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/vcf_filtering/Qrobur/ra
```

Get data

In [3]:
```
callset_var_fn = '/users/mcevoysu/scratch/output/scikit-allel/Qrobur/raw_
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=(661793,) 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=113.0M cbytes=25.1M cratio=4.5 values=h5py._hl.group.Group>

	AC	AF	ALT	AN	BaseQRankSum	CHROM	DP
0	[10 -1 -1]	[0.012 nan nan]	[b'A' b'' b'']	804	nan	b'Qrob_Chr01'	115
1	[2 -1 -1]	[0.002463 nan nan]	[b'G' b'' b'']	804	nan	b'Qrob_Chr01'	117
2	[2 -1 -1]	[0.002463 nan nan]	[b'A' b'' b'']	804	nan	b'Qrob_Chr01'	71
...							
661790	[26 -1 -1]	[0.032 nan nan]	[b'G' b'' b'']	804	nan	b'Qrob_H2.3_Sc0001194'	83
661791	[28 -1 -1]	[0.034 nan nan]	[b'T' b'' b'']	804	nan	b'Qrob_H2.3_Sc0001194'	83
661792	[2 -1 -1]	[0.002463 nan nan]	[b'C' b'' b'']	804	nan	b'Qrob_H2.3_Sc0001194'	85

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

Out[7]: <VariantTable shape=(437335,) 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
0	[10 -1 -1]	[0.012 nan nan]	[b'A' b'' b'']	804	nan	b'Qrob_Chr01'	115
1	[2 -1 -1]	[0.002463 nan nan]	[b'G' b'' b'']	804	nan	b'Qrob_Chr01'	117
2	[2 -1 -1]	[0.002463 nan nan]	[b'A' b'' b'']	804	nan	b'Qrob_Chr01'	71
...							
437332	[26 -1 -1]	[0.032 nan nan]	[b'G' b'' b'']	804	nan	b'Qrob_H2.3_Sc0001194'	83
437333	[28 -1 -1]	[0.034 nan nan]	[b'T' b'' b'']	804	nan	b'Qrob_H2.3_Sc0001194'	83
437334	[2 -1 -1]	[0.002463 nan nan]	[b'C' b'' b'']	804	nan	b'Qrob_H2.3_Sc0001194'	85

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

Out[8]: <VariantTable shape=(224458,) 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
0	[11 -1 -1]	[0.014 nan nan]	[b'*' b'' b'']	804	nan	b'Qrob_Chr01'	38
1	[74 -1 -1]	[0.096 nan nan]	[b'*' b'' b'']	798	nan	b'Qrob_Chr01'	42
2	[67 -1 -1]	[0.087 nan nan]	[b'*' b'' b'']	800	nan	b'Qrob_Chr01'	42
...							
224455	[2 -1 -1]	[0.002463 nan nan]	[b'*' b'' b'']	804	-1.243	b'Qrob_H2.3_Sc0001028'	682
224456	[696 -1 -1]	[0.867 nan nan]	[b'*' b'' b'']	804	nan	b'Qrob_H2.3_Sc0001028'	687
224457	[1 -1 -1]	[0.001232 nan nan]	[b'*' b'' b'']	804	nan	b'Qrob_H2.3_Sc0001163'	50

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, 420303,  16639,    393])
```

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

Out[12]:
```
17032
```

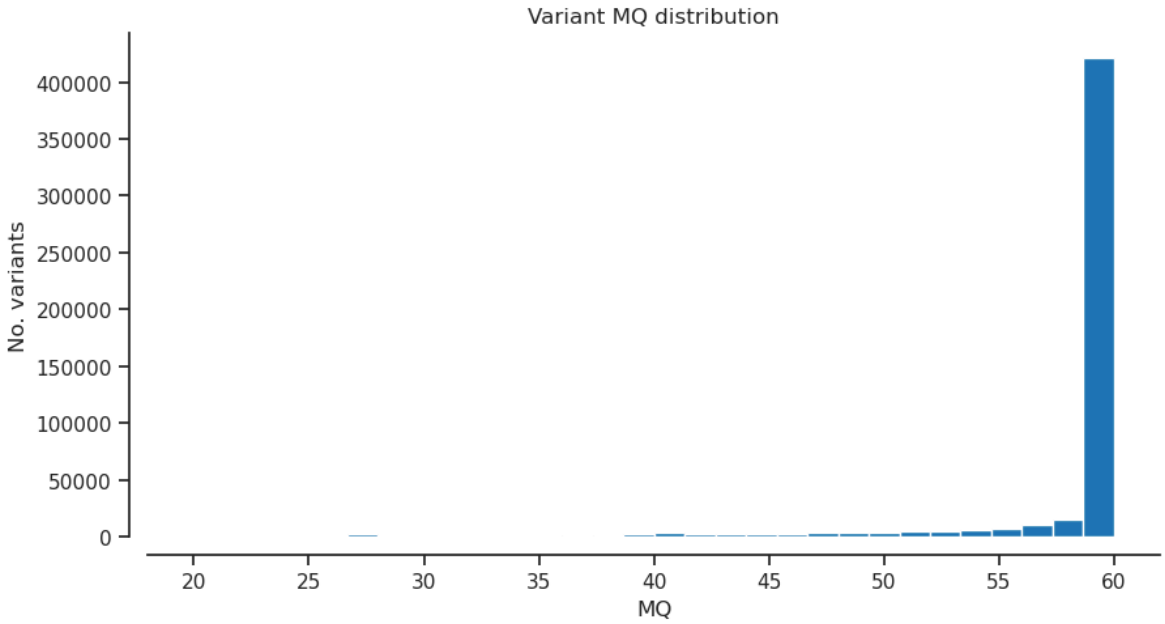
In [13]:
```
filter_expression = '(numalt == 1)'
biallelic_np = rawsnps.query(filter_expression)[:]
biallelic_np
```

Out[13]: <VariantTable shape=(420303,) 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')])>

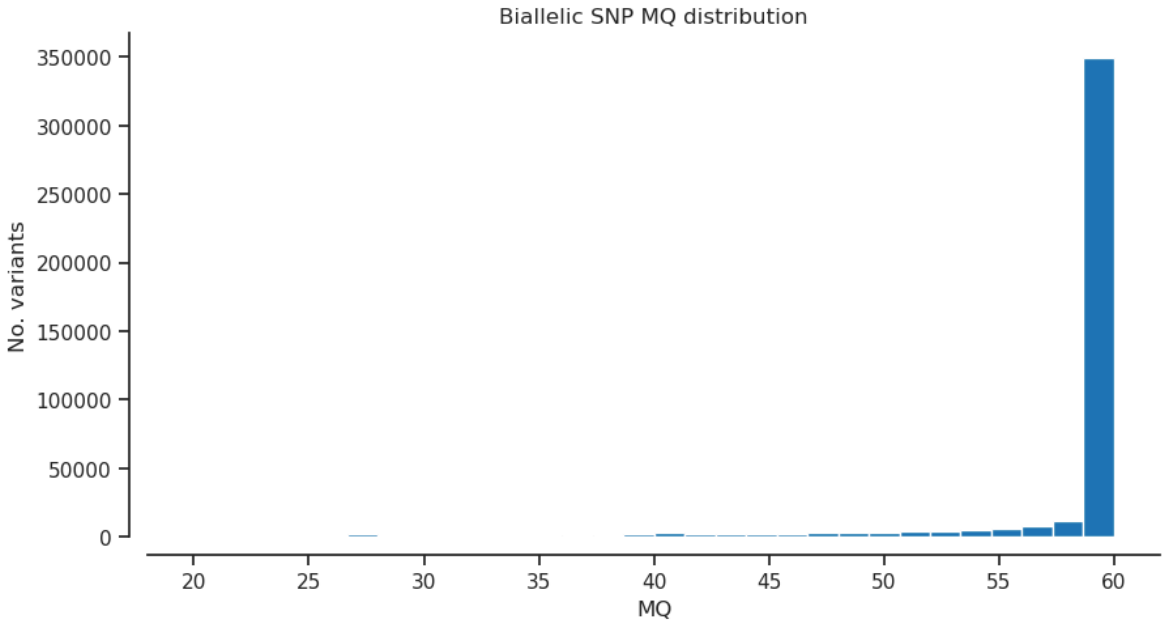
	AC	AF	ALT	AN	BaseQRankSum	CHROM	DP
0	[10 -1 -1]	[0.012 nan nan]	[b'A' b'' b'']	804	nan	b'Qrob_Chr01'	115
1	[2 -1 -1]	[0.002463 nan nan]	[b'G' b'' b'']	804	nan	b'Qrob_Chr01'	117
2	[2 -1 -1]	[0.002463 nan nan]	[b'A' b'' b'']	804	nan	b'Qrob_Chr01'	71
...							
420300	[26 -1 -1]	[0.032 nan nan]	[b'G' b'' b'']	804	nan	b'Qrob_H2.3_Sc0001194'	83
420301	[28 -1 -1]	[0.034 nan nan]	[b'T' b'' b'']	804	nan	b'Qrob_H2.3_Sc0001194'	83
420302	[2 -1 -1]	[0.002463 nan nan]	[b'C' b'' b'']	804	nan	b'Qrob_H2.3_Sc0001194'	85

MQ - RMS mapping quality

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

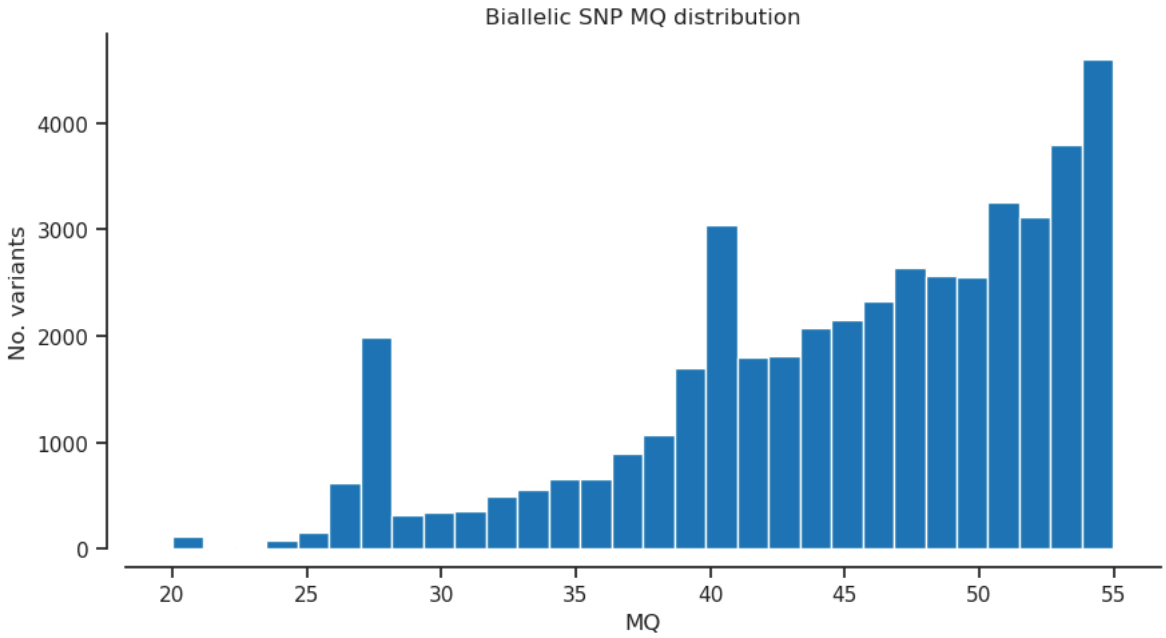


In [15]:
```
plot_hist('MQ','biallelic') # RMS mapping quality
```



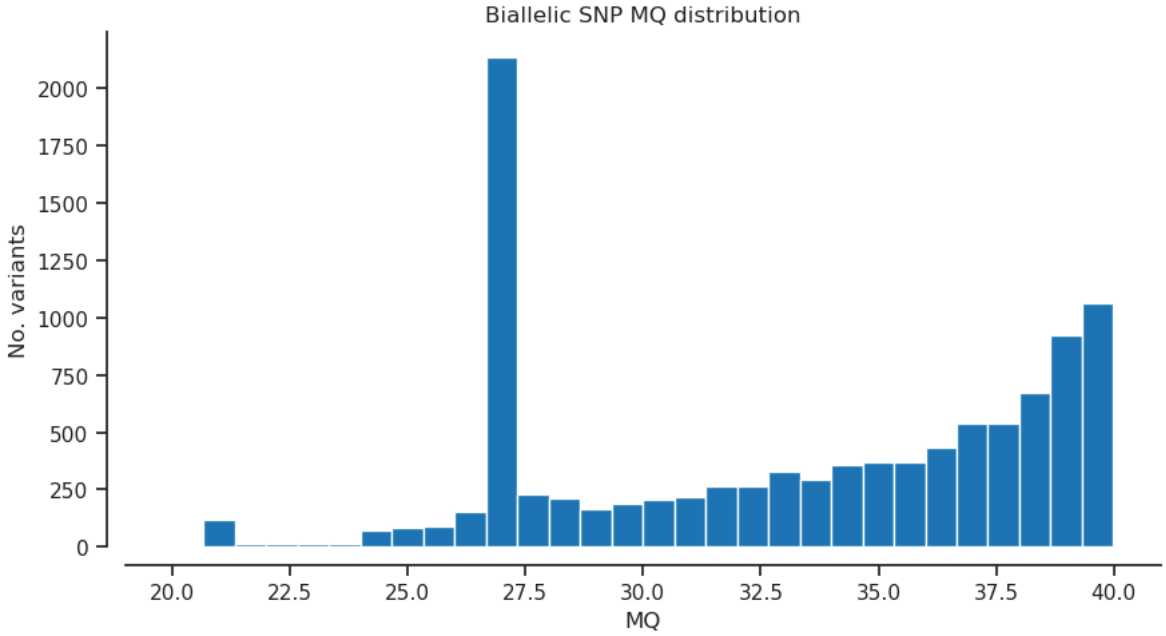
In [16]:
```
filter_expression = '(MQ < 55)'
bi_selection = biallelic_np.query(filter_expression)[:]
#np.count_nonzero(var_selection)
```

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

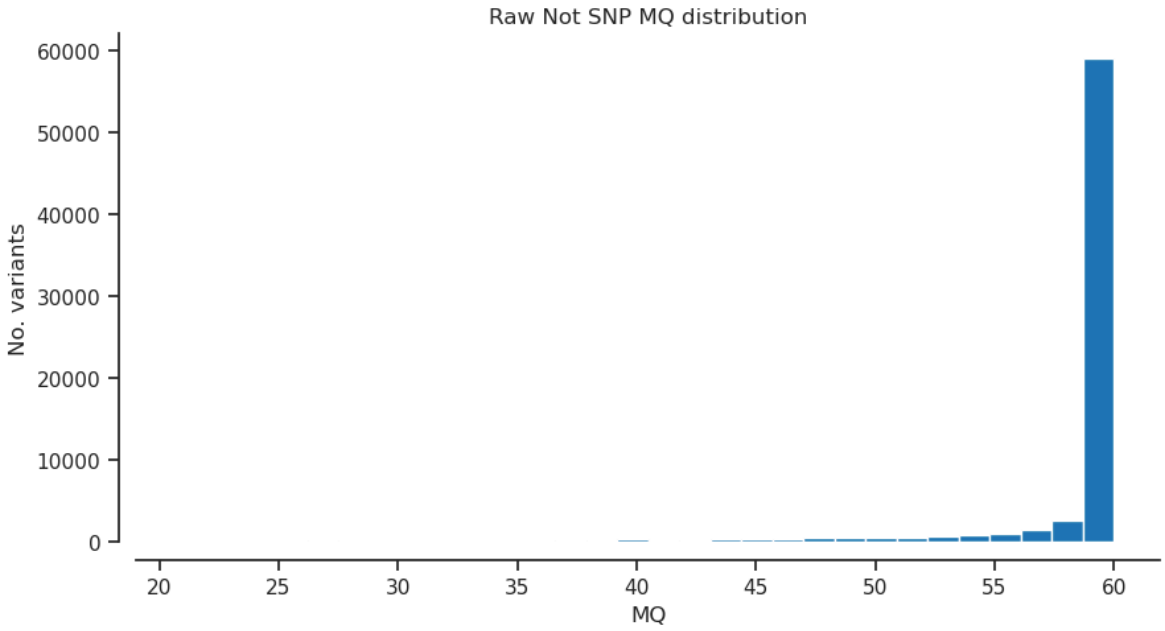


In [18]:
```
filter_expression = '(MQ < 40)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

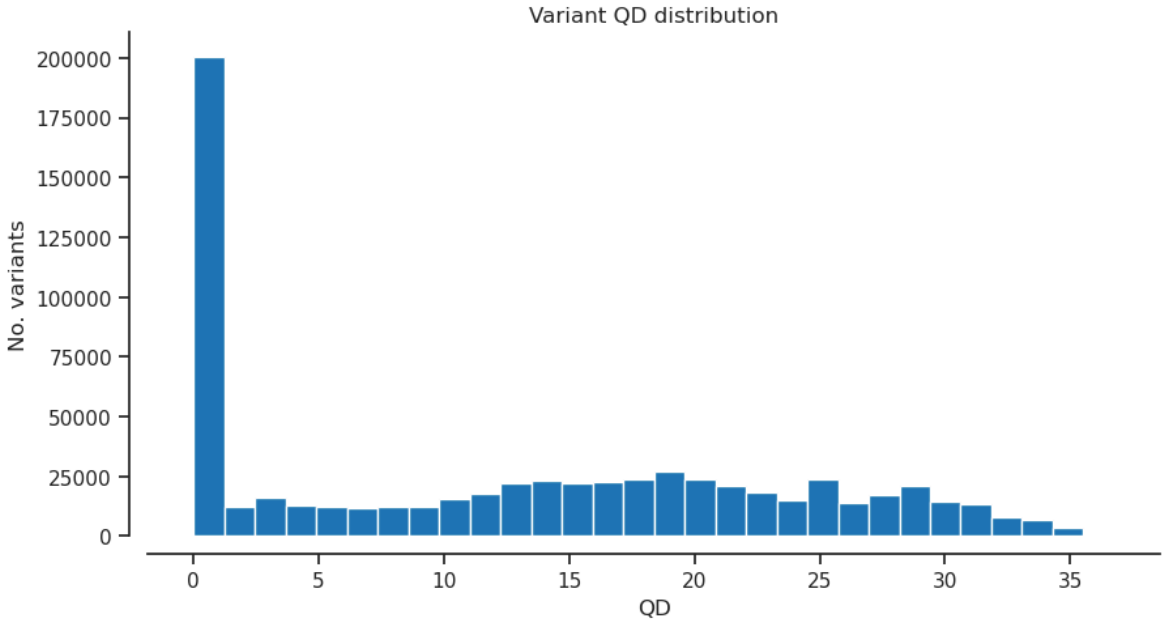


In [20]:
```
plot_hist('MQ','notsnp')
```

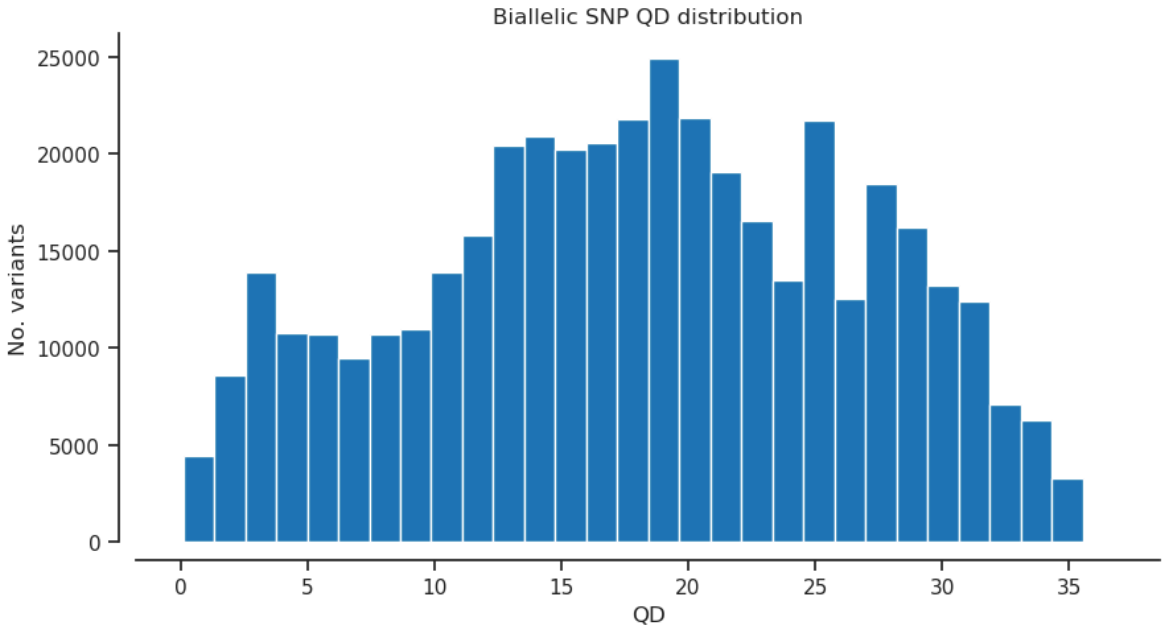


QD - Variant Confidence/Quality by Depth

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

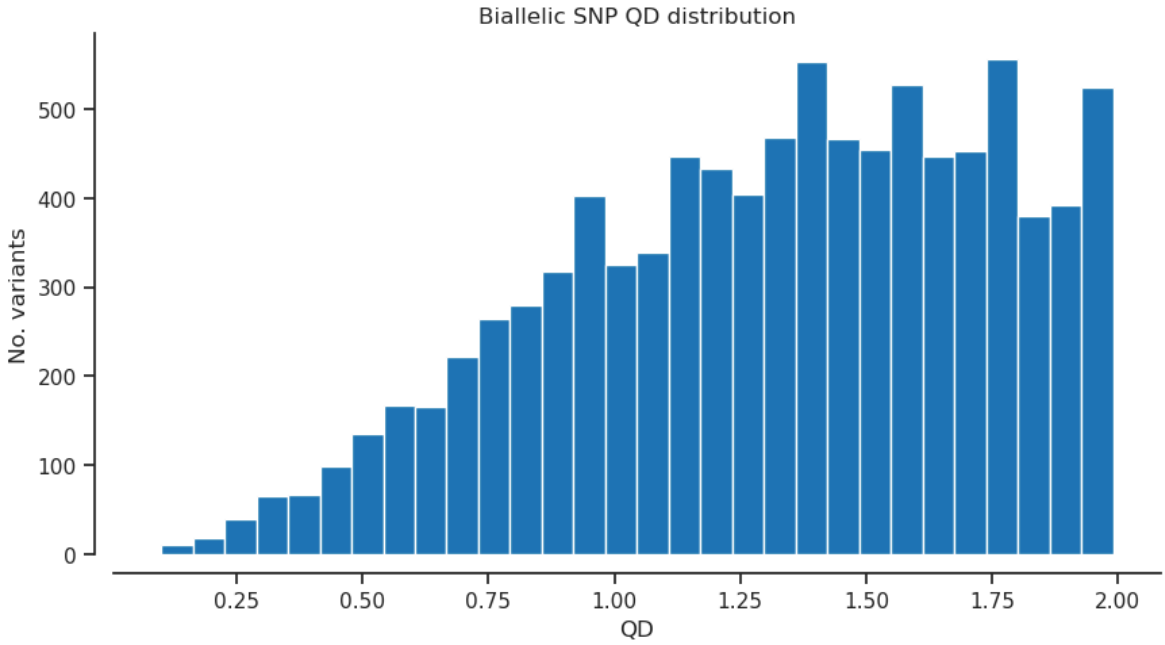


In [22]:
```
plot_hist('QD','biallelic') # Variant Confidence/Quality by Depth
```

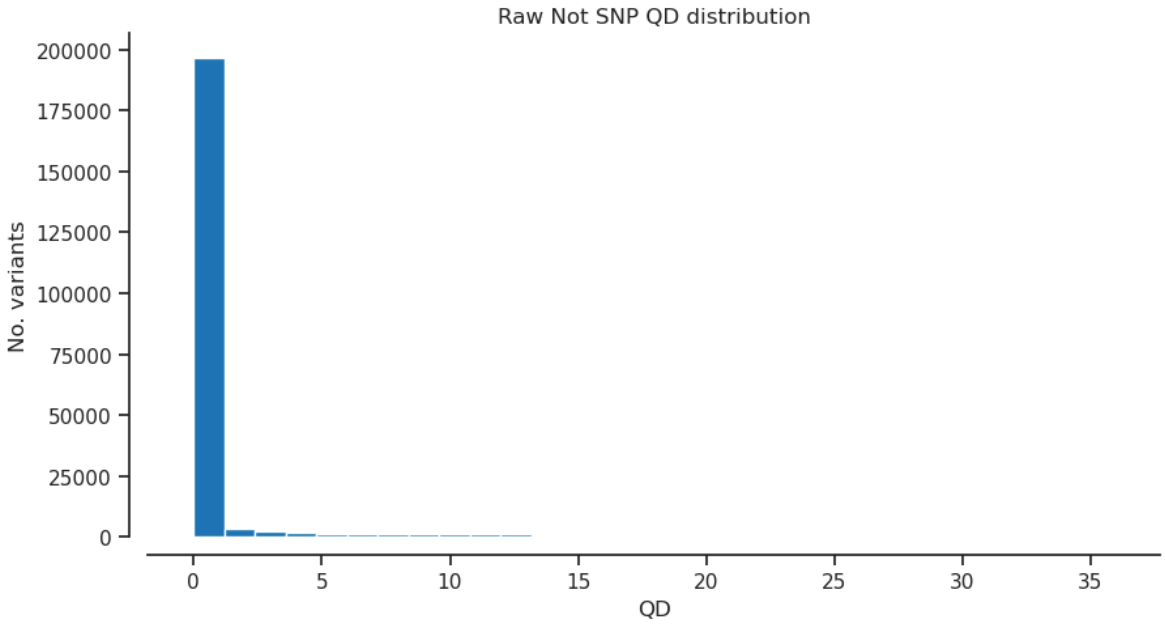


In [23]:
```
filter_expression = '(QD < 2)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

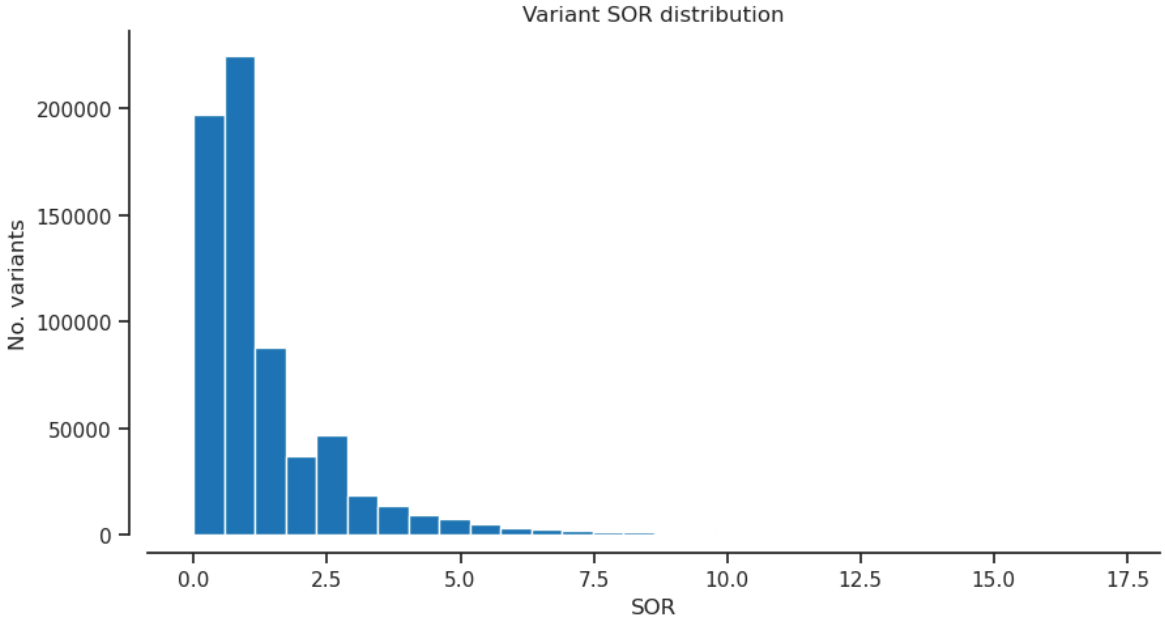


In [25]:
```
plot_hist('QD','notsnp') # Variant Confidence/Quality by Depth
```

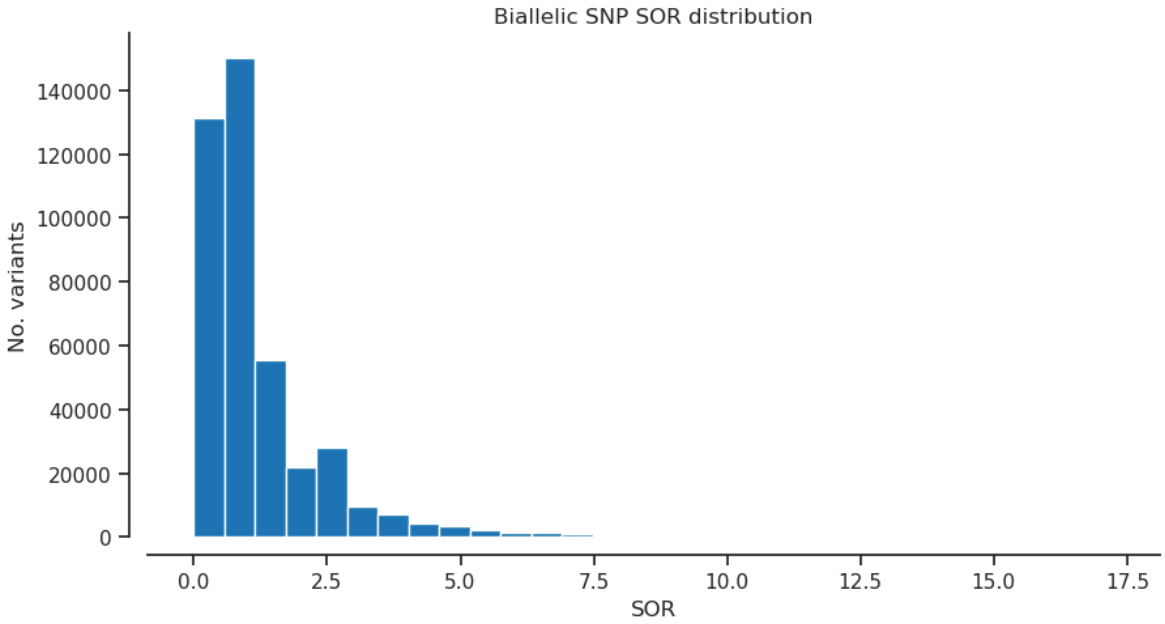


SOR - Symmetric Odds Ratio of 2x2 contingency table to detect strand bias

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

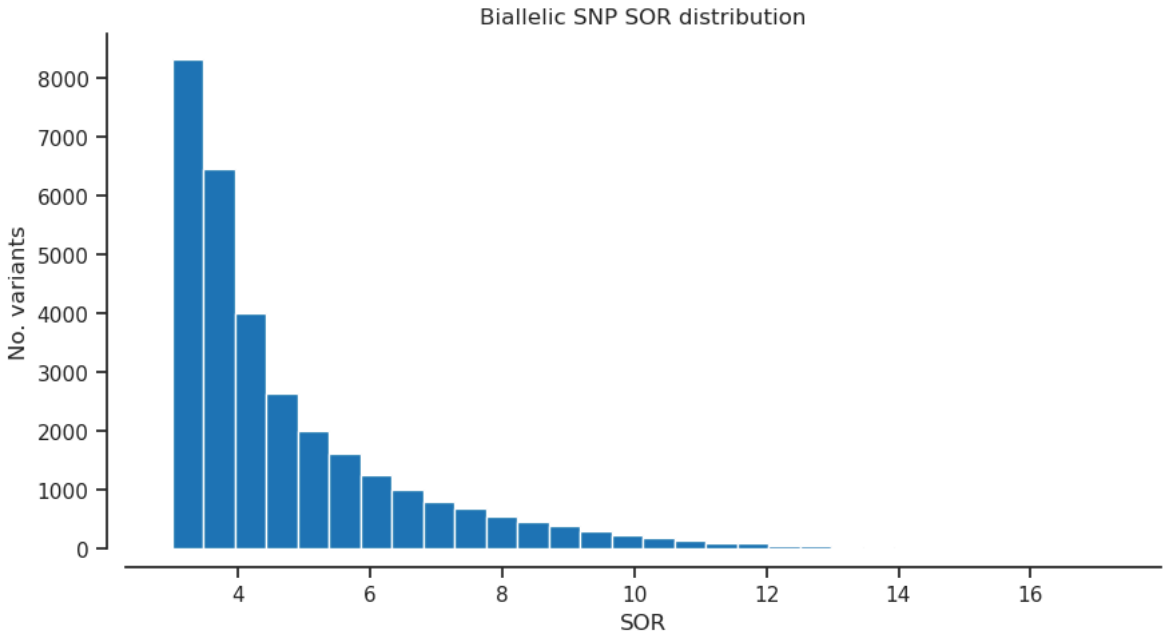


In [27]:
```
plot_hist('SOR','biallelic') # Symmetric Odds Ratio of 2x2 contingency ta
```

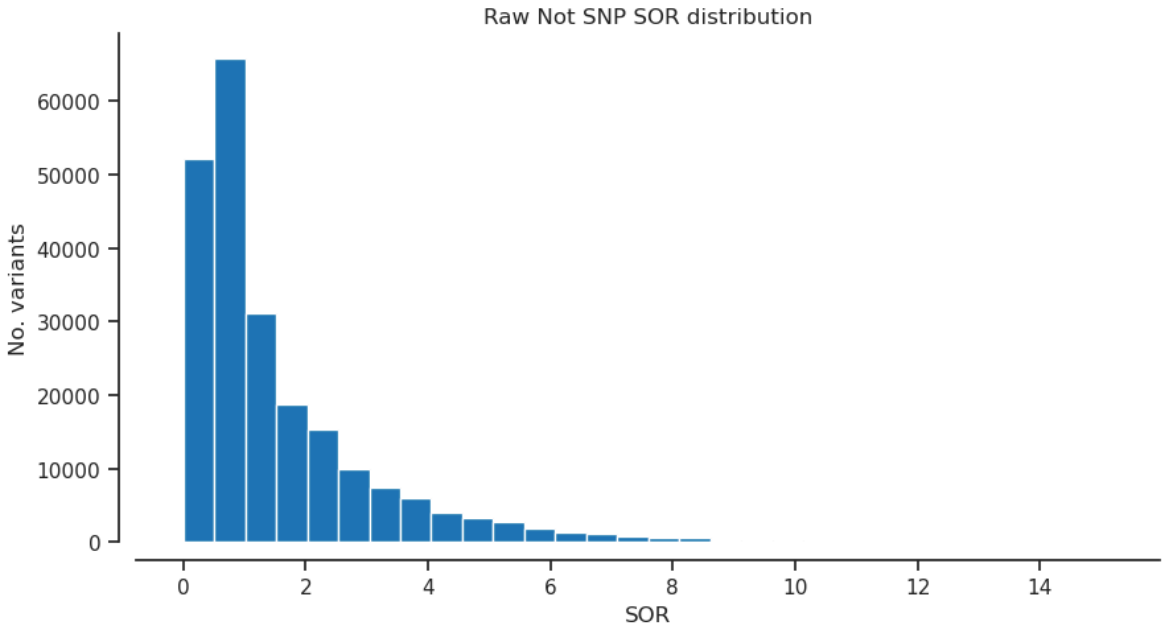


In [28]:
```
filter_expression = '(SOR > 3)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

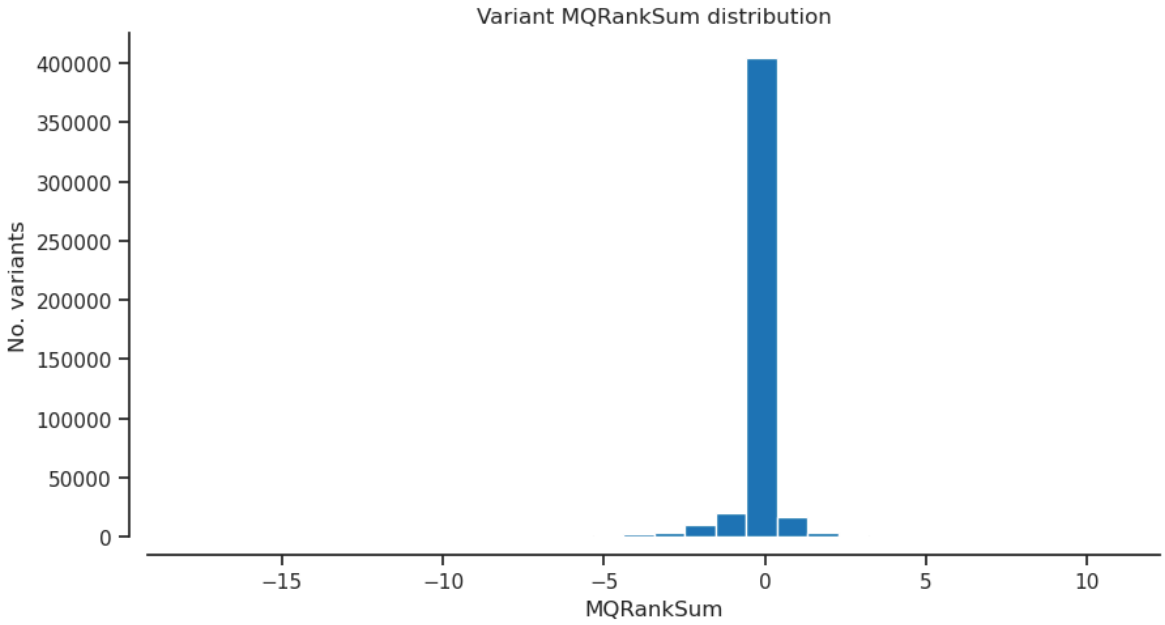


In [30]:
```
plot_hist('SOR','notsnp') # Symmetric Odds Ratio of 2x2 contingency table
```

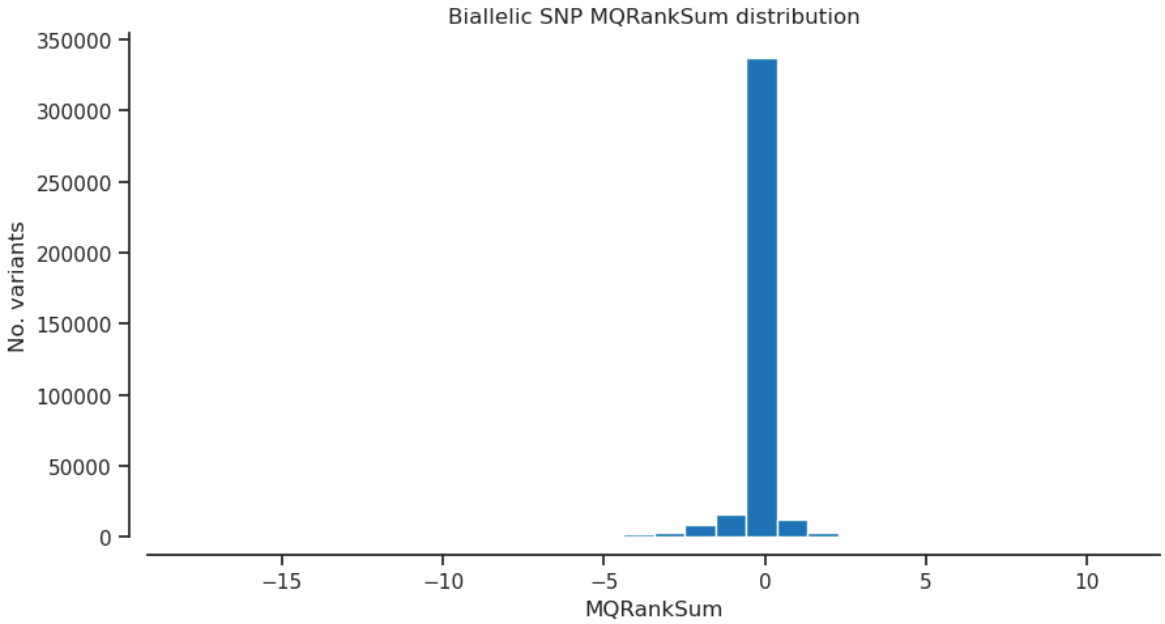


MQRankSum - Z-score From Wilcoxon rank sum test of Alt vs. Ref read mapping qualities

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

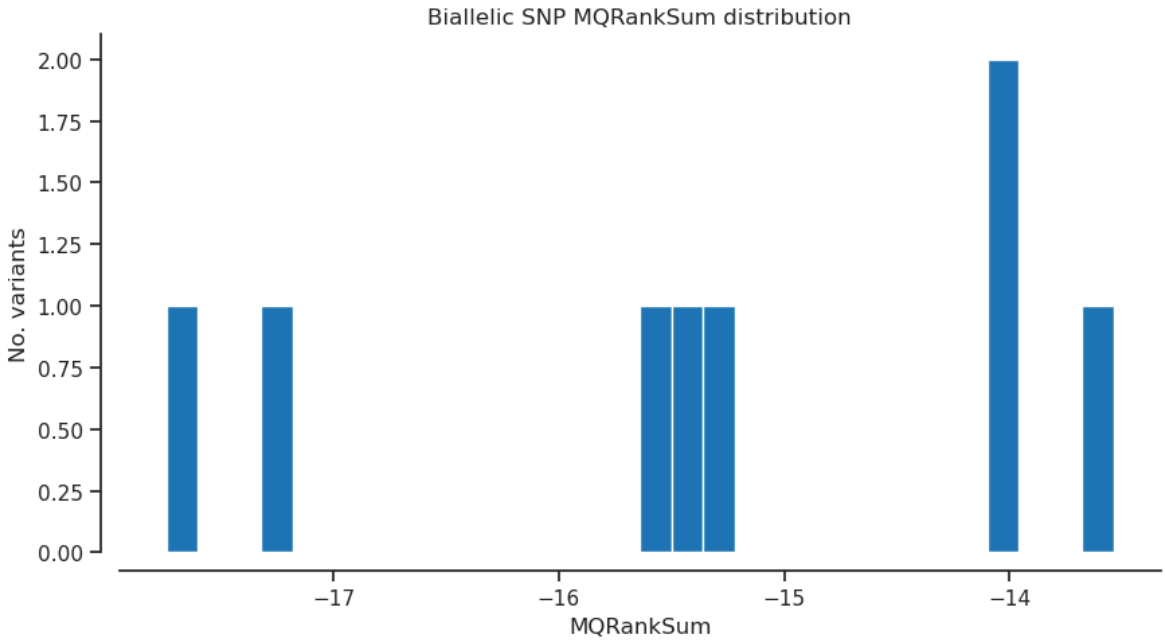


In [32]:
```
plot_hist('MQRankSum','biallelic') # Z-score From Wilcoxon rank sum test
```

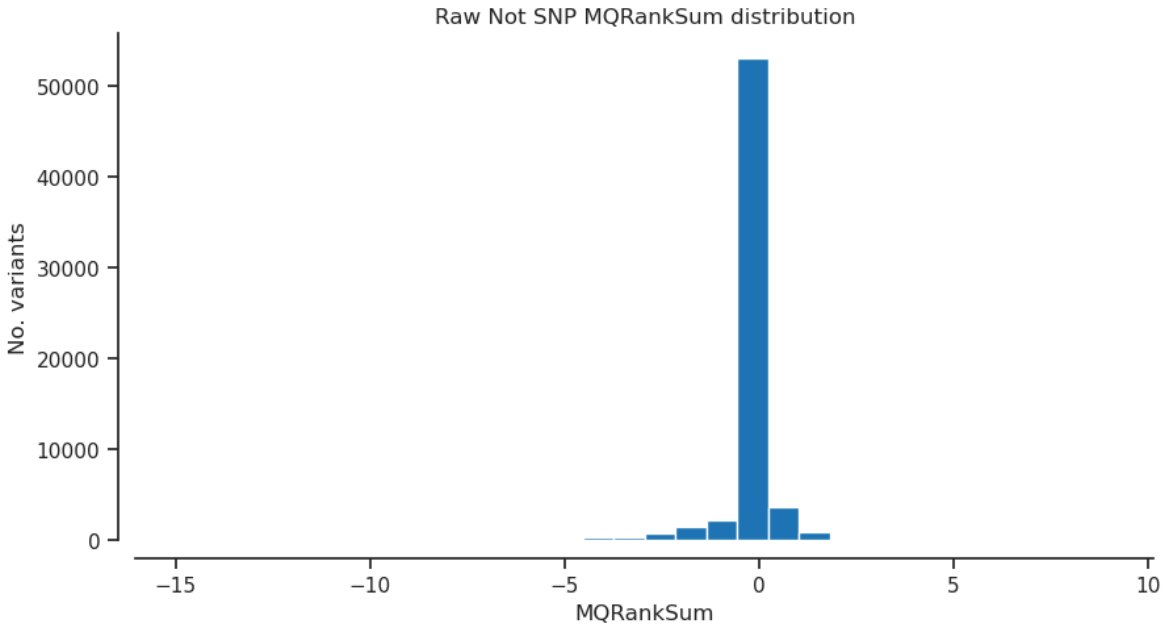


In [33]:
```
filter_expression = '(MQRankSum < -12.5)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

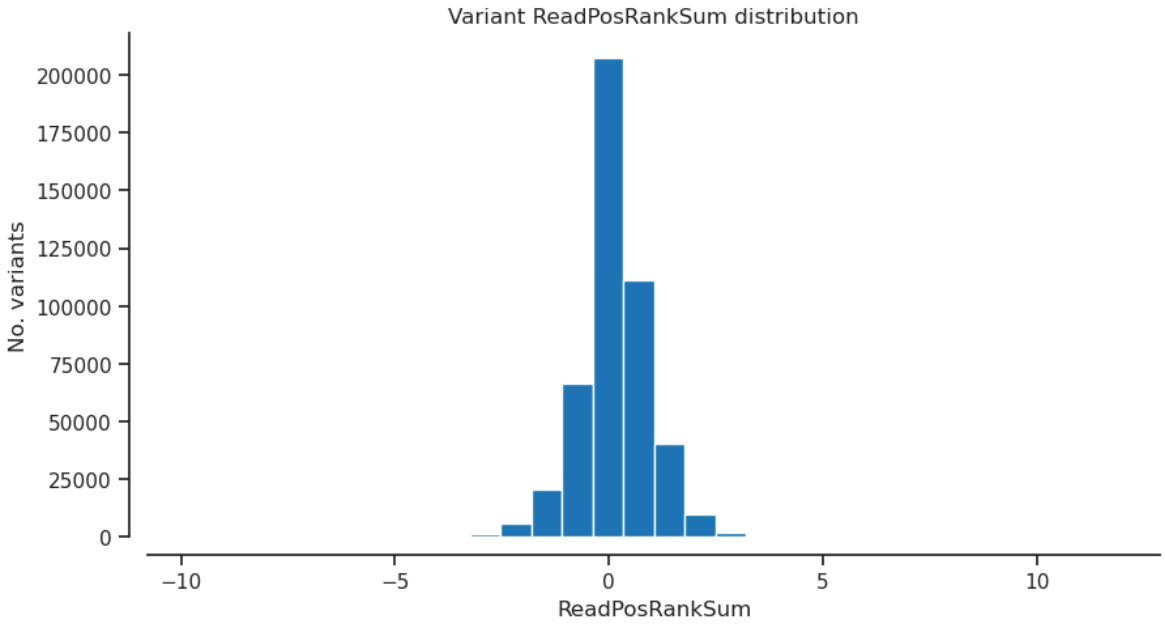


In [35]:
```
plot_hist('MQRankSum','notsnp') # Z-score From Wilcoxon rank sum test of
```

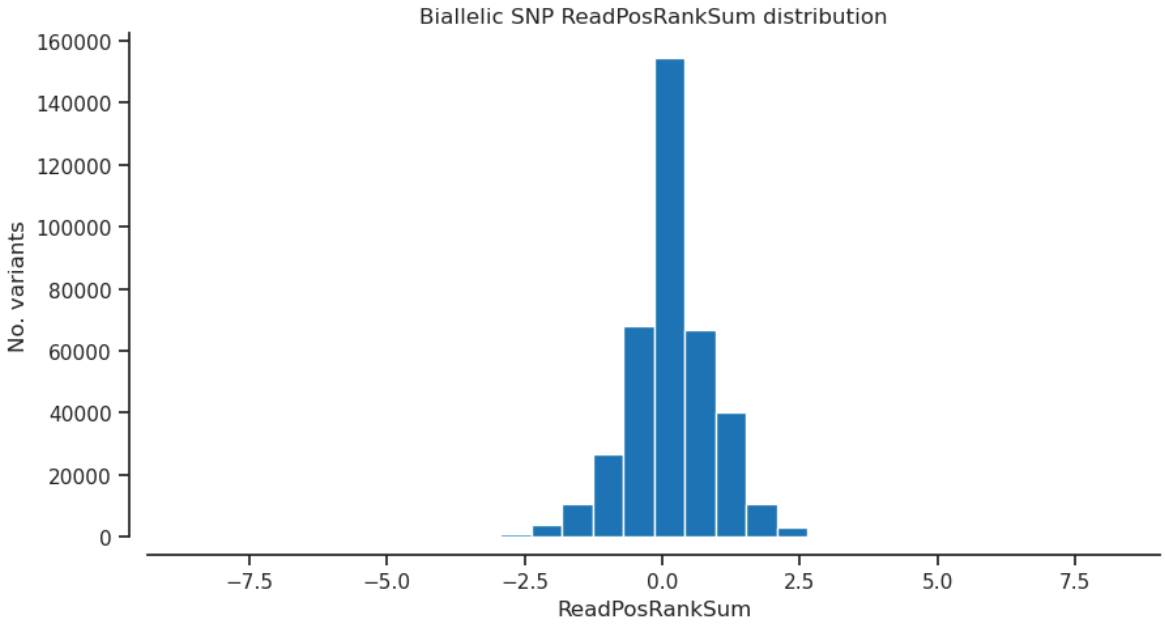


ReadPosRankSum - Z-score from Wilcoxon rank sum test of Alt vs. Ref read position bias

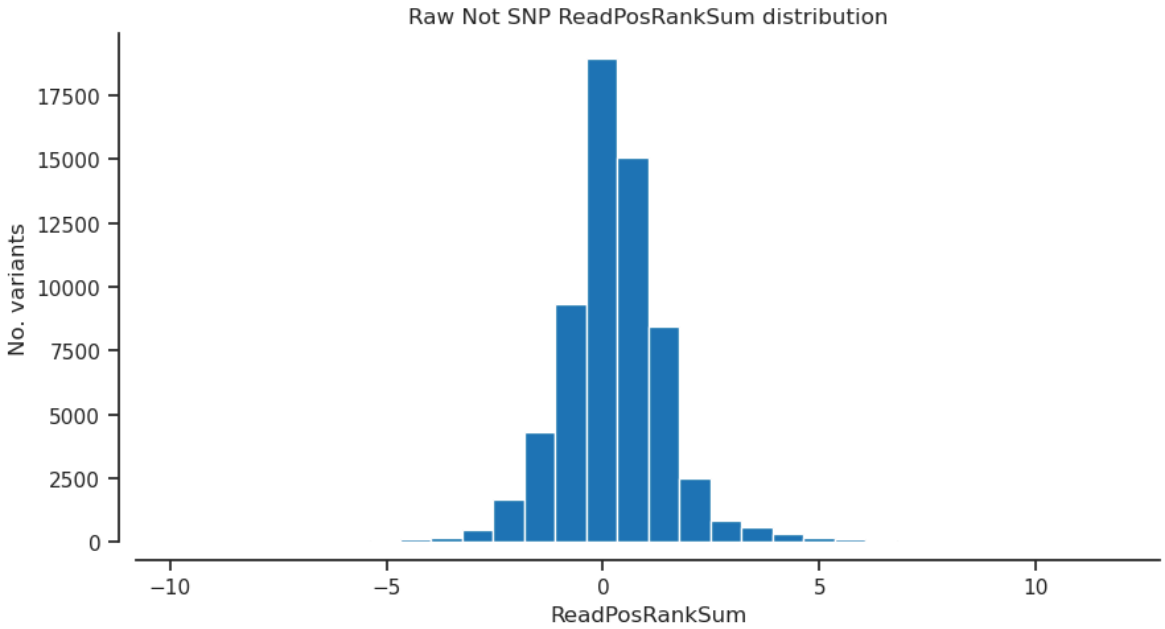
In [36]:
```
plot_hist('ReadPosRankSum','var') # Z-score from Wilcoxon rank sum test o
```



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

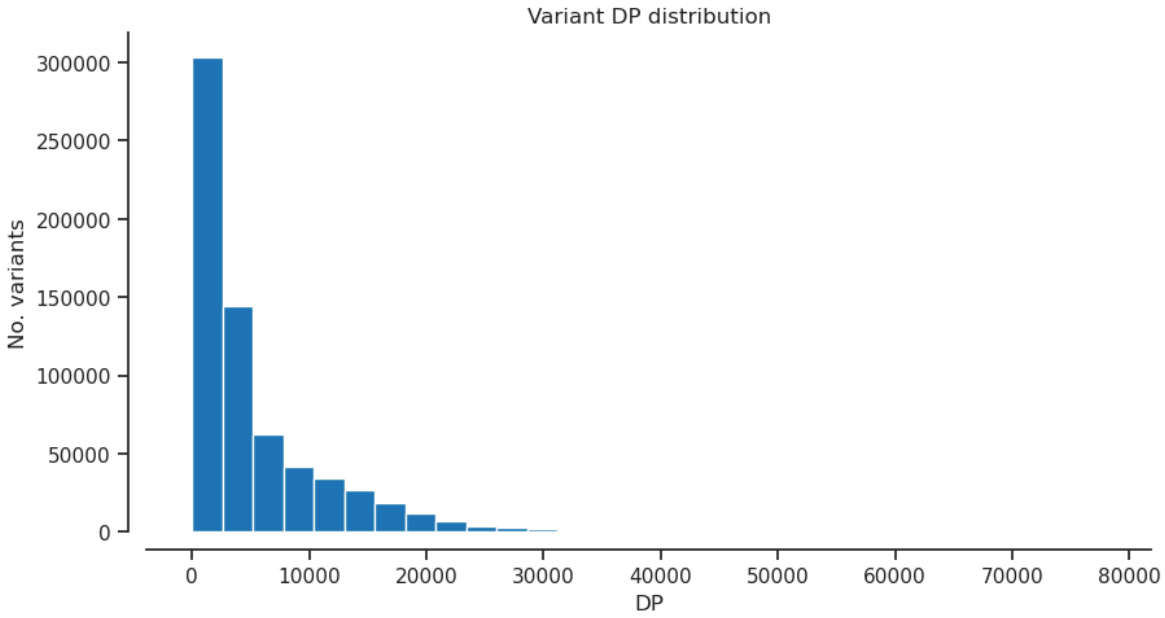


In [38]:
```
plot_hist('ReadPosRankSum','notsnp') # Z-score from Wilcoxon rank sum tes
```

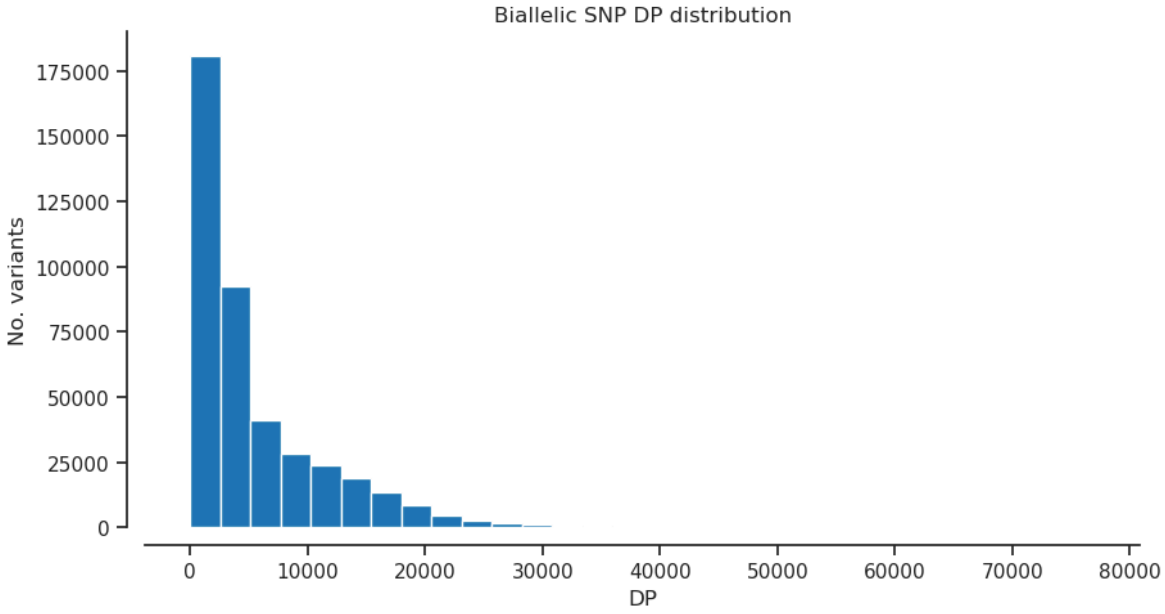


DP - Approximate read depth

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

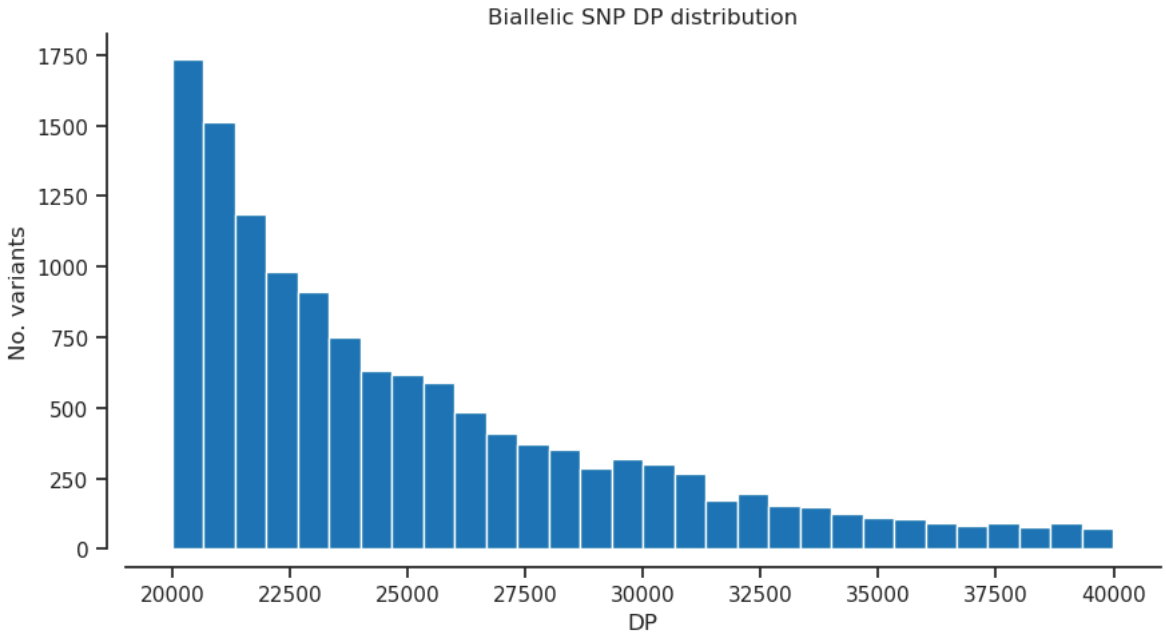


In [40]:
```
plot_hist('DP','biallelic')
```

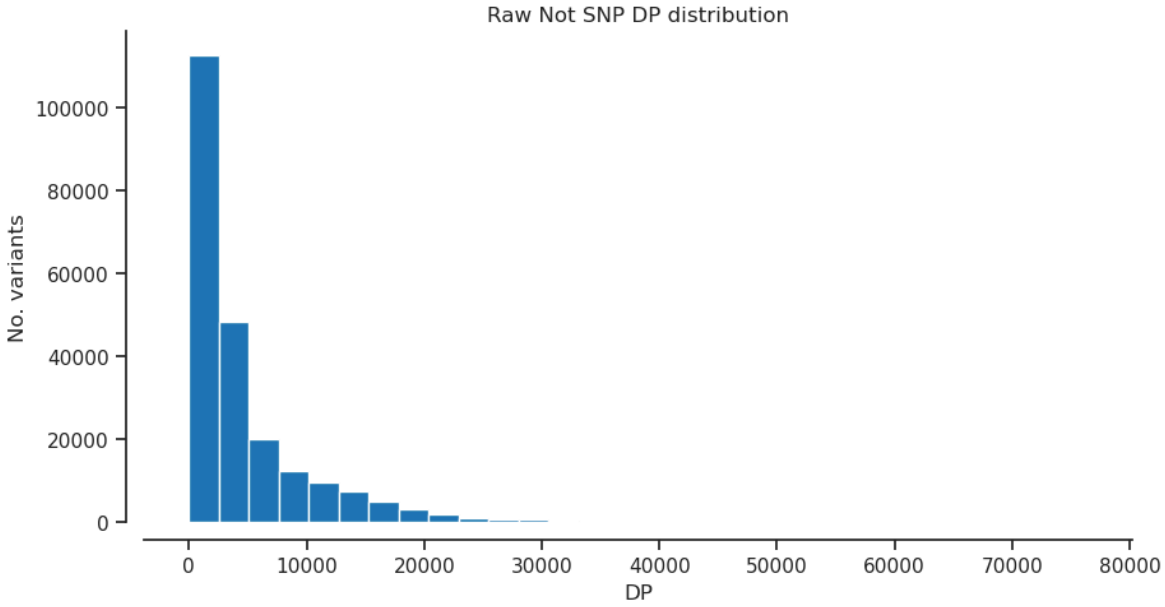


In [41]:
```
filter_expression = '(DP > 20000) & (DP < 40000)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

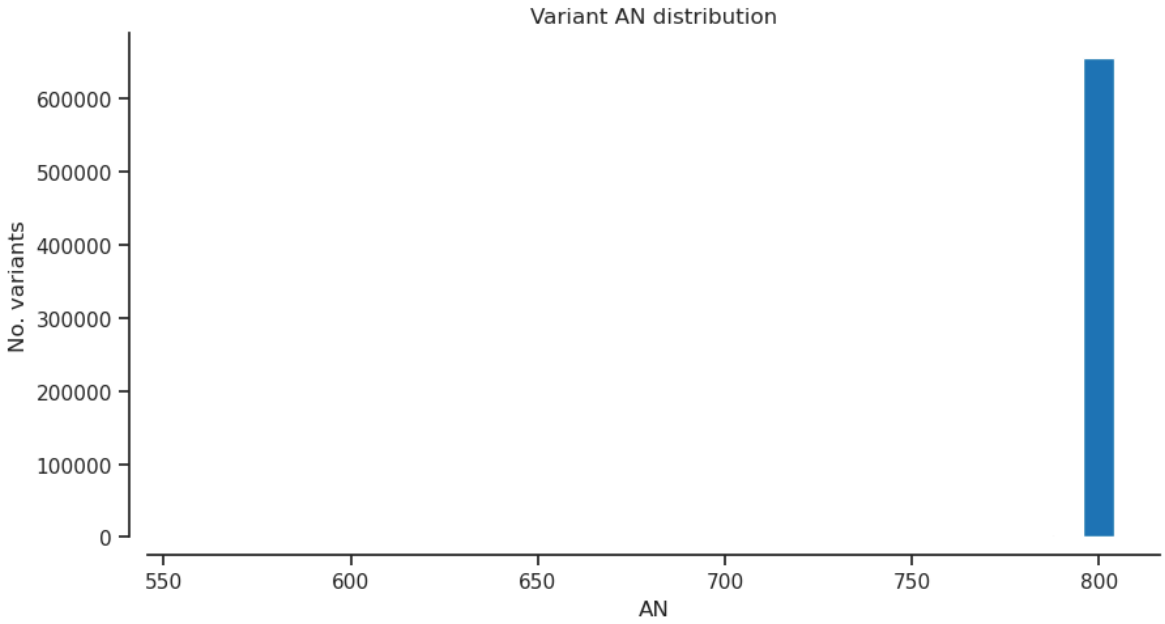


In [43]:
```
plot_hist('DP','notsnp')
```

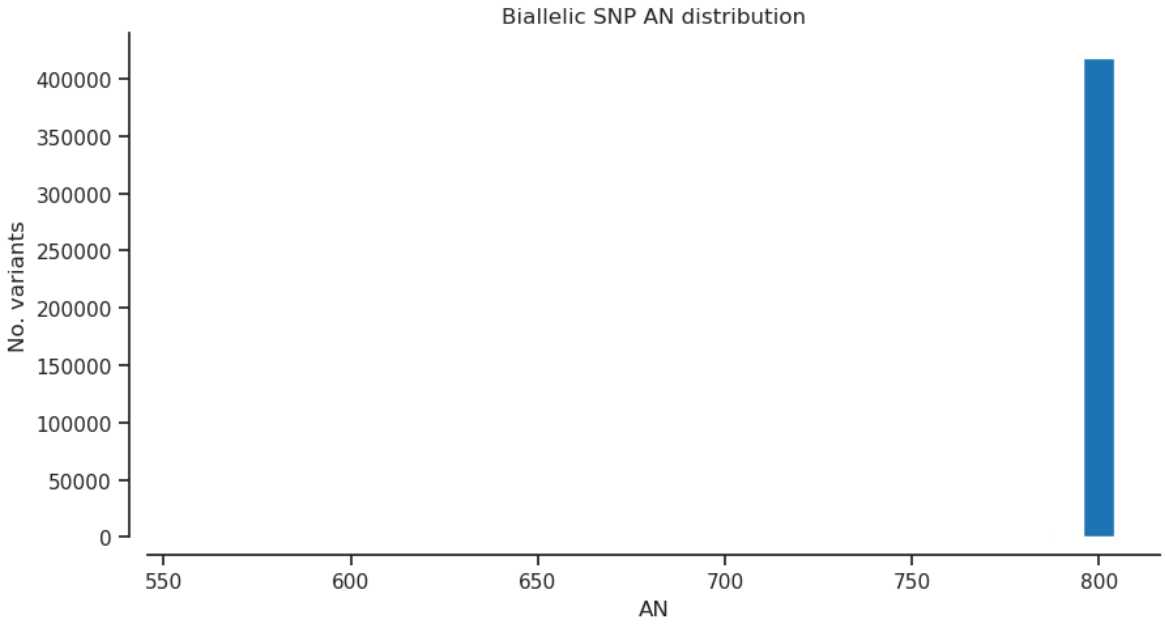


AN - Total number of alleles in called genotypes

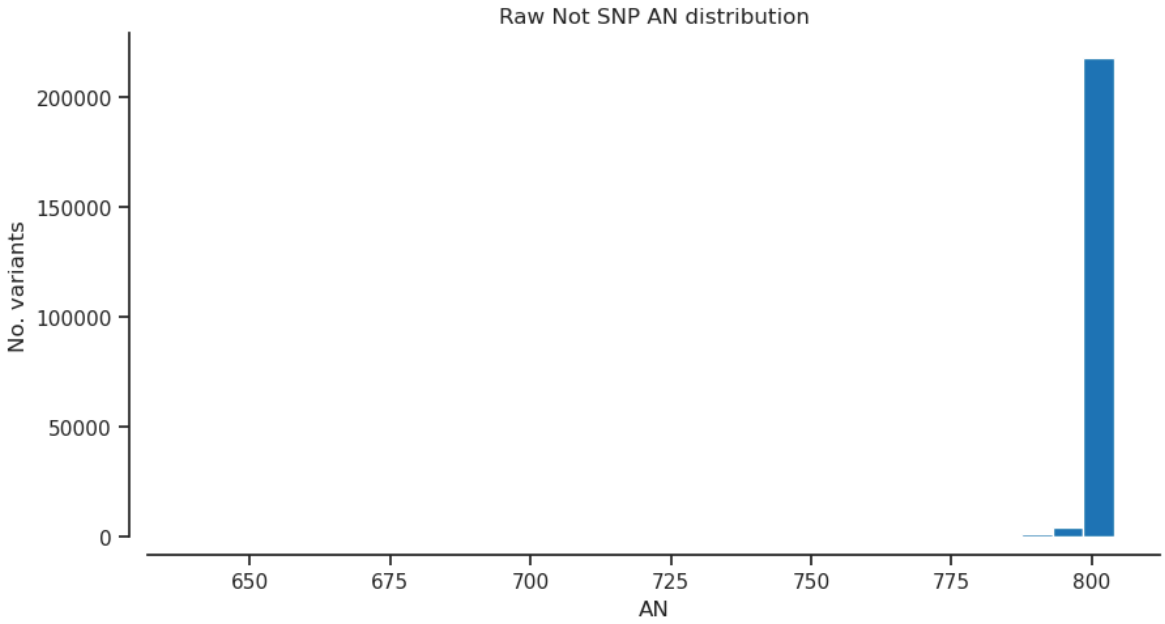
In [44]:
```
plot_hist('AN','var') # Total number of alleles in called genotypes
```



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]:
```
385809
```

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=(661793, 402, 2) dtype=int8 chunks=(65536, 64, 2) nbytes=507.4M cbytes=25.2M cratio=20.2 compression=gzip compression_opts=1 values=h5py._hl.dataset.Dataset>

	0	1	2	3	4	...	397	398	399	400	401
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
...						...					
661790	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
661791	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
661792	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=(385809, 402, 2) dtype=int8 chunks=(1508, 402, 2) nbytes=295.8M cbytes=26.7M cratio=11.1 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3	4	...	397	398	399	400	401
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
...						...					
385806	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
385807	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
385808	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=(385809, 4) dtype=int32 chunks=(24114, 4) nbytes=5.9M cbytes=1.1M cratio=5.4 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3
0	802	2	0	0
1	800	4	0	0
2	801	3	0	0
...		...		
385806	803	1	0	0
385807	758	46	0	0
385808	801	3	0	0

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

Out[52]: <AlleleCountsArray shape=(385809, 4) dtype=int32>

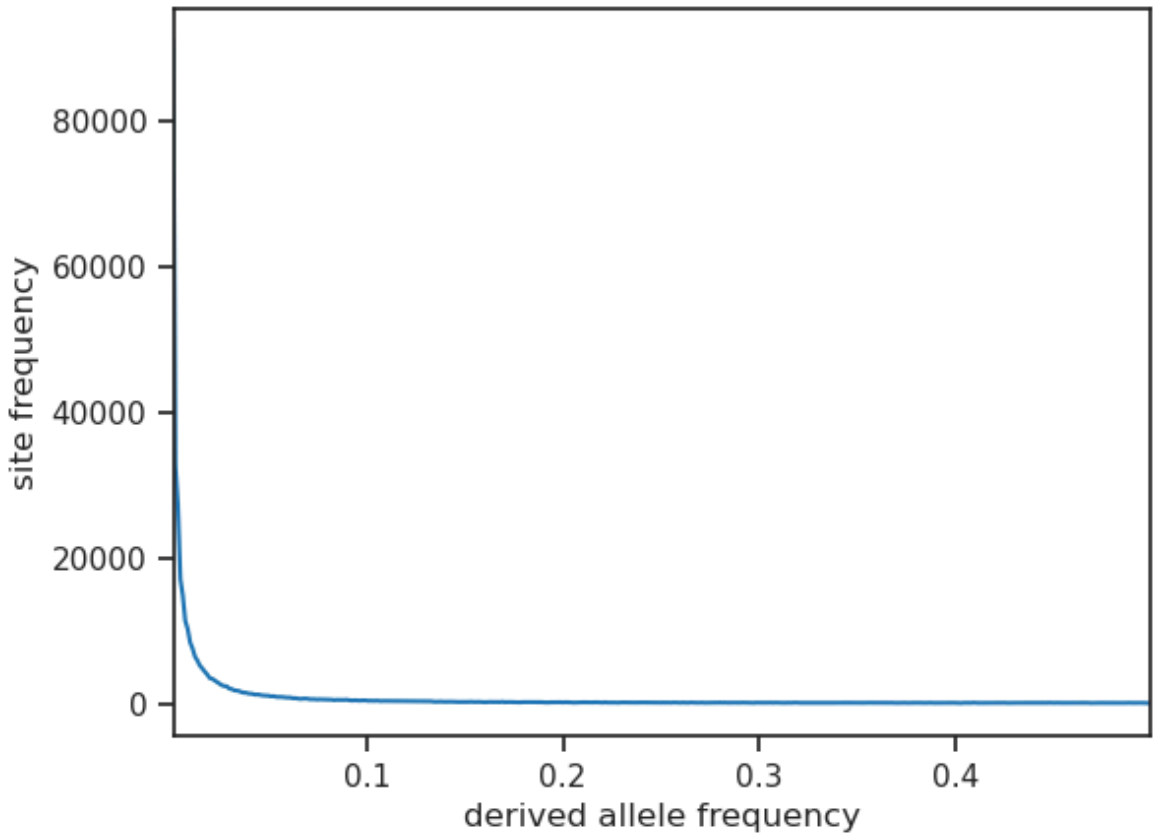
	0	1	2	3
0	802	2	0	0
1	800	4	0	0
2	801	3	0	0
...		...		
385806	803	1	0	0
385807	758	46	0	0
385808	801	3	0	0

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([[802,   2],
       [800,   4],
       [801,   3],
       ...,
       [803,   1],
       [758,  46],
       [801,   3]], 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=(385809,) dtype=bool chunks=(192905,)
   nbytes=376.8K cbytes=64.1K cratio=5.9
   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=(369185, 402, 2) dtype=int8 chunks=(1443, 402, 2) nbytes=283.1M cbytes=24.6M cratio=11.5 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3	4	...	397	398	399	400	401
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
...						...					
369182	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
369183	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
369184	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]: 369185

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'AUT00104-001',
 b'AUT00104-002',
 b'AUT00104-003',
 b'AUT00104-004',
 b'AUT00104-005',
 b'AUT00104-006',
 b'AUT00104-007',
 b'AUT00104-008',
 b'AUT00104-009',
 b'AUT00104-010',
 b'AUT00104-011',
 b'AUT00104-012',
 b'AUT00104-013',
 b'AUT00104-014',
 b'AUT00104-015',
 b'AUT00104-016',
 b'AUT00104-017',
 b'AUT00104-018',
 b'AUT00104-019',
 b'AUT00104-020',
 b'AUT00104-021',
 b'AUT00104-022',
 b'AUT00104-023',
 b'AUT00104-024',
 b'AUT00104-025',
 b'AUT00272-001',
 b'AUT00272-002',
 b'AUT00272-003',
 b'AUT00272-004',
 b'AUT00272-005',
 b'AUT00272-006',
 b'AUT00272-007',
 b'AUT00272-008',
 b'AUT00272-009',
 b'AUT00272-010',
 b'AUT00272-011',
 b'AUT00272-012',
 b'AUT00272-013',
 b'AUT00272-014',
 b'AUT00272-015',
 b'AUT00272-016',
 b'AUT00272-017',
 b'AUT00272-018',
 b'AUT00272-019',
 b'AUT00272-020',
 b'AUT00272-021',
 b'AUT00272-022',
 b'AUT00272-023',
 b'AUT00272-024',
 b'AUT00272-025',
 b'DEU00146-001',
 b'DEU00146-002',
 b'DEU00146-003',
 b'DEU00146-004',
 b'DEU00146-005',
 b'DEU00146-006',
 b'DEU00146-007',
 b'DEU00146-008',
 b'DEU00146-009',
 b'DEU00146-010',
```

```
b'DEU00146-011',
b'DEU00146-012',
b'DEU00146-013',
b'DEU00146-014',
b'DEU00146-015',
b'DEU00146-016',
b'DEU00146-017',
b'DEU00146-018',
b'DEU00146-019',
b'DEU00146-020',
b'DEU00146-021',
b'DEU00146-022',
b'DEU00146-023',
b'DEU00146-024',
b'DEU00146-025',
b'ESP00242-001',
b'ESP00242-002',
b'ESP00242-003',
b'ESP00242-004',
b'ESP00242-005',
b'ESP00242-006',
b'ESP00242-007',
b'ESP00242-008',
b'ESP00242-009',
b'ESP00242-010',
b'ESP00242-011',
b'ESP00242-012',
b'ESP00242-013',
b'ESP00242-014',
b'ESP00242-015',
b'ESP00242-016',
b'ESP00242-017',
b'ESP00242-018',
b'ESP00242-019',
b'ESP00242-020',
b'ESP00242-021',
b'ESP00242-022',
b'ESP00242-023',
b'ESP00242-024',
b'ESP00242-025',
b'FIN00042-001',
b'FIN00042-002',
b'FIN00042-003',
b'FIN00042-004',
b'FIN00042-005',
b'FIN00042-006',
b'FIN00042-007',
b'FIN00042-008',
b'FIN00042-009',
b'FIN00042-010',
b'FIN00042-011',
b'FIN00042-012',
b'FIN00042-013',
b'FIN00042-014',
b'FIN00042-015',
b'FIN00042-016',
b'FIN00042-017',
b'FIN00042-018',
b'FIN00042-019',
b'FIN00042-020',
```

```
b'FIN00042-021',
b'FIN00042-022',
b'FIN00042-023',
b'FIN00042-024',
b'FIN00042-025',
b'GBR00015-301',
b'GBR00015-302',
b'GBR00015-303',
b'GBR00015-304',
b'GBR00015-305',
b'GBR00015-306',
b'GBR00015-307',
b'GBR00015-308',
b'GBR00015-309',
b'GBR00015-310',
b'GBR00015-311',
b'GBR00015-312',
b'GBR00015-313',
b'GBR00015-314',
b'GBR00015-315',
b'GBR00015-316',
b'GBR00015-317',
b'GBR00015-318',
b'GBR00015-319',
b'GBR00015-320',
b'GBR00015-321',
b'GBR00015-322',
b'GBR00015-323',
b'GBR00015-324',
b'GBR00015-325',
b'ITA00011-001',
b'ITA00011-002',
b'ITA00011-003',
b'ITA00011-004',
b'ITA00011-005',
b'ITA00011-006',
b'ITA00011-008',
b'ITA00011-009',
b'ITA00011-011',
b'ITA00011-012',
b'ITA00011-013',
b'ITA00011-014',
b'ITA00011-015',
b'ITA00011-016',
b'ITA00011-017',
b'ITA00011-018',
b'ITA00011-022',
b'ITA00011-023',
b'ITA00011-024',
b'ITA00011-026',
b'ITA00011-027',
b'ITA00011-028',
b'ITA00011-029',
b'ITA00011-030',
b'ITA00011-031',
b'ITA00011-032',
b'ITA00045-201',
b'ITA00045-202',
b'ITA00045-203',
b'ITA00045-204',
```

```
b'ITA00045-205',
b'ITA00045-206',
b'ITA00045-207',
b'ITA00045-208',
b'ITA00045-209',
b'ITA00045-210',
b'ITA00045-211',
b'ITA00045-212',
b'ITA00045-213',
b'ITA00045-214',
b'ITA00045-215',
b'ITA00045-216',
b'ITA00045-217',
b'ITA00045-218',
b'ITA00045-219',
b'ITA00045-220',
b'ITA00045-221',
b'ITA00045-222',
b'ITA00045-223',
b'ITA00045-224',
b'ITA00045-225',
b'ITA00045-226',
b'ITA00105-201',
b'ITA00105-202',
b'ITA00105-203',
b'ITA00105-204',
b'ITA00105-205',
b'ITA00105-206',
b'ITA00105-207',
b'ITA00105-208',
b'ITA00105-209',
b'ITA00105-210',
b'ITA00105-211',
b'ITA00105-212',
b'ITA00105-213',
b'ITA00105-214',
b'ITA00105-215',
b'ITA00105-216',
b'ITA00105-217',
b'ITA00105-218',
b'ITA00105-219',
b'ITA00105-220',
b'ITA00105-221',
b'ITA00105-222',
b'ITA00105-223',
b'ITA00105-224',
b'ITA00258-002',
b'ITA00258-003',
b'ITA00258-004',
b'ITA00258-006',
b'ITA00258-009',
b'ITA00258-012',
b'ITA00258-014',
b'ITA00258-015',
b'ITA00258-016',
b'ITA00258-017',
b'ITA00258-028',
b'ITA00258-031',
b'ITA00258-033',
b'ITA00258-034',
```

```
b'ITA00258-035',
b'ITA00258-036',
b'ITA00258-038',
b'ITA00258-042',
b'ITA00258-044',
b'ITA00258-045',
b'ITA00258-047',
b'ITA00258-050',
b'ITA00258-051',
b'ITA00258-057',
b'ITA00258-062',
b'ITA00258-065',
b'ROU00013-001',
b'ROU00013-002',
b'ROU00013-003',
b'ROU00013-004',
b'ROU00013-005',
b'ROU00013-006',
b'ROU00013-007',
b'ROU00013-008',
b'ROU00013-009',
b'ROU00013-010',
b'ROU00013-011',
b'ROU00013-012',
b'ROU00013-013',
b'ROU00013-014',
b'ROU00013-015',
b'ROU00013-016',
b'ROU00013-017',
b'ROU00013-018',
b'ROU00013-019',
b'ROU00013-020',
b'ROU00013-021',
b'ROU00013-022',
b'ROU00013-023',
b'ROU00013-024',
b'ROU00013-025',
b'ROU00137-001',
b'ROU00137-002',
b'ROU00137-003',
b'ROU00137-004',
b'ROU00137-005',
b'ROU00137-006',
b'ROU00137-007',
b'ROU00137-008',
b'ROU00137-009',
b'ROU00137-010',
b'ROU00137-011',
b'ROU00137-012',
b'ROU00137-013',
b'ROU00137-014',
b'ROU00137-015',
b'ROU00137-016',
b'ROU00137-017',
b'ROU00137-018',
b'ROU00137-019',
b'ROU00137-020',
b'ROU00137-021',
b'ROU00137-022',
b'ROU00137-023',
```

```
b'ROU00137-024',
b'ROU00137-025',
b'ROU00181-001',
b'ROU00181-002',
b'ROU00181-003',
b'ROU00181-004',
b'ROU00181-005',
b'ROU00181-006',
b'ROU00181-007',
b'ROU00181-008',
b'ROU00181-009',
b'ROU00181-010',
b'ROU00181-011',
b'ROU00181-012',
b'ROU00181-013',
b'ROU00181-014',
b'ROU00181-015',
b'ROU00181-016',
b'ROU00181-017',
b'ROU00181-018',
b'ROU00181-019',
b'ROU00181-020',
b'ROU00181-021',
b'ROU00181-022',
b'ROU00181-023',
b'ROU00181-024',
b'ROU00181-025',
b'ROU00269-001',
b'ROU00269-002',
b'ROU00269-003',
b'ROU00269-004',
b'ROU00269-005',
b'ROU00269-006',
b'ROU00269-007',
b'ROU00269-008',
b'ROU00269-009',
b'ROU00269-010',
b'ROU00269-011',
b'ROU00269-012',
b'ROU00269-013',
b'ROU00269-014',
b'ROU00269-015',
b'ROU00269-016',
b'ROU00269-017',
b'ROU00269-018',
b'ROU00269-019',
b'ROU00269-020',
b'ROU00269-021',
b'ROU00269-022',
b'ROU00269-023',
b'ROU00269-024',
b'ROU00269-025',
b'SVN00009-001',
b'SVN00009-002',
b'SVN00009-003',
b'SVN00009-004',
b'SVN00009-005',
b'SVN00009-006',
b'SVN00009-007',
b'SVN00009-008',
```

```
 b'SVN00009-009',
 b'SVN00009-010',
 b'SVN00009-011',
 b'SVN00009-012',
 b'SVN00009-013',
 b'SVN00009-014',
 b'SVN00009-015',
 b'SVN00009-016',
 b'SVN00009-017',
 b'SVN00009-018',
 b'SVN00009-019',
 b'SVN00009-020',
 b'SVN00009-021',
 b'SVN00009-022',
 b'SVN00009-023',
 b'SVN00009-024',
 b'SVN00009-025',
 b'SVN00011-001',
 b'SVN00011-002',
 b'SVN00011-003',
 b'SVN00011-004',
 b'SVN00011-005',
 b'SVN00011-006',
 b'SVN00011-007',
 b'SVN00011-008',
 b'SVN00011-009',
 b'SVN00011-010',
 b'SVN00011-011',
 b'SVN00011-012',
 b'SVN00011-013',
 b'SVN00011-014',
 b'SVN00011-015',
 b'SVN00011-016',
 b'SVN00011-017',
 b'SVN00011-018',
 b'SVN00011-019',
 b'SVN00011-020',
 b'SVN00011-021',
 b'SVN00011-022',
 b'SVN00011-023',
 b'SVN00011-024',
 b'SVN00011-025']
```

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

Out[60]:

	ID	Population
0	AUT00104-001	AUT00104
1	AUT00104-002	AUT00104
2	AUT00104-003	AUT00104
3	AUT00104-004	AUT00104
4	AUT00104-005	AUT00104
...	...	...
397	SVN00011-021	SVN00011
398	SVN00011-022	SVN00011
399	SVN00011-023	SVN00011
400	SVN00011-024	SVN00011
401	SVN00011-025	SVN00011

402 rows × 2 columns

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

Out[61]:
```
Population
ITA00011    26
ITA00045    26
ITA00258    26
AUT00104    25
ESP00242    25
DEU00146    25
GBR00015    25
AUT00272    25
FIN00042    25
ROU00013    25
ROU00137    25
ROU00181    25
SVN00009    25
ROU00269    25
SVN00011    25
ITA00105    24
Name: count, dtype: int64
```

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

Out[62]:
```
array(['AUT00104', 'AUT00272', 'DEU00146', 'ESP00242', 'FIN00042',
       'GBR00015', 'ITA00011', 'ITA00045', 'ITA00105', 'ITA00258',
       'ROU00013', 'ROU00137', 'ROU00181', 'ROU00269', 'SVN00009',
       'SVN00011'], dtype=object)
```

Gt frequency function

In [64]:
```
def plot_genotype_frequency(pc, title):
    fig, ax = plt.subplots(figsize=(24, 5))
```

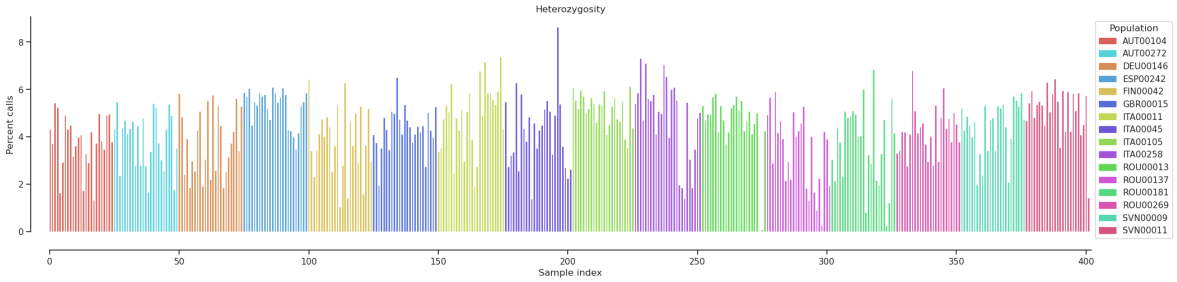
```
    sns.despine(ax=ax, offset=24)
    left = np.arange(len(pc))
    palette = sns.color_palette("hls", 16)
    pop2color = {'AUT00104': palette[0],
                 'AUT00272': palette[8],
                 'DEU00146': palette[1],
                 'ESP00242': palette[9],
                 'FIN00042': palette[2],
                 'GBR00015': palette[10],
                 'ITA00011': palette[3],
                 'ITA00045': palette[11],
                 'ITA00105': palette[4],
                 'ITA00258': palette[12],
                 'ROU00013': palette[5],
                 'ROU00137': palette[13],
                 'ROU00181': palette[6],
                 'ROU00269': palette[14],
                 'SVN00009': palette[7],
                 'SVN00011': 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=['AUT00104', 'AUT00272', 'DEU00146'
       'GBR00015', 'ITA00011', 'ITA00045', 'ITA00105', 'ITA00258',
       'ROU00013', 'ROU00137', 'ROU00181', 'ROU00269', 'SVN00009',
       'SVN00011'], title='Population',
              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 [68]:
```
palette = sns.color_palette("hls",16)
pop_colours = {
                'AUT00104': palette[0],
                 'AUT00272': palette[8],
                 'DEU00146': palette[1],
                 'ESP00242': palette[9],
                 'FIN00042': palette[2],
                 'GBR00015': palette[10],
                 'ITA00011': palette[3],
                 'ITA00045': palette[11],
                 'ITA00105': palette[4],
                 'ITA00258': palette[12],
                 'ROU00013': palette[5],
                 'ROU00137': palette[13],
                 'ROU00181': palette[6],
                 'ROU00269': palette[14],
                 'SVN00009': palette[7],
                 'SVN00011': palette[15]
}
```

In [69]:
```
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 [70]:
```
ac2 = gt_biallelic.count_alleles()
ac2
```

Out[70]: <AlleleCountsChunkedArray shape=(369185, 2) dtype=int32 chunks=(46149, 2) nbytes=2.8M cbytes=848.5K cratio=3.4 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1
0	802	2
1	800	4
2	801	3
...	...	
369182	803	1
369183	758	46
369184	801	3

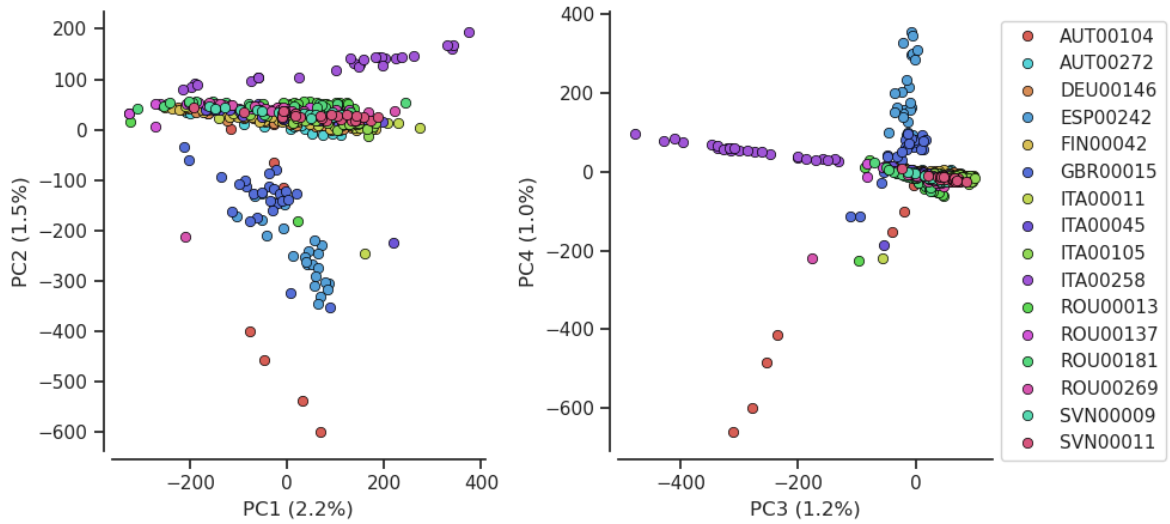
In [71]:
```
flt = (ac2[:, :2].min(axis=1) > 1)
gf = gt_biallelic.compress(flt, axis=0)
gn = gf.to_n_alt()
gn
```

Out[71]:
```
<ChunkedArrayWrapper shape=(278230, 402) dtype=int8 chunks=(2174, 402)
   nbytes=106.7M cbytes=16.8M cratio=6.4
   compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shu
ffle': 1, 'blocksize': 0}
   values=zarr.core.Array>
```

In [72]:
```
coords1, model1 = allel.pca(gn, n_components=10, scaler='patterson')
```

In [73]:
```
fig_pca(coords1, model1, 'Figure 1. Conventional PCA.')
```

Figure 1. Conventional PCA.



In [ ]: