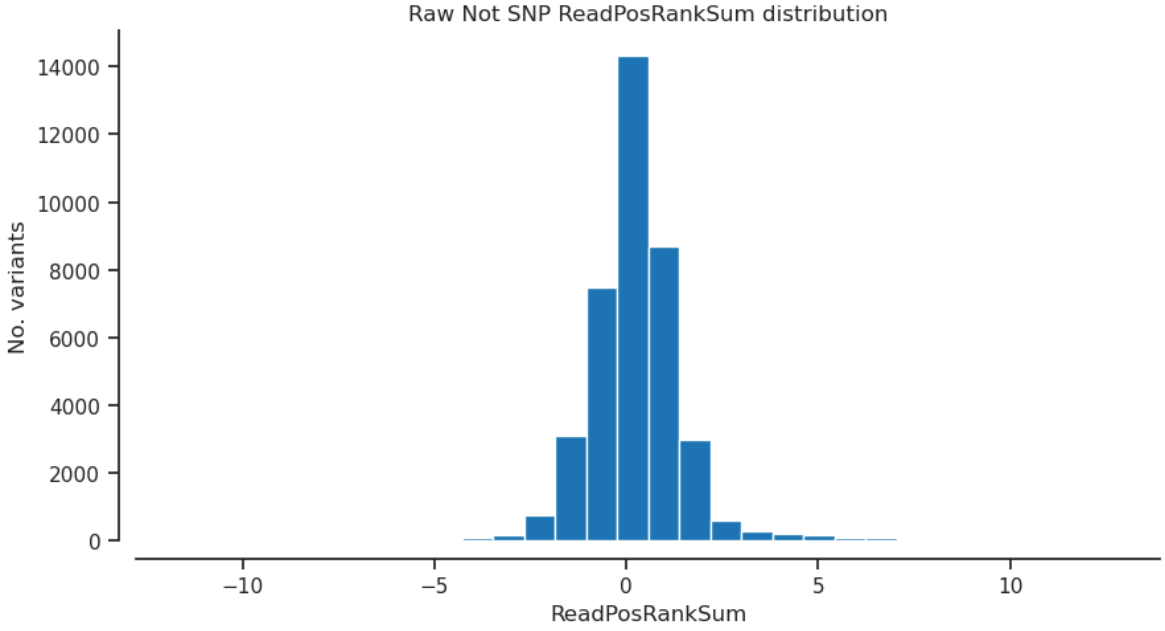
In [36]:
```
plot_hist('ReadPosRankSum','var') # Z-score from Wilcoxon rank sum test o
```



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

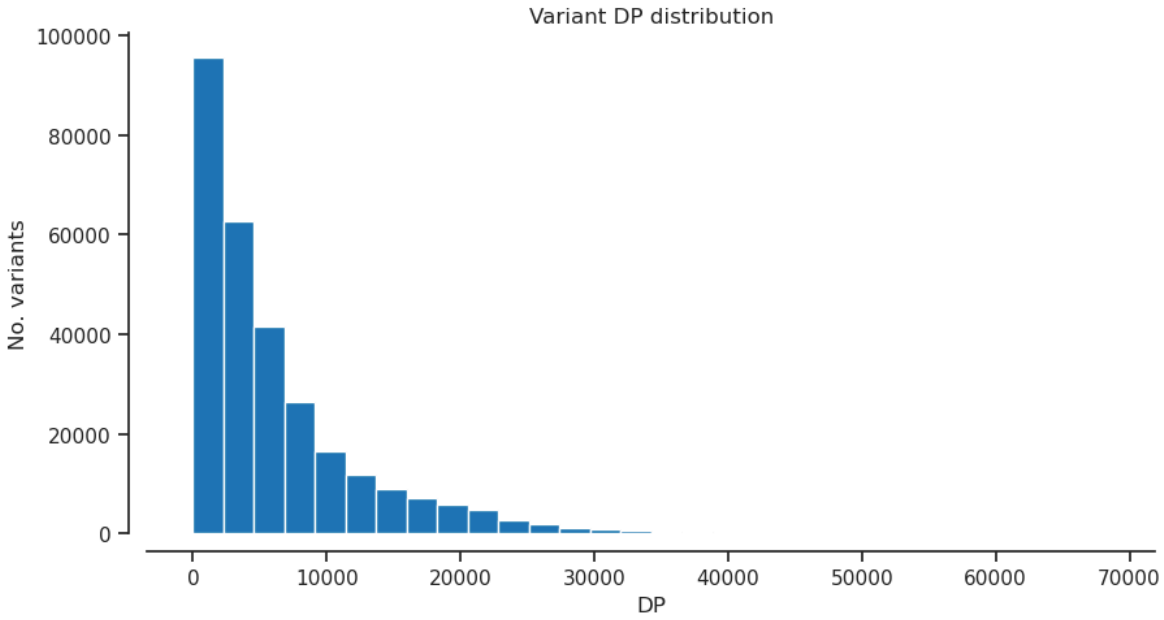


In [38]:
```
plot_hist('ReadPosRankSum','notsnp') # Z-score from Wilcoxon rank sum tes
```

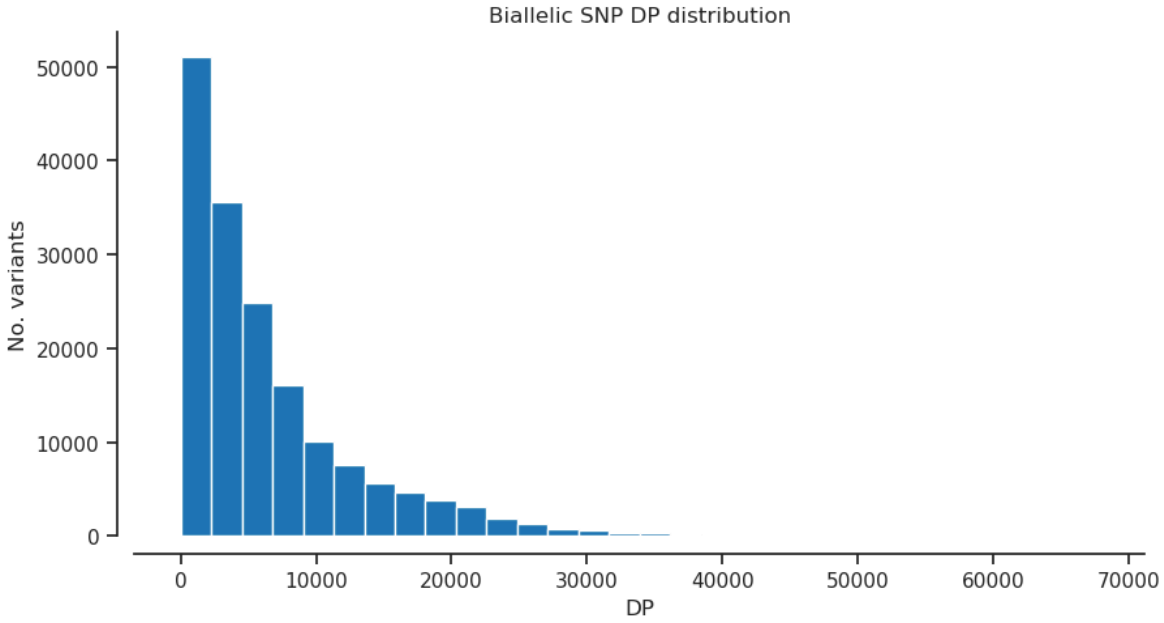


DP - Approximate read depth

In [39]:
```
plot_hist('DP','var')
```

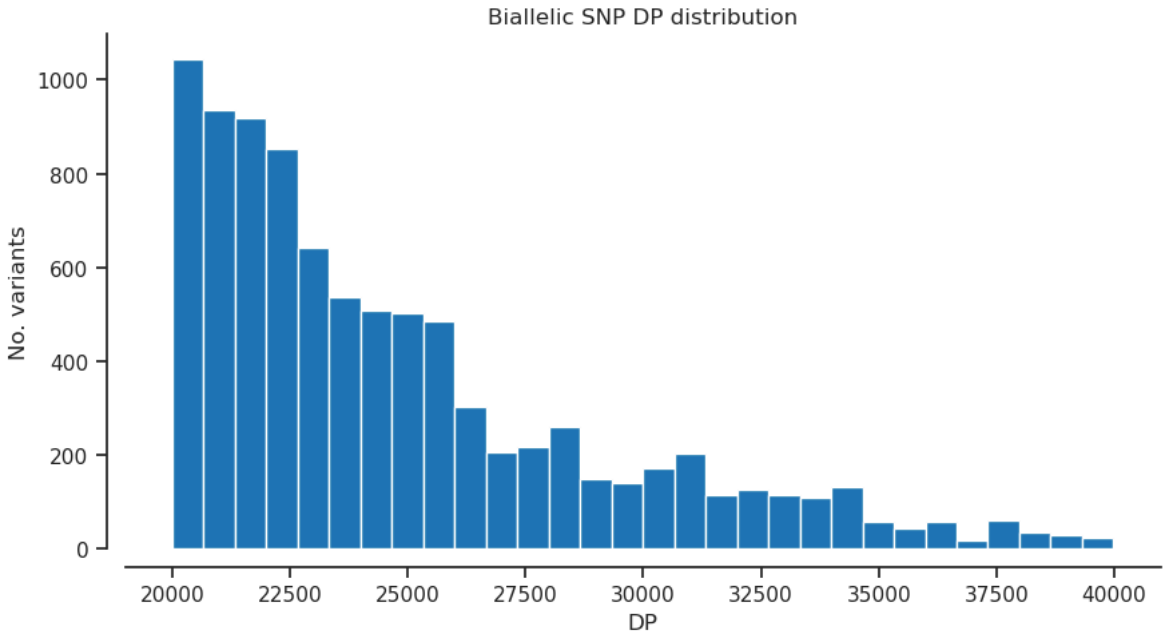


In [40]:
```
plot_hist('DP','biallelic')
```

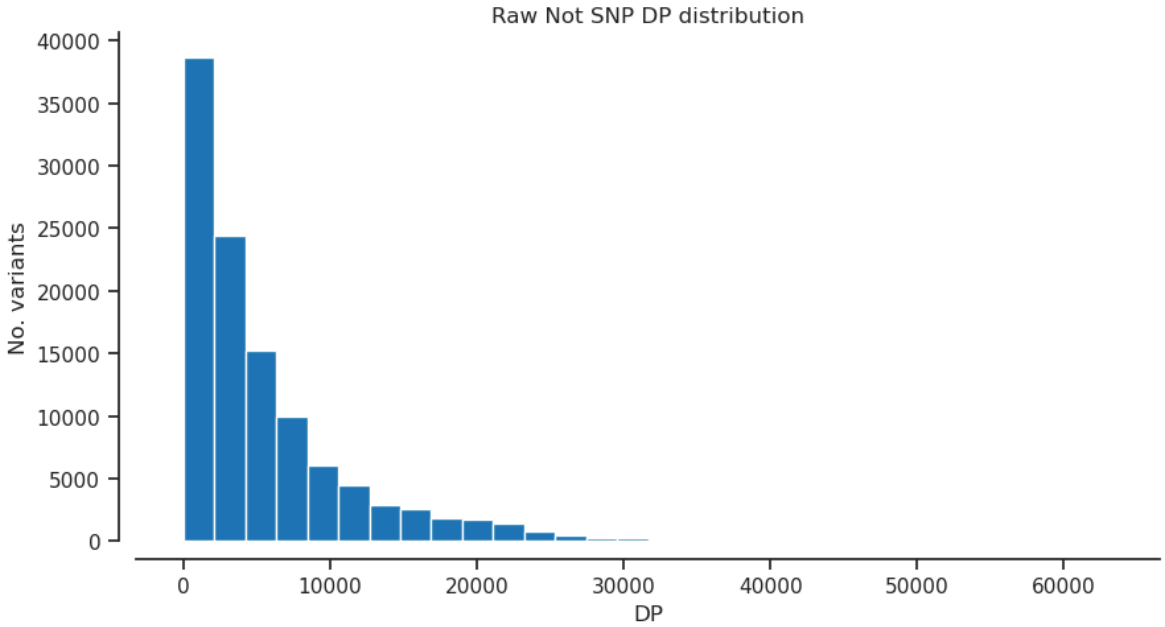


In [41]:
```
filter_expression = '(DP > 20000) & (DP < 40000)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

In [42]:
```
plot_hist('DP')
```

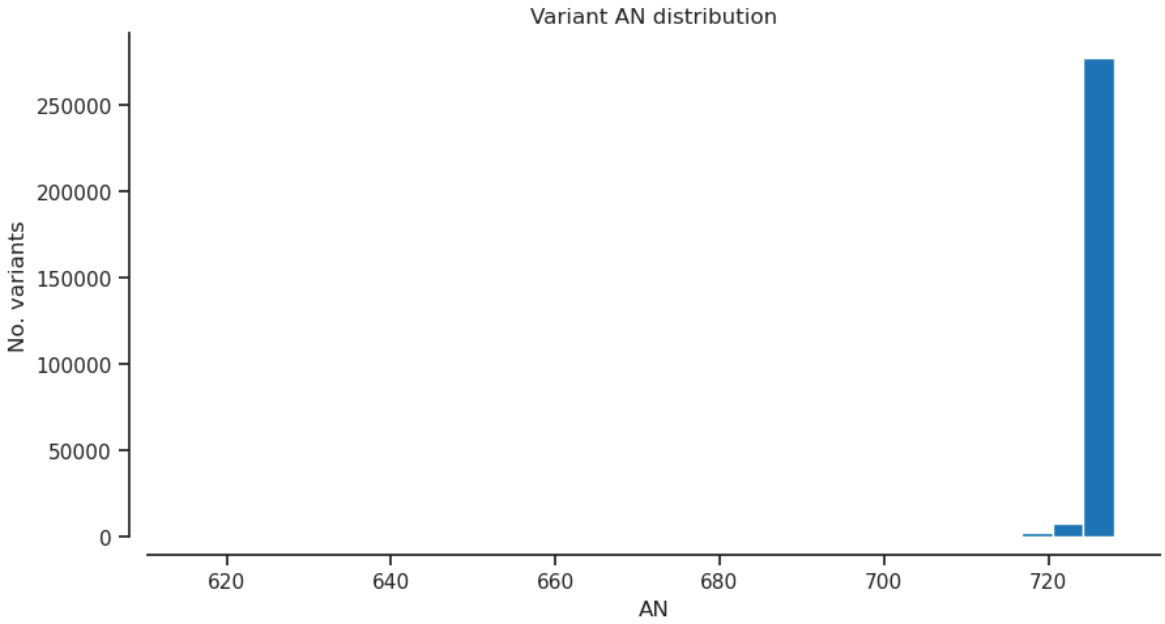


In [43]:
```
plot_hist('DP','notsnp')
```

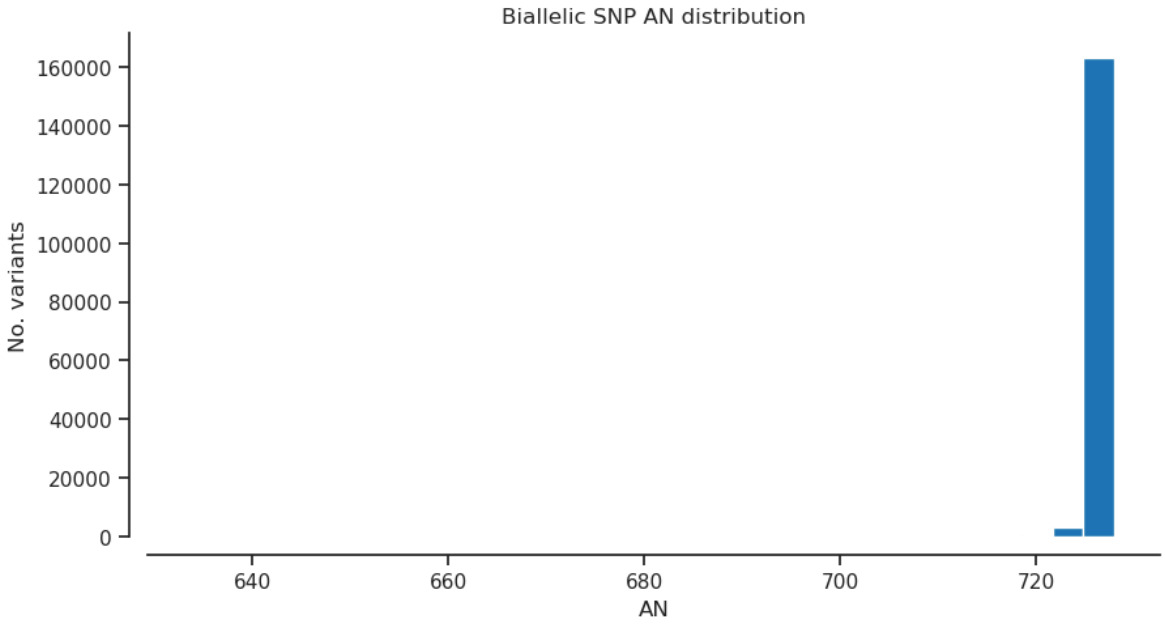


AN - Total number of alleles in called genotypes

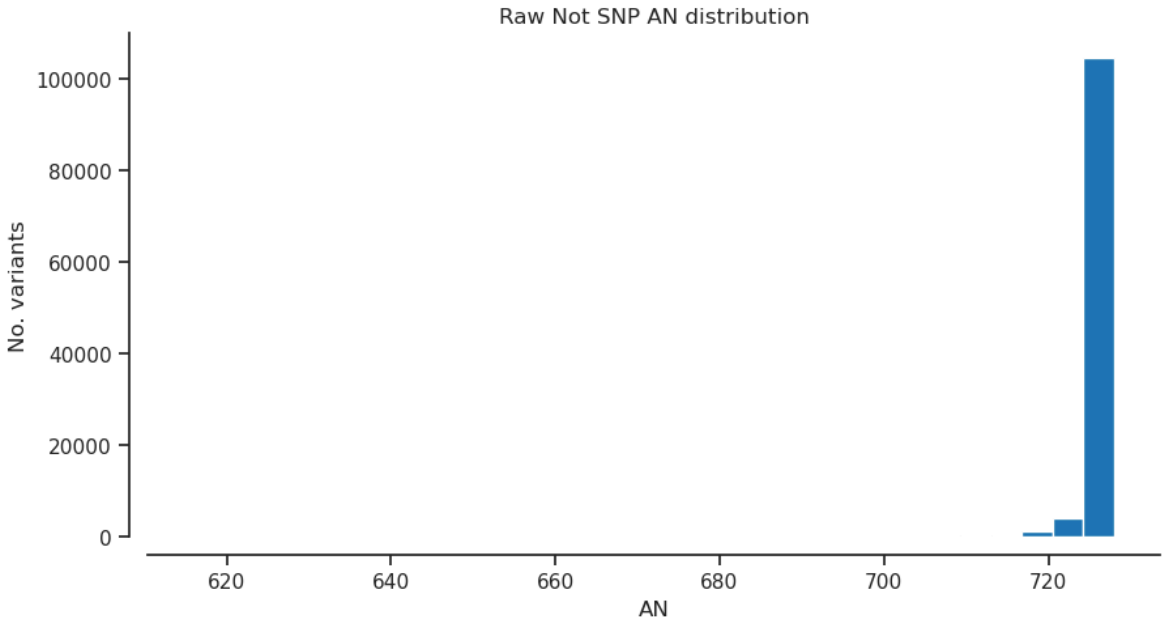
In [44]:
```
plot_hist('AN','var') # Total number of alleles in called genotypes
```



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



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



Selected filter

In [47]:
```
# 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[47]:
```
162041
```

Genotype

In [48]:
```
calldata_var = callset_var['calldata']
list(calldata_var)
```

Out[48]:
```
['AD', 'DP', 'GQ', 'GT', 'MIN_DP', 'PGT', 'PID', 'PL', 'PS', 'RGQ', 'S
B']
```

In [49]:
```
genotypes_var = allel.GenotypeChunkedArray(calldata_var['GT'])
genotypes_var
```

Out[49]: <GenotypeChunkedArray shape=(289868, 364, 2) dtype=int8 chunks=(65536, 64, 2) nbytes=201.2M cbytes=9.4M cratio=21.3 compression=gzip compression_opts=1 values=h5py._hl.dataset.Dataset>

	0	1	2	3	4	...	359	360	361	362	363
0	0/0	0/0	0/0	0/0	0/0	...	0/0	0/1	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
...						...					
289865	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
289866	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
289867	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0

In [50]:
```
# using the selected filters set above
gt_filtered_snps = genotypes_var.subset(variant_selection)
gt_filtered_snps
```

Out[50]: <GenotypeChunkedArray shape=(162041, 364, 2) dtype=int8 chunks=(1266, 364, 2) nbytes=112.5M cbytes=9.7M cratio=11.6 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3	4	...	359	360	361	362	363
0	0/0	0/0	0/0	0/0	0/0	...	0/0	0/1	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	1/1	0/1	0/1	0/1	0/1	...	0/1	0/1	0/0	0/1	0/1
...						...					
162038	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
162039	0/0	1/1	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
162040	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0

In [51]:
```
# grab the allele counts for the populations
ac = gt_filtered_snps.count_alleles()
ac
```

Out[51]: <AlleleCountsChunkedArray shape=(162041, 4) dtype=int32 chunks=(20256, 4) nbytes=2.5M cbytes=406.0K cratio=6.2 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3
0	696	23	9	0
1	727	1	0	0
2	250	478	0	0
...		...		
162038	726	2	0	0
162039	724	4	0	0
162040	726	2	0	0

In [52]:
```
ac[:]
```

Out[52]: <AlleleCountsArray shape=(162041, 4) dtype=int32>

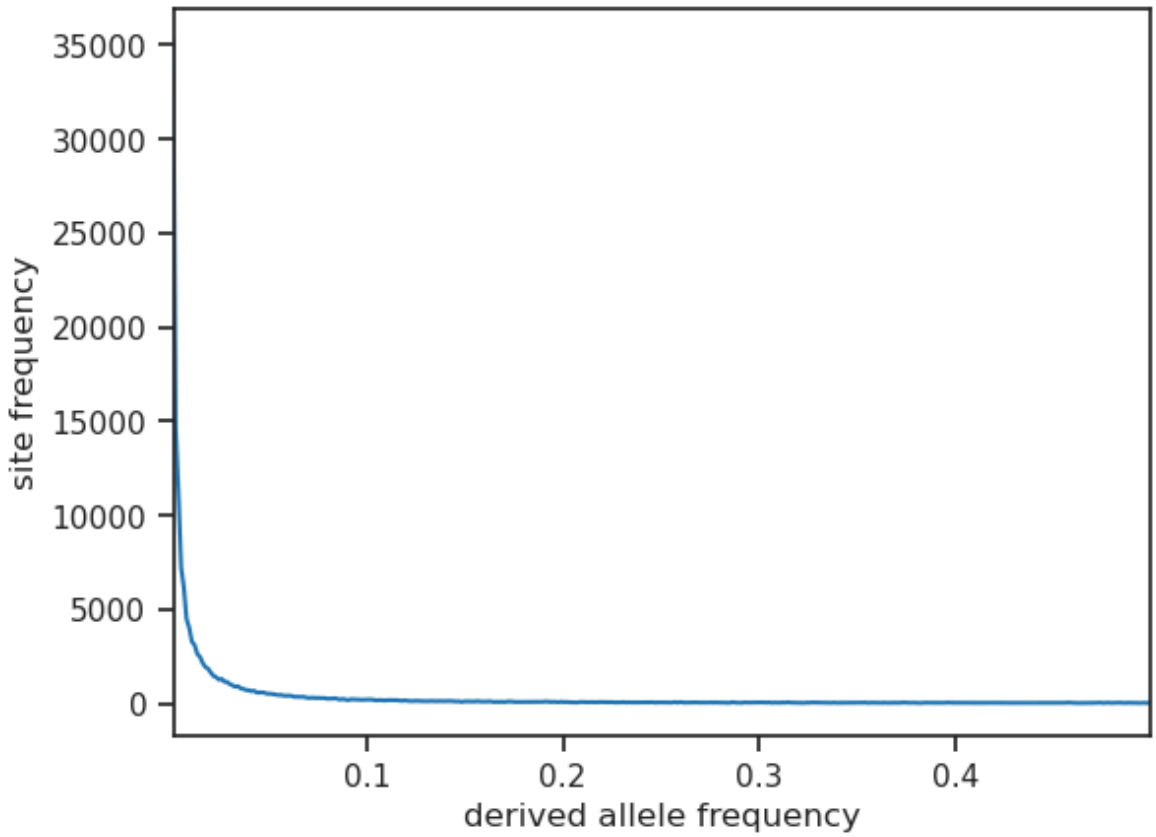
	0	1	2	3
0	696	23	9	0
1	727	1	0	0
2	250	478	0	0
...		...		
162038	726	2	0	0
162039	724	4	0	0
162040	726	2	0	0

In [53]:
```
# 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[53]:
```
array([[727,   1],
       [250, 478],
       [705,  23],
       ...,
       [726,   2],
       [724,   4],
       [726,   2]], dtype=int32)
```

In [54]:
```
# 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[54]:
```
<Axes: xlabel='derived allele frequency', ylabel='site frequency'>
```



In [55]:
```
biallelic = (ac.max_allele() == 1)
###This is the filter expression for biallelic sites
biallelic
```

Out[55]:
```
<ChunkedArrayWrapper shape=(162041,) dtype=bool chunks=(162041,)
   nbytes=158.2K cbytes=36.3K cratio=4.4
   compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shu
ffle': 1, 'blocksize': 0}
   values=zarr.core.Array>
```

In [56]:
```
# select only the biallelic variants
gt_biallelic = gt_filtered_snps.compress(biallelic)
gt_biallelic
```

Out[56]:
<GenotypeChunkedArray shape=(151255, 364, 2) dtype=int8 chunks=(1182, 364, 2) nbytes=105.0M cbytes=8.6M cratio=12.2 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3	4	...	359	360	361	362	363
0	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
1	1/1	0/1	0/1	0/1	0/1	...	0/1	0/1	0/0	0/1	0/1
2	0/0	0/0	0/0	0/0	0/0	...	0/0	0/1	0/0	0/0	0/0
...						...					
151252	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
151253	0/0	1/1	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
151254	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0

In [57]:
```
n_variants = len(gt_biallelic)
n_variants
```

Out[57]: 151255

In [58]:
```
pc_missing = gt_biallelic.count_missing(axis=0)[:] * 100 / n_variants
pc_het = gt_biallelic.count_het(axis=0)[:] * 100 / n_variants
```

Samples

In [59]:
```
samples_var = callset_var['samples']
samples_var = list(samples_var)
samples_var
```

Out[59]:
```
[b'ESP00102-001',
 b'ESP00102-002',
 b'ESP00102-003',
 b'ESP00102-004',
 b'ESP00102-005',
 b'ESP00102-006',
 b'ESP00102-007',
 b'ESP00102-008',
 b'ESP00102-009',
 b'ESP00102-010',
 b'ESP00102-011',
 b'ESP00102-012',
 b'ESP00102-013',
 b'ESP00102-014',
 b'ESP00102-015',
 b'ESP00102-016',
 b'ESP00102-017',
 b'ESP00102-018',
 b'ESP00102-019',
 b'ESP00102-020',
 b'ESP00102-021',
 b'ESP00102-022',
 b'ESP00102-023',
 b'ESP00102-024',
 b'ESP00102-025',
 b'ESP00199-001',
 b'ESP00199-002',
 b'ESP00199-003',
 b'ESP00199-004',
 b'ESP00199-005',
 b'ESP00199-006',
 b'ESP00199-007',
 b'ESP00199-008',
 b'ESP00199-009',
 b'ESP00199-010',
 b'ESP00199-011',
 b'ESP00199-012',
 b'ESP00199-013',
 b'ESP00199-014',
 b'ESP00199-015',
 b'ESP00199-016',
 b'ESP00199-017',
 b'ESP00199-018',
 b'ESP00199-019',
 b'ESP00199-020',
 b'ESP00199-021',
 b'ESP00199-022',
 b'ESP00199-023',
 b'ESP00199-024',
 b'ESP00199-025',
 b'ESP00337-001',
 b'ESP00337-002',
 b'ESP00337-003',
 b'ESP00337-004',
 b'ESP00337-005',
 b'ESP00337-006',
 b'ESP00337-007',
 b'ESP00337-008',
 b'ESP00337-009',
 b'ESP00337-010',
```

```
b'ESP00337-011',
b'ESP00337-012',
b'ESP00337-013',
b'ESP00337-014',
b'ESP00337-015',
b'ESP00337-016',
b'ESP00337-017',
b'ESP00337-018',
b'ESP00337-019',
b'ESP00337-020',
b'ESP00337-021',
b'ESP00337-022',
b'ESP00337-023',
b'ESP00337-024',
b'ESP00337-025',
b'FIN00015-001',
b'FIN00015-002',
b'FIN00015-003',
b'FIN00015-004',
b'FIN00015-005',
b'FIN00015-006',
b'FIN00015-007',
b'FIN00015-008',
b'FIN00015-009',
b'FIN00015-010',
b'FIN00015-011',
b'FIN00015-012',
b'FIN00015-013',
b'FIN00015-014',
b'FIN00015-015',
b'FIN00015-016',
b'FIN00015-017',
b'FIN00015-018',
b'FIN00015-019',
b'FIN00015-020',
b'FIN00015-021',
b'FIN00015-022',
b'FIN00015-023',
b'FIN00015-024',
b'FIN00015-025',
b'FIN00020-001',
b'FIN00020-002',
b'FIN00020-003',
b'FIN00020-004',
b'FIN00020-005',
b'FIN00020-006',
b'FIN00020-007',
b'FIN00020-008',
b'FIN00020-009',
b'FIN00020-010',
b'FIN00020-011',
b'FIN00020-012',
b'FIN00020-013',
b'FIN00020-014',
b'FIN00020-015',
b'FIN00020-016',
b'FIN00020-017',
b'FIN00020-018',
b'FIN00020-019',
b'FIN00020-020',
```

```
b'FIN00020-021',
b'FIN00020-022',
b'FIN00020-023',
b'FIN00020-024',
b'FIN00020-025',
b'FIN00044-001',
b'FIN00044-002',
b'FIN00044-003',
b'FIN00044-004',
b'FIN00044-005',
b'FIN00044-006',
b'FIN00044-007',
b'FIN00044-009',
b'FIN00044-010',
b'FIN00044-011',
b'FIN00044-012',
b'FIN00044-013',
b'FIN00044-014',
b'FIN00044-015',
b'FIN00044-016',
b'FIN00044-017',
b'FIN00044-018',
b'FIN00044-019',
b'FIN00044-020',
b'FIN00044-021',
b'FIN00044-022',
b'FIN00044-023',
b'FIN00044-024',
b'FIN00044-025',
b'FIN00046-001',
b'FIN00046-002',
b'FIN00046-003',
b'FIN00046-004',
b'FIN00046-005',
b'FIN00046-006',
b'FIN00046-007',
b'FIN00046-008',
b'FIN00046-009',
b'FIN00046-010',
b'FIN00046-011',
b'FIN00046-012',
b'FIN00046-013',
b'FIN00046-014',
b'FIN00046-015',
b'FIN00046-016',
b'FIN00046-017',
b'FIN00046-018',
b'FIN00046-019',
b'FIN00046-020',
b'FIN00046-021',
b'FIN00046-022',
b'FIN00046-023',
b'FIN00046-024',
b'FIN00046-025',
b'GBR00013-001',
b'GBR00013-002',
b'GBR00013-003',
b'GBR00013-004',
b'GBR00013-005',
b'GBR00013-006',
```

```
b'GBR00013-007',
b'GBR00013-008',
b'GBR00013-009',
b'GBR00013-010',
b'GBR00013-013',
b'GBR00013-014',
b'GBR00013-015',
b'GBR00013-016',
b'GBR00013-017',
b'GBR00013-018',
b'GBR00013-019',
b'GBR00013-020',
b'GBR00013-021',
b'GBR00013-022',
b'GBR00013-023',
b'GBR00013-024',
b'GBR00013-025',
b'GBR00015-101',
b'GBR00015-102',
b'GBR00015-103',
b'GBR00015-104',
b'GBR00015-105',
b'GBR00015-106',
b'GBR00015-107',
b'GBR00015-108',
b'GBR00015-109',
b'GBR00015-110',
b'GBR00015-111',
b'GBR00015-112',
b'GBR00015-113',
b'GBR00015-114',
b'GBR00015-115',
b'GBR00015-116',
b'GBR00015-117',
b'GBR00015-118',
b'GBR00015-119',
b'GBR00015-120',
b'GBR00015-121',
b'GBR00015-122',
b'GBR00015-123',
b'GBR00015-124',
b'IRL00017-001',
b'IRL00017-002',
b'IRL00017-003',
b'IRL00017-004',
b'IRL00017-005',
b'IRL00017-006',
b'IRL00017-007',
b'IRL00017-008',
b'IRL00017-009',
b'IRL00017-010',
b'IRL00017-011',
b'IRL00017-012',
b'IRL00017-013',
b'IRL00017-014',
b'IRL00017-015',
b'IRL00017-016',
b'IRL00017-017',
b'IRL00017-018',
b'IRL00017-019',
```

```
b'IRL00017-020',
b'ITA00243-001',
b'ITA00243-002',
b'ITA00243-003',
b'ITA00243-004',
b'ITA00243-005',
b'ITA00243-006',
b'ITA00243-007',
b'ITA00243-008',
b'ITA00243-009',
b'ITA00243-010',
b'ITA00243-011',
b'ITA00243-012',
b'ITA00243-013',
b'ITA00243-014',
b'ITA00243-015',
b'ITA00243-016',
b'ITA00243-017',
b'ITA00243-018',
b'ITA00243-019',
b'ITA00243-020',
b'ITA00243-021',
b'ITA00243-022',
b'ITA00243-023',
b'ITA00243-024',
b'ITA00243-025',
b'ITA00248-001',
b'ITA00248-002',
b'ITA00248-003',
b'ITA00248-004',
b'ITA00248-005',
b'ITA00248-006',
b'ITA00248-007',
b'ITA00248-009',
b'ITA00248-010',
b'ITA00248-011',
b'ITA00248-012',
b'ITA00248-013',
b'ITA00248-014',
b'ITA00248-015',
b'ITA00248-017',
b'ITA00248-018',
b'ITA00248-019',
b'ITA00248-020',
b'ITA00248-021',
b'ITA00248-022',
b'ITA00248-023',
b'ITA00248-024',
b'ITA00248-025',
b'LUX00021-001',
b'LUX00021-002',
b'LUX00021-003',
b'LUX00021-004',
b'LUX00021-005',
b'LUX00021-006',
b'LUX00021-007',
b'LUX00021-008',
b'LUX00021-009',
b'LUX00021-010',
b'LUX00021-011',
```

```
b'LUX00021-012',
b'LUX00021-013',
b'LUX00021-014',
b'LUX00021-015',
b'LUX00021-016',
b'LUX00021-017',
b'LUX00021-018',
b'LUX00021-019',
b'LUX00021-020',
b'LUX00021-021',
b'LUX00021-022',
b'LUX00021-023',
b'LUX00021-024',
b'LUX00021-025',
b'NLD00013-001',
b'NLD00013-002',
b'NLD00013-003',
b'NLD00013-004',
b'NLD00013-005',
b'NLD00013-006',
b'NLD00013-007',
b'NLD00013-008',
b'NLD00013-009',
b'NLD00013-010',
b'NLD00013-011',
b'NLD00013-012',
b'NLD00013-013',
b'NLD00013-014',
b'NLD00013-015',
b'NLD00013-016',
b'NLD00013-017',
b'NLD00013-018',
b'NLD00013-019',
b'NLD00013-020',
b'NLD00013-021',
b'NLD00013-022',
b'NLD00013-023',
b'NLD00013-024',
b'NLD00013-025',
b'SWE00093-001',
b'SWE00093-002',
b'SWE00093-003',
b'SWE00093-004',
b'SWE00093-005',
b'SWE00093-006',
b'SWE00093-007',
b'SWE00093-008',
b'SWE00093-009',
b'SWE00093-010',
b'SWE00093-011',
b'SWE00093-012',
b'SWE00093-013',
b'SWE00093-014',
b'SWE00093-015',
b'SWE00093-016',
b'SWE00093-017',
b'SWE00093-018',
b'SWE00093-019',
b'SWE00093-020',
b'SWE00093-021',
```

```
 b'SWE00093-022',
 b'SWE00093-023',
 b'SWE00093-024',
 b'SWE00093-025']
```

In [60]:
```
samples_fn = '~/scratch/data/Bpendula/Betula_pendula_sample_list_scikit-a
samples = pandas.read_csv(samples_fn, sep='\t')
samples
```

Out[60]:

	ID	Population
0	ESP00102-001	ESP00102
1	ESP00102-002	ESP00102
2	ESP00102-003	ESP00102
3	ESP00102-004	ESP00102
4	ESP00102-005	ESP00102
...	...	...
359	SWE00093-021	SWE00093
360	SWE00093-022	SWE00093
361	SWE00093-023	SWE00093
362	SWE00093-024	SWE00093
363	SWE00093-025	SWE00093

364 rows × 2 columns

In [61]:
```
samples.Population.value_counts()
```

Out[61]:
```
Population
ESP00102    25
ESP00199    25
ESP00337    25
FIN00015    25
FIN00020    25
FIN00046    25
ITA00243    25
NLD00013    25
SWE00093    25
LUX00021    25
FIN00044    24
GBR00015    24
GBR00013    23
ITA00248    23
IRL00017    20
Name: count, dtype: int64
```

In [62]:
```
populations = samples.Population.unique()
populations
###This identifiers come from the metadata file
```

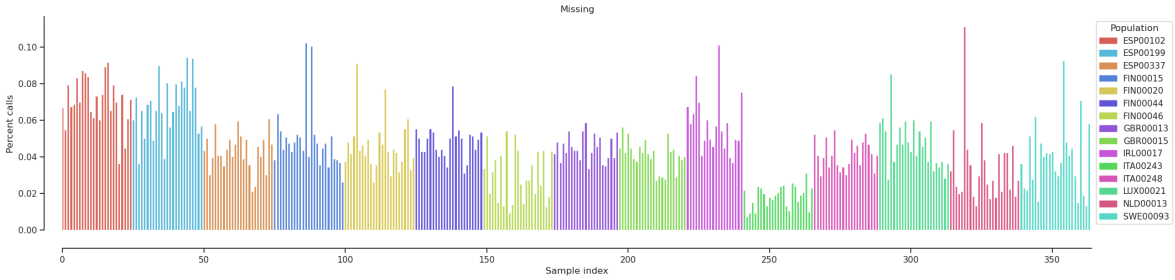
Out[62]:
```
array(['ESP00102', 'ESP00199', 'ESP00337', 'FIN00015', 'FIN00020',
       'FIN00044', 'FIN00046', 'GBR00013', 'GBR00015', 'IRL00017',
       'ITA00243', 'ITA00248', 'LUX00021', 'NLD00013', 'SWE00093'],
      dtype=object)
```

Gt frequency function

In [63]:
```
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", 15)
    pop2color = {'ESP00102': palette[0],
                 'ESP00199': palette[8],
                 'ESP00337': palette[1],
                 'FIN00015': palette[9],
                 'FIN00020': palette[2],
                 'FIN00044': palette[10],
                 'FIN00046': palette[3],
                 'GBR00013': palette[11],
                 'GBR00015': palette[4],
                 'IRL00017': palette[12],
                 'ITA00243': palette[5],
                 'ITA00248': palette[13],
                 'LUX00021': palette[6],
                 'NLD00013': palette[14],
                 'SWE00093': palette[7]}
    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])]
    ax.legend(handles=handles, labels=['ESP00102', 'ESP00199', 'ESP00337'
       'FIN00044', 'FIN00046', 'GBR00013', 'GBR00015', 'IRL00017',
       'ITA00243', 'ITA00248', 'LUX00021', 'NLD00013', 'SWE00093'], title
              bbox_to_anchor=(1, 1), loc='upper left')
```

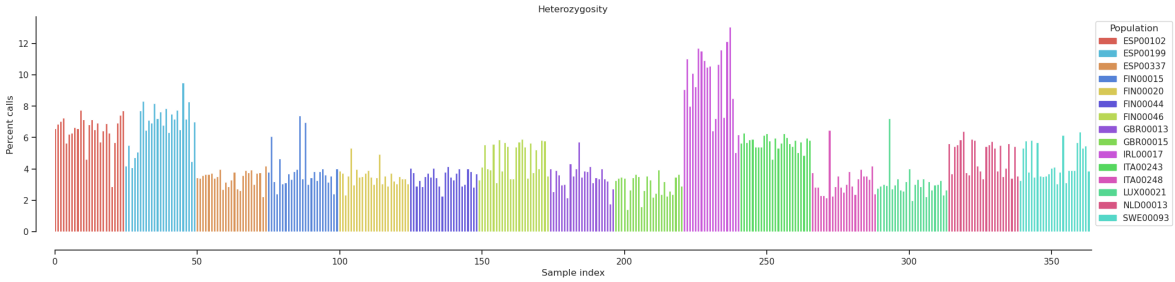
Plot missing

In [64]:
```
plot_genotype_frequency(pc_missing, 'Missing')
```



Plot heterozygosity

In [65]:
```
plot_genotype_frequency(pc_het, 'Heterozygosity')
```



PCA

In [66]:
```
palette = sns.color_palette("hls",15)
pop_colours = {
                'ESP00102': palette[0],
                 'ESP00199': palette[8],
                 'ESP00337': palette[1],
                 'FIN00015': palette[9],
                 'FIN00020': palette[2],
                 'FIN00044': palette[10],
                 'FIN00046': palette[3],
                 'GBR00013': palette[11],
                 'GBR00015': palette[4],
                 'IRL00017': palette[12],
                 'ITA00243': palette[5],
                 'ITA00248': palette[13],
                 'LUX00021': palette[6],
                 'NLD00013': palette[14],
                 'SWE00093': palette[7]
}
```

In [67]:
```
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 [68]:
```
ac2 = gt_biallelic.count_alleles()
ac2
```

Out[68]: <AlleleCountsChunkedArray shape=(151255, 2) dtype=int32 chunks=(37814, 2) nbytes=1.2M cbytes=298.6K cratio=4.0 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1
0	727	1
1	250	478
2	705	23
...	...	
151252	726	2
151253	724	4
151254	726	2

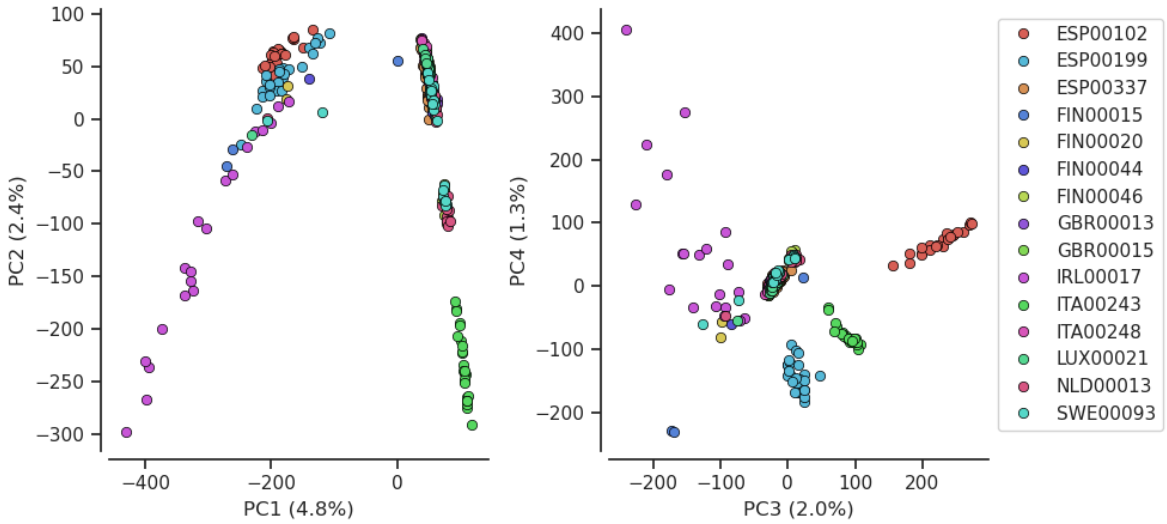
In [69]:
```
flt = (ac2[:, :2].min(axis=1) > 1)
gf = gt_biallelic.compress(flt, axis=0)
gn = gf.to_n_alt()
gn
```

Out[69]:
```
<ChunkedArrayWrapper shape=(116045, 364) dtype=int8 chunks=(1814, 364)
   nbytes=40.3M cbytes=5.7M cratio=7.0
   compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shu
ffle': 1, 'blocksize': 0}
   values=zarr.core.Array>
```

In [70]:
```
coords1, model1 = allel.pca(gn, n_components=10, scaler='patterson')
```

In [71]:
```
fig_pca(coords1, model1, 'Figure 1. Conventional PCA.')
```

Figure 1. Conventional PCA.



In [72]:
```
outliers = coords1[:,3]<-200
samples[outliers]
```

Out[72]:

	ID	Population
86	FIN00015-012	FIN00015
88	FIN00015-014	FIN00015

In [73]:
```
pc_het[outliers]
```

Out[73]:
```
array([7.41396979, 6.98092625])
```

In [74]:
```
pc_missing[outliers]
```

Out[74]:
```
array([0.10247595, 0.10049255])
```