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/Qpetraea/
```

Get data

In [3]:
```
callset_var_fn = '/users/mcevoysu/scratch/output/scikit-allel/Qpetraea/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=(859843,) 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=146.8M cbytes=32.6M cratio=4.5 values=h5py._hl.group.Group>

	AC	AF	ALT	AN	BaseQRankSum	CHROM	DP
0	[6 -1 -1]	[0.009036 nan nan]	[b'A' b'' b'']	658	nan	b'Qrob_Chr01'	32
1	[2 -1 -1]	[0.003012 nan nan]	[b'T' b'' b'']	658	nan	b'Qrob_Chr01'	99
2	[2 -1 -1]	[0.003012 nan nan]	[b'G' b'' b'']	658	nan	b'Qrob_Chr01'	111
...							
859840	[2 -1 -1]	[0.003012 nan nan]	[b'A' b'' b'']	658	nan	b'Qrob_H2.3_Sc0001194'	42
859841	[2 -1 -1]	[0.003012 nan nan]	[b'G' b'' b'']	658	nan	b'Qrob_H2.3_Sc0001194'	149
859842	[1 -1 -1]	[0.001506 nan nan]	[b'C' b'' b'']	658	0.0	b'Qrob_H2.3_Sc0001194'	152

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

Out[7]: <VariantTable shape=(568516,) 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	[6 -1 -1]	[0.009036 nan nan]	[b'A' b'' b'']	658	nan	b'Qrob_Chr01'	32
1	[2 -1 -1]	[0.003012 nan nan]	[b'T' b'' b'']	658	nan	b'Qrob_Chr01'	99
2	[2 -1 -1]	[0.003012 nan nan]	[b'G' b'' b'']	658	nan	b'Qrob_Chr01'	111
...							
568513	[2 -1 -1]	[0.003012 nan nan]	[b'A' b'' b'']	658	nan	b'Qrob_H2.3_Sc0001194'	42
568514	[2 -1 -1]	[0.003012 nan nan]	[b'G' b'' b'']	658	nan	b'Qrob_H2.3_Sc0001194'	149
568515	[1 -1 -1]	[0.001506 nan nan]	[b'C' b'' b'']	658	0.0	b'Qrob_H2.3_Sc0001194'	152

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

Out[8]: <VariantTable shape=(291327,) 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	[5 -1 -1]	[0.00753 nan nan]	[b'*' b'' b'']	658	nan	b'Qrob_Chr01'	336
1	[76 -1 -1]	[0.115 nan nan]	[b'*' b'' b'']	656	nan	b'Qrob_Chr01'	334
2	[67 -1 -1]	[0.101 nan nan]	[b'*' b'' b'']	658	nan	b'Qrob_Chr01'	334
...							
291324	[2 -1 -1]	[0.003012 nan nan]	[b'*' b'' b'']	658	-0.862	b'Qrob_H2.3_Sc0001028'	650
291325	[551 -1 -1]	[0.841 nan nan]	[b'*' b'' b'']	656	nan	b'Qrob_H2.3_Sc0001028'	651
291326	[1 -1 -1]	[0.001506 nan nan]	[b'*' b'' b'']	658	nan	b'Qrob_H2.3_Sc0001163'	752

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, 539273,  28455,    788])
```

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

Out[12]:
```
29243
```

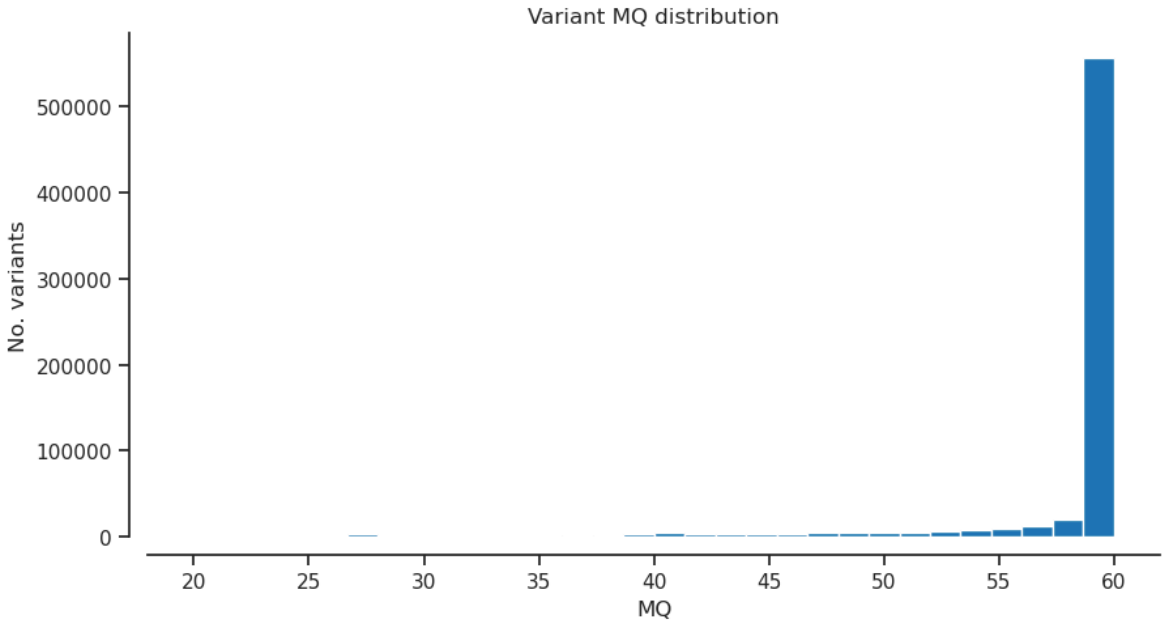
In [13]:
```
filter_expression = '(numalt == 1)'
biallelic_np = rawsnps.query(filter_expression)[:]
biallelic_np
```

Out[13]: <VariantTable shape=(539273,) 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')])>

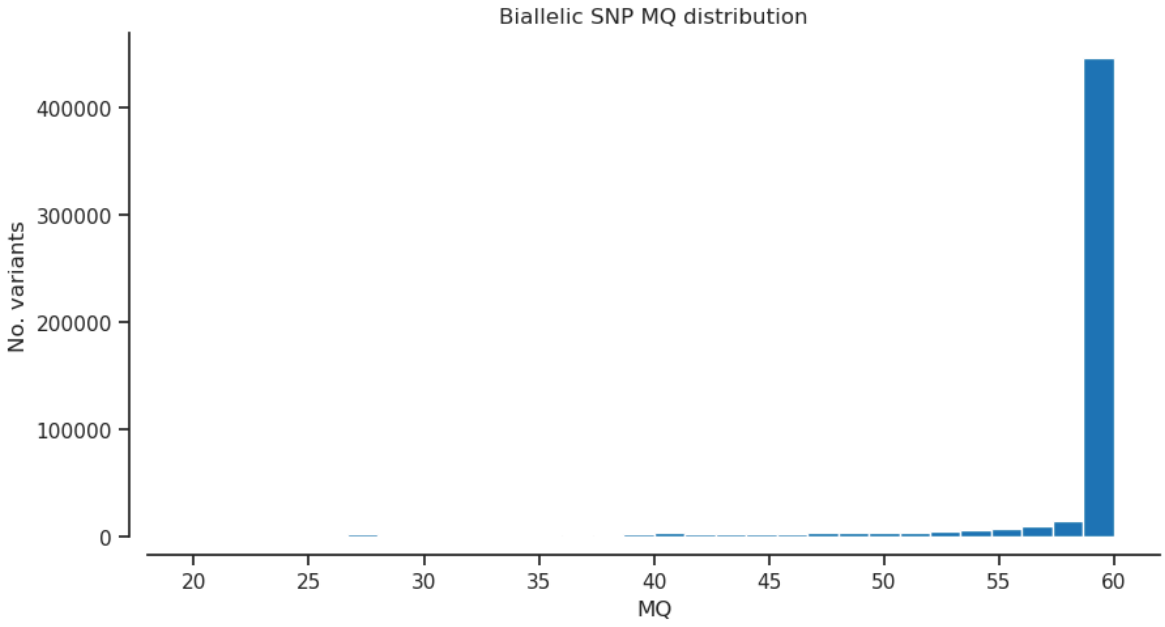
	AC	AF	ALT	AN	BaseQRankSum	CHROM	DP
0	[6 -1 -1]	[0.009036 nan nan]	[b'A' b'' b'']	658	nan	b'Qrob_Chr01'	32
1	[2 -1 -1]	[0.003012 nan nan]	[b'T' b'' b'']	658	nan	b'Qrob_Chr01'	99
2	[2 -1 -1]	[0.003012 nan nan]	[b'G' b'' b'']	658	nan	b'Qrob_Chr01'	111
...							
539270	[2 -1 -1]	[0.003012 nan nan]	[b'A' b'' b'']	658	nan	b'Qrob_H2.3_Sc0001194'	42
539271	[2 -1 -1]	[0.003012 nan nan]	[b'G' b'' b'']	658	nan	b'Qrob_H2.3_Sc0001194'	149
539272	[1 -1 -1]	[0.001506 nan nan]	[b'C' b'' b'']	658	0.0	b'Qrob_H2.3_Sc0001194'	152

MQ - RMS mapping quality

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

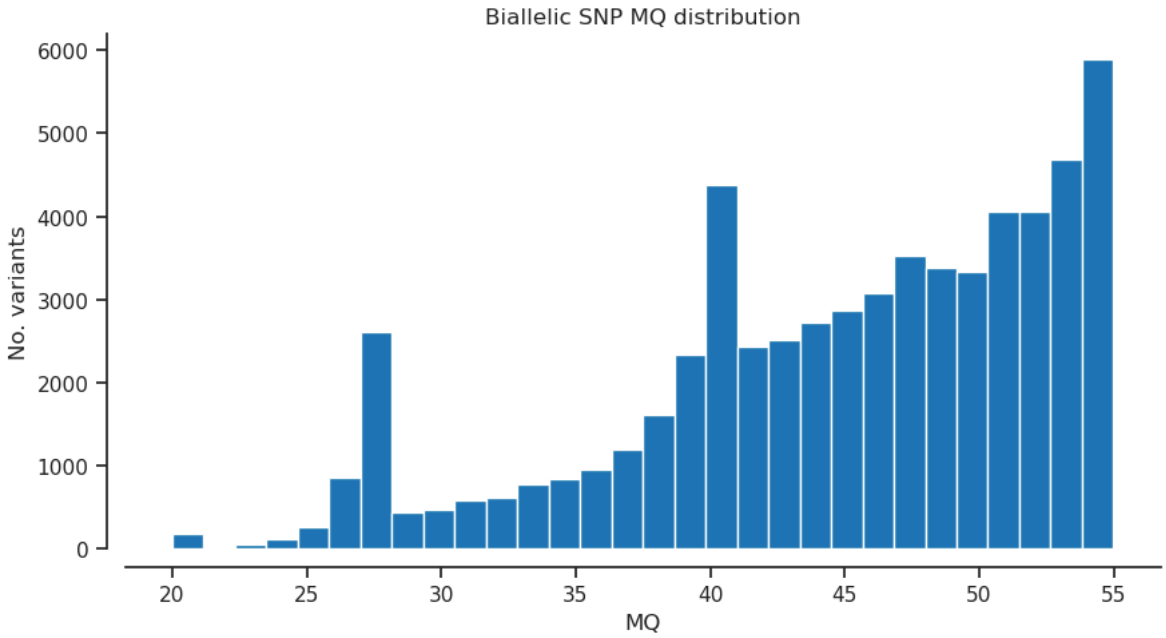


In [15]:
```
plot_hist('MQ','biallelic') # RMS mapping quality
```



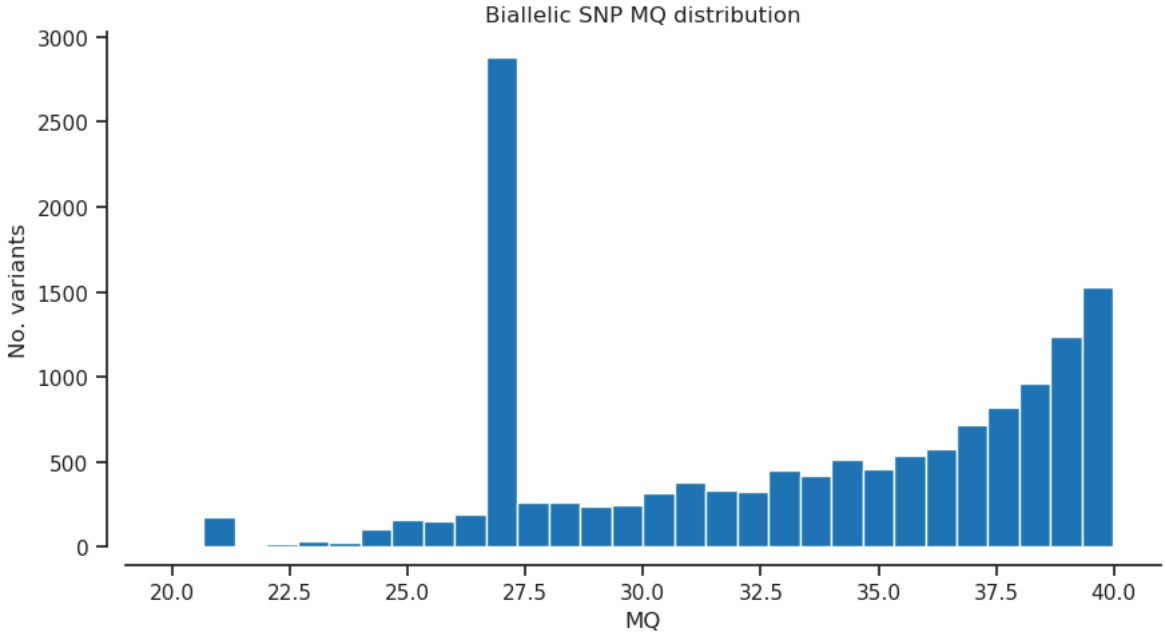
In [16]:
```
filter_expression = '(MQ < 55)'
bi_selection = biallelic_np.query(filter_expression)[:]
#np.count_nonzero(var_selection)
```

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

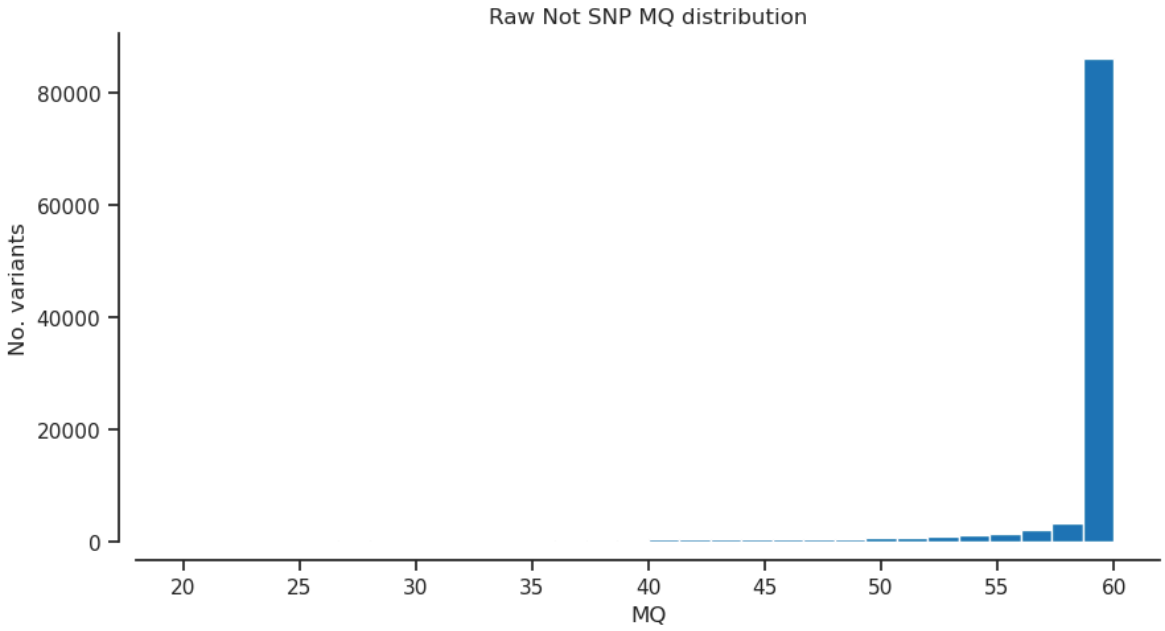


In [18]:
```
filter_expression = '(MQ < 40)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

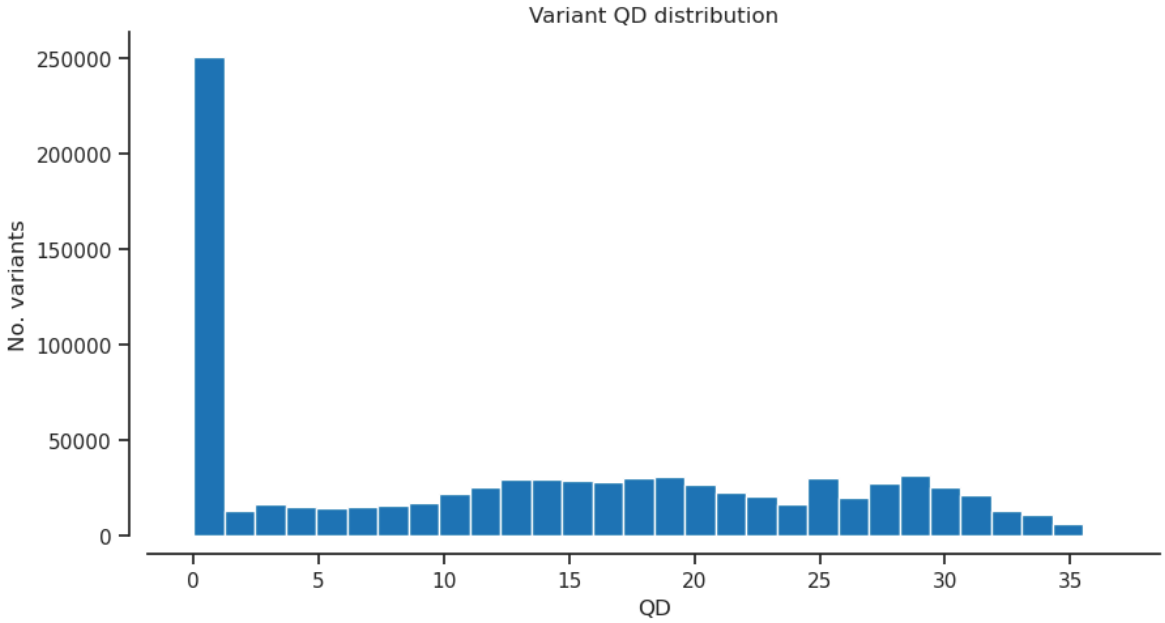


In [20]:
```
plot_hist('MQ','notsnp')
```

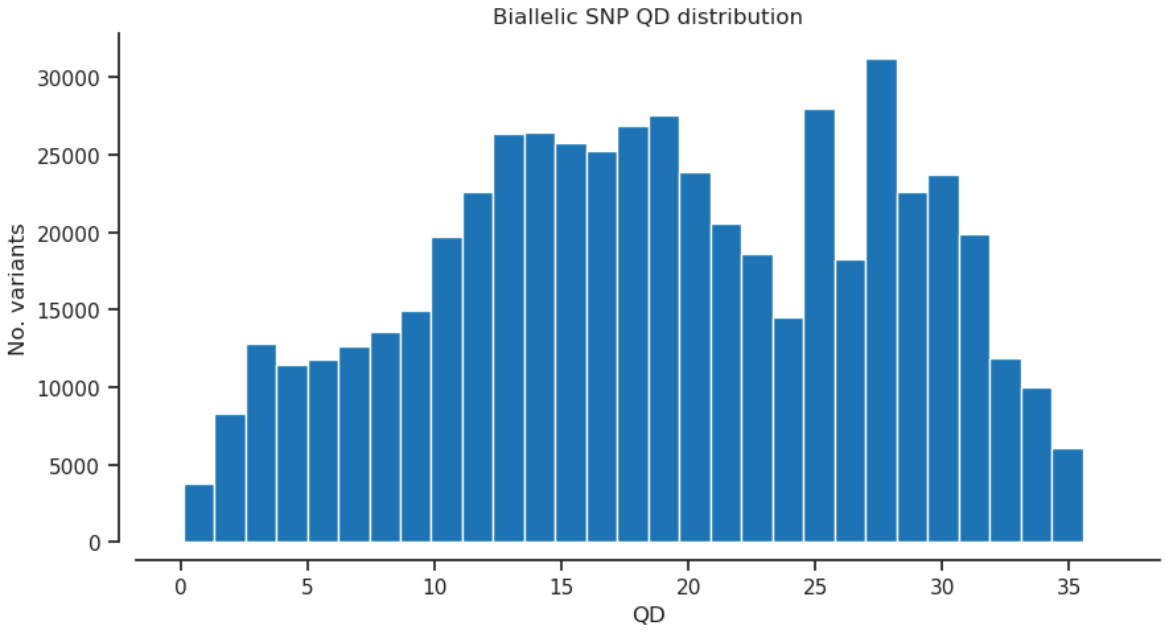


QD - Variant Confidence/Quality by Depth

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

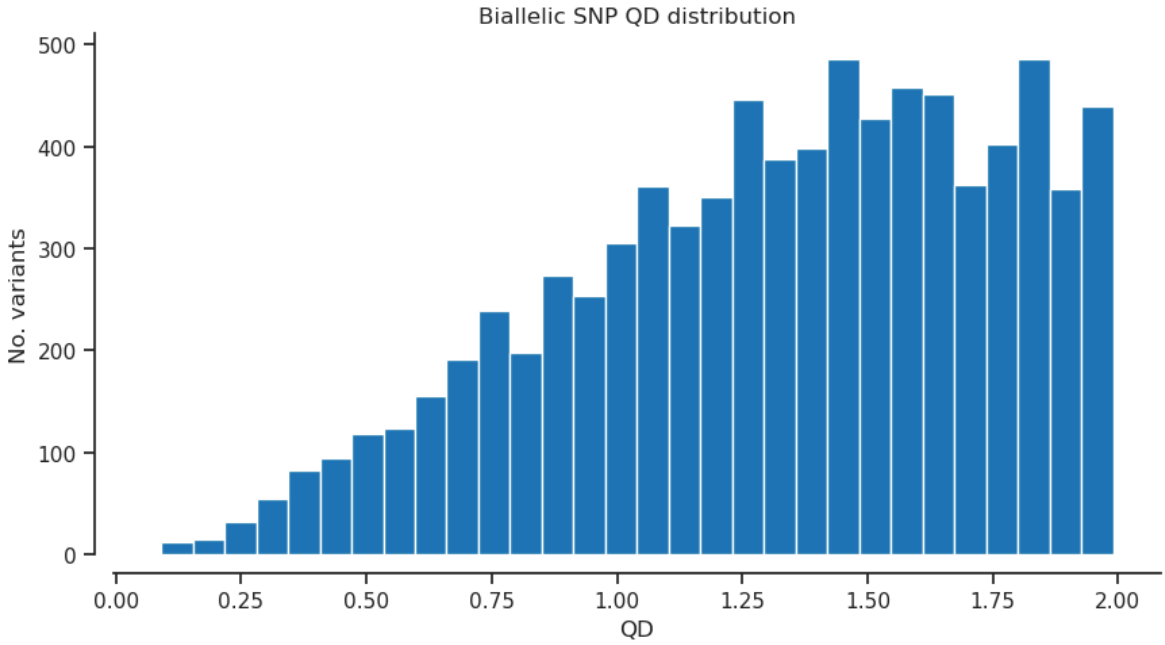


In [22]:
```
plot_hist('QD','biallelic') # Variant Confidence/Quality by Depth
```

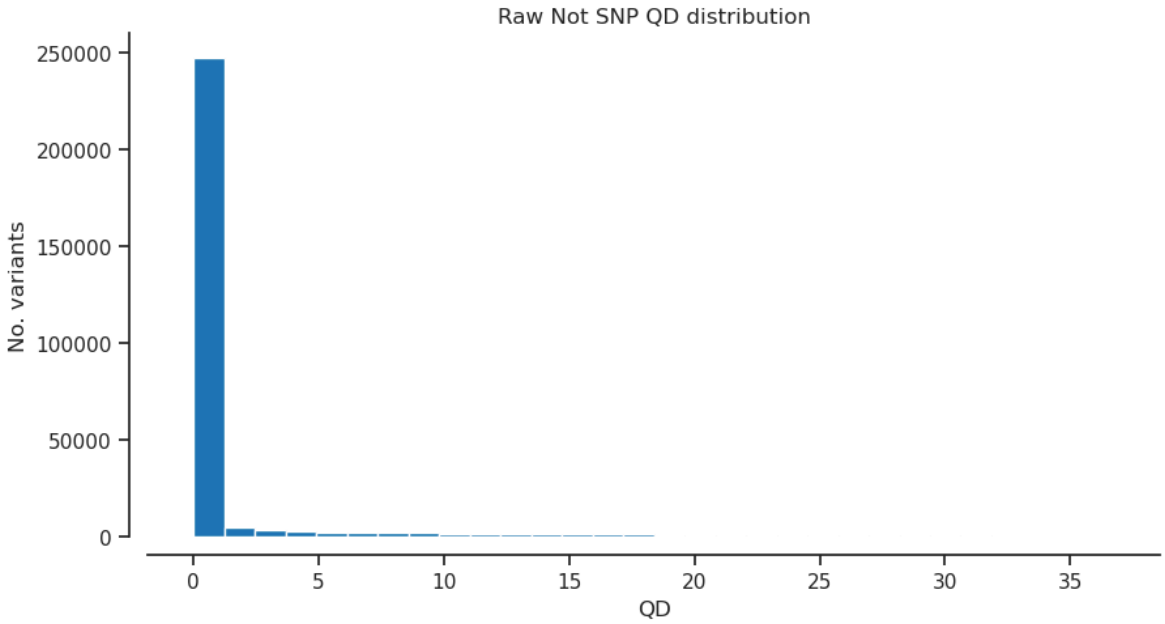


In [23]:
```
filter_expression = '(QD < 2)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

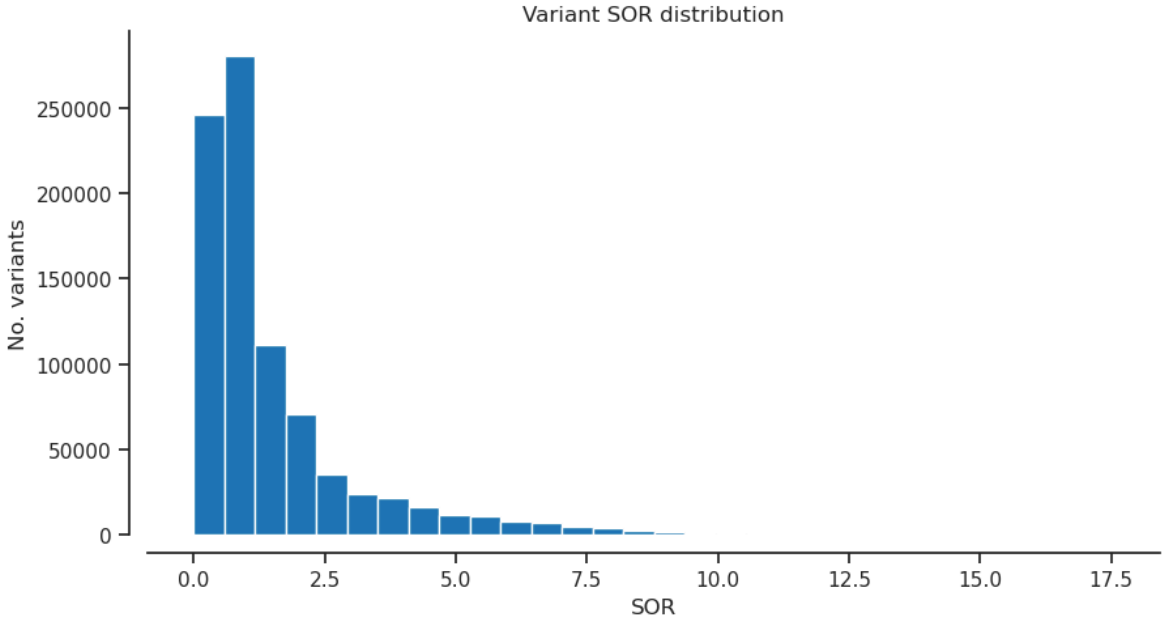


In [25]:
```
plot_hist('QD','notsnp') # Variant Confidence/Quality by Depth
```

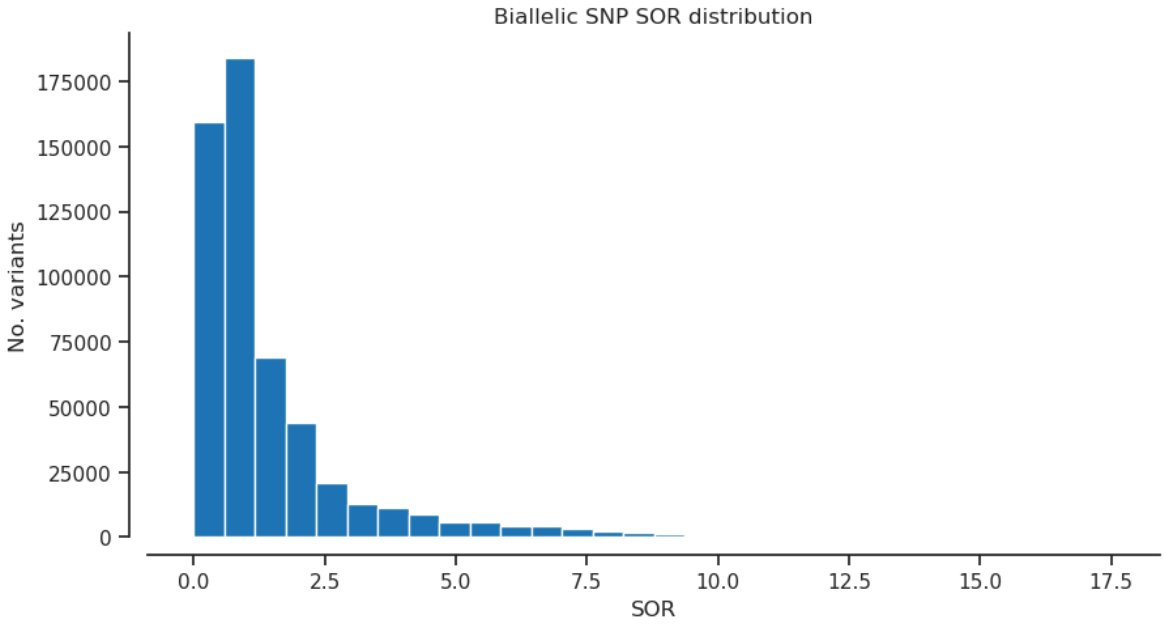


SOR - Symmetric Odds Ratio of 2x2 contingency table to detect strand bias

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

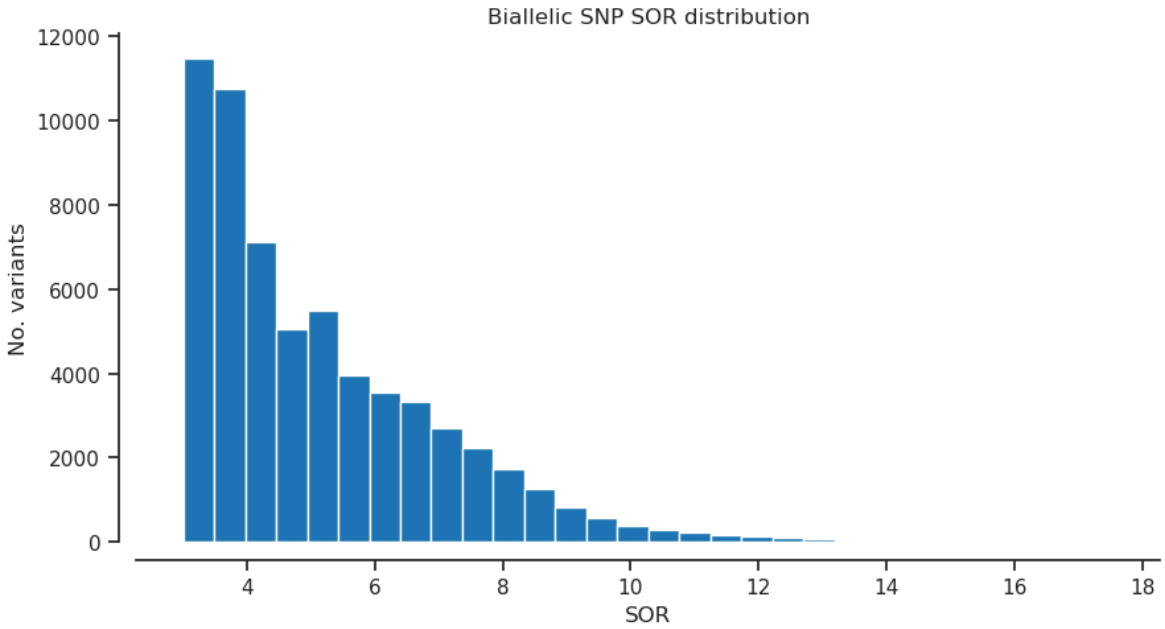


In [27]:
```
plot_hist('SOR','biallelic') # Symmetric Odds Ratio of 2x2 contingency ta
```

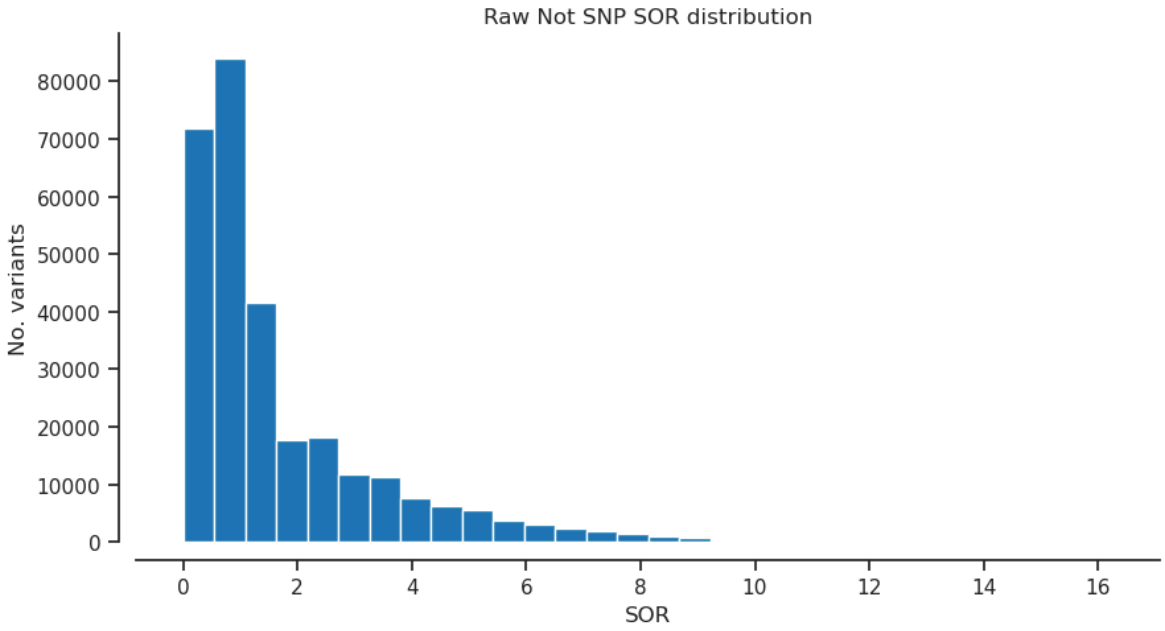


In [28]:
```
filter_expression = '(SOR > 3)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

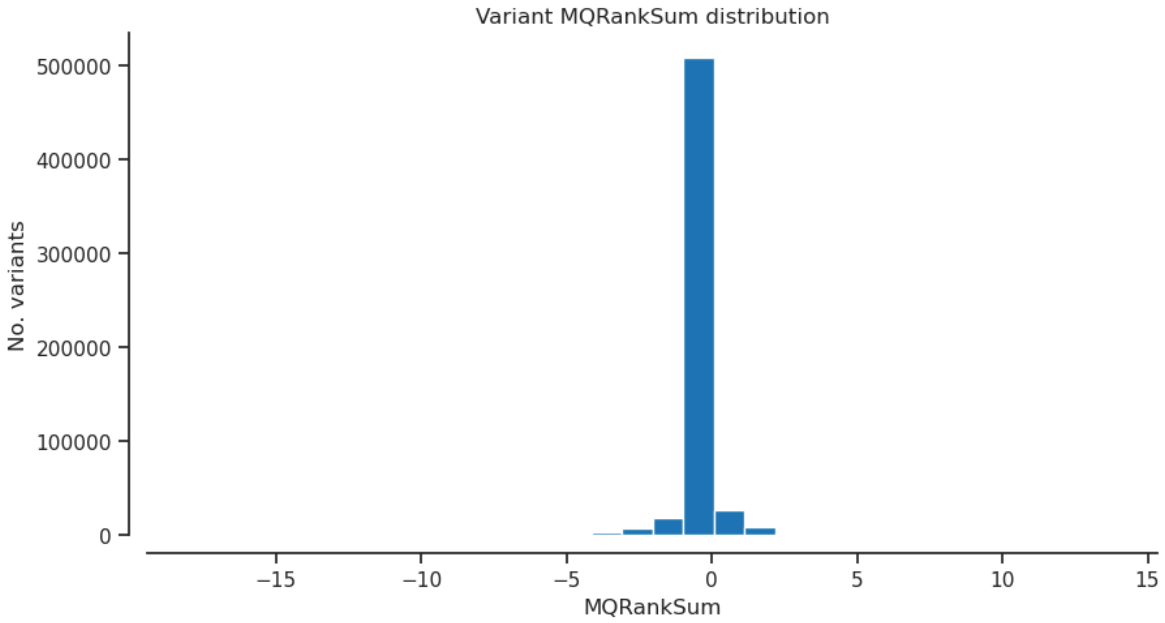


In [30]:
```
plot_hist('SOR','notsnp') # Symmetric Odds Ratio of 2x2 contingency table
```

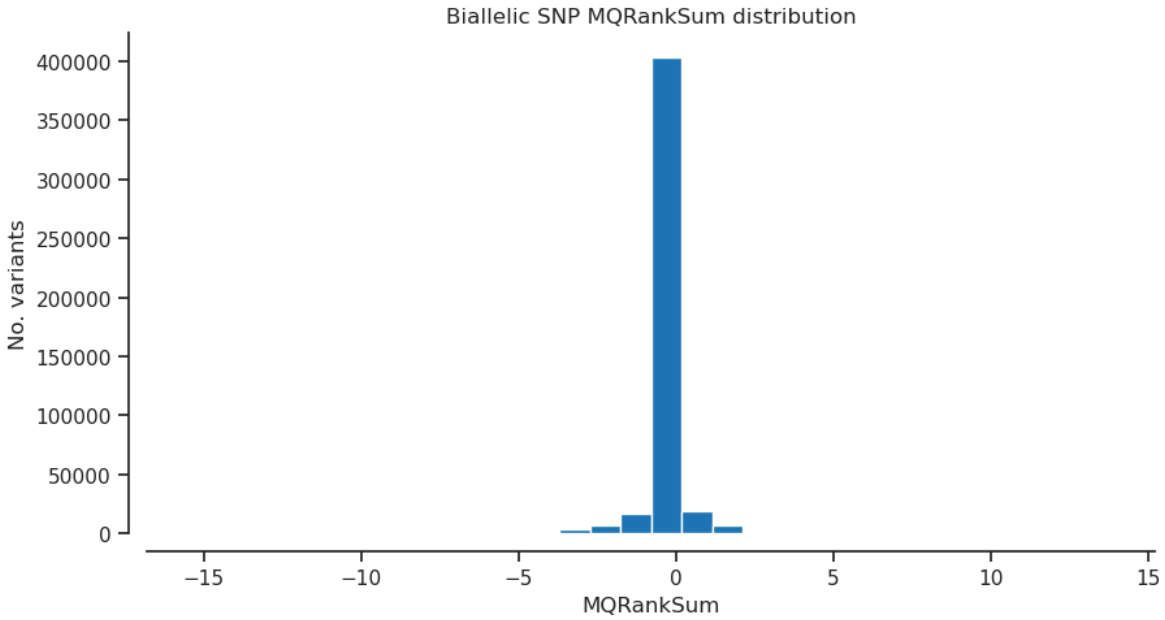


MQRankSum - Z-score From Wilcoxon rank sum test of Alt vs. Ref read mapping qualities

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

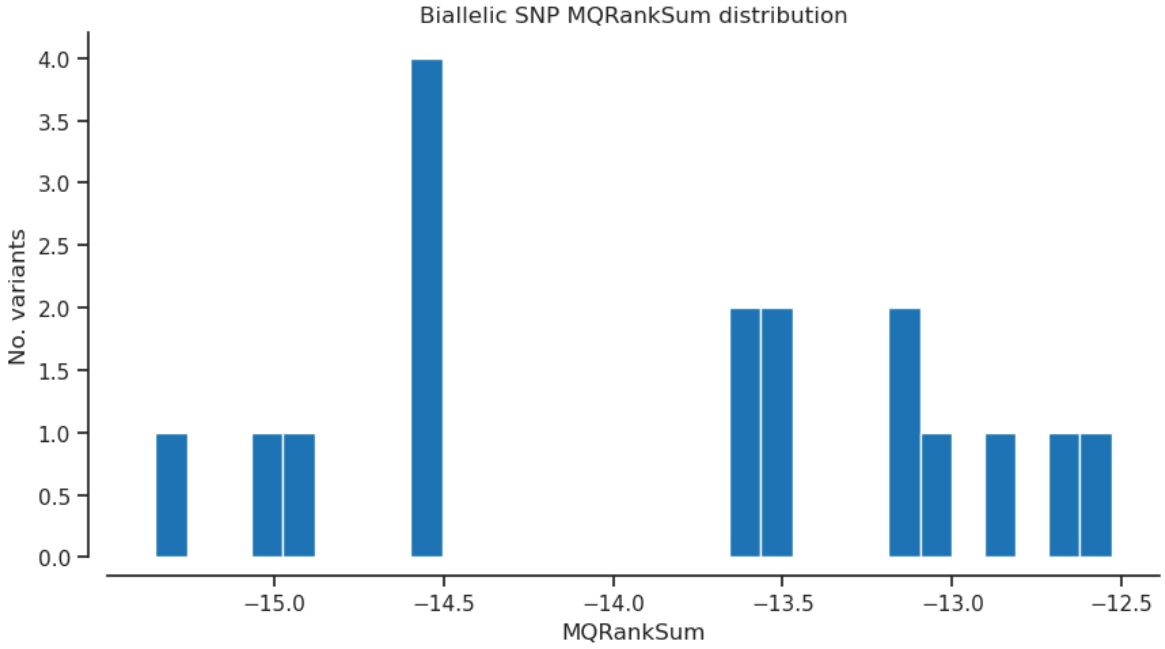


In [32]:
```
plot_hist('MQRankSum','biallelic') # Z-score From Wilcoxon rank sum test
```

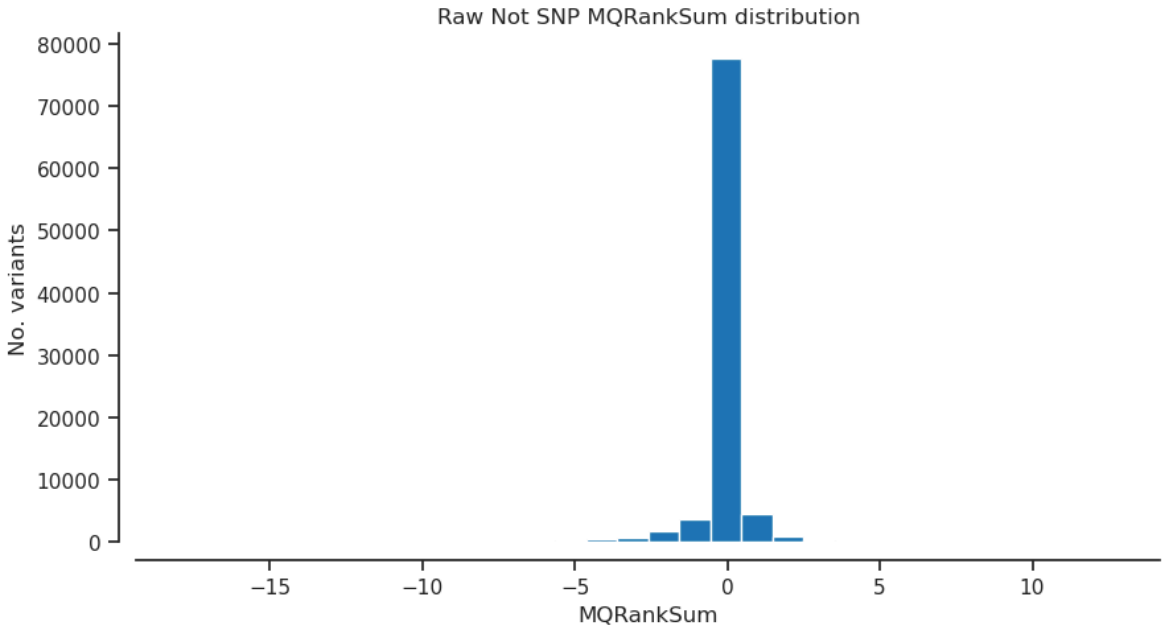


In [33]:
```
filter_expression = '(MQRankSum < -12.5)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

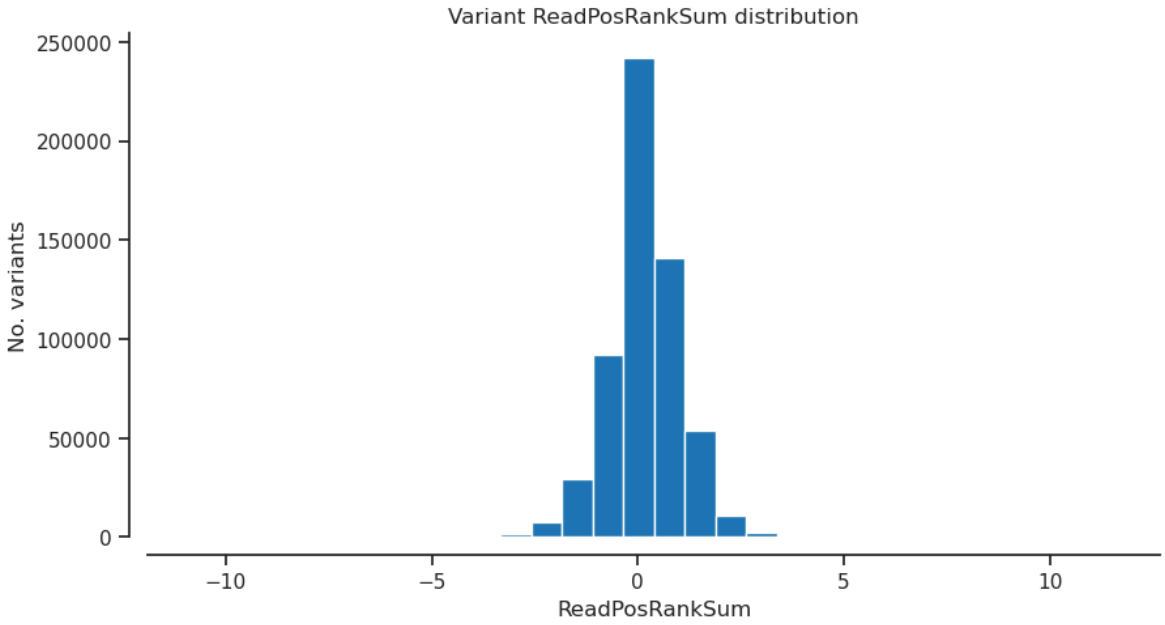


In [35]:
```
plot_hist('MQRankSum','notsnp') # Z-score From Wilcoxon rank sum test of
```

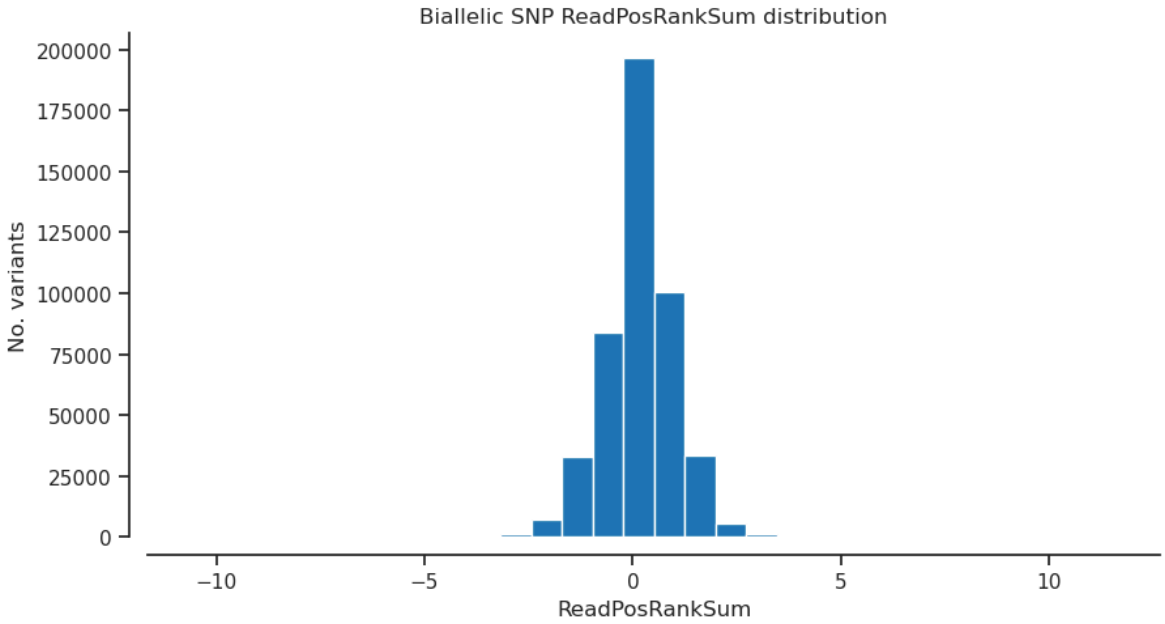


ReadPosRankSum - Z-score from Wilcoxon rank sum test of Alt vs. Ref read position bias

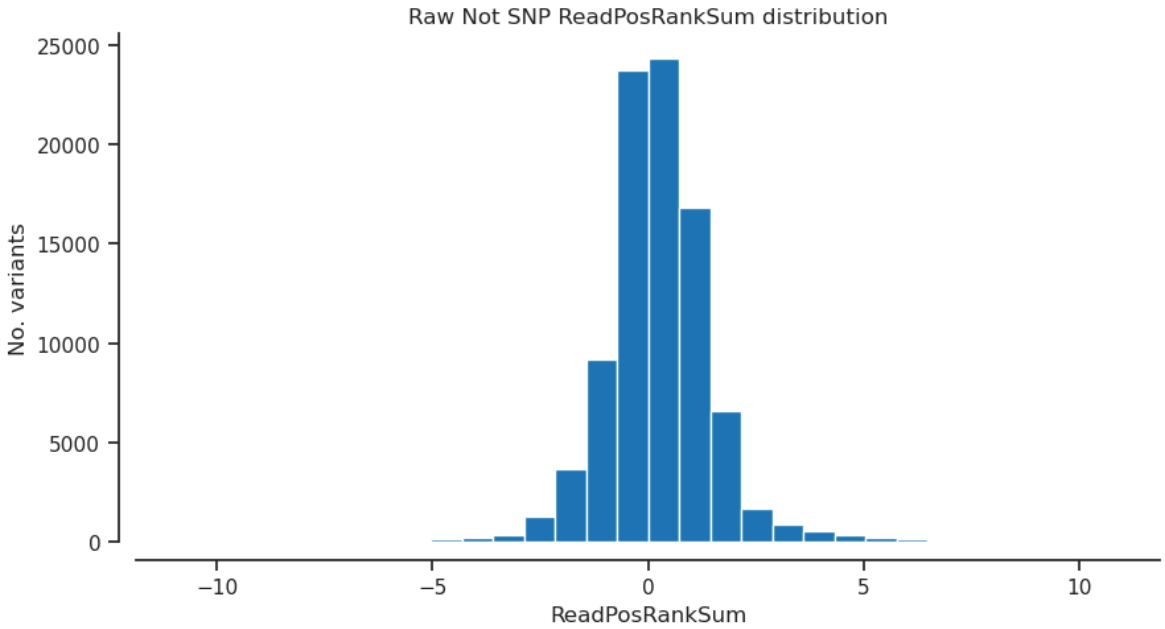
In [36]:
```
plot_hist('ReadPosRankSum','var') # Z-score from Wilcoxon rank sum test o
```



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

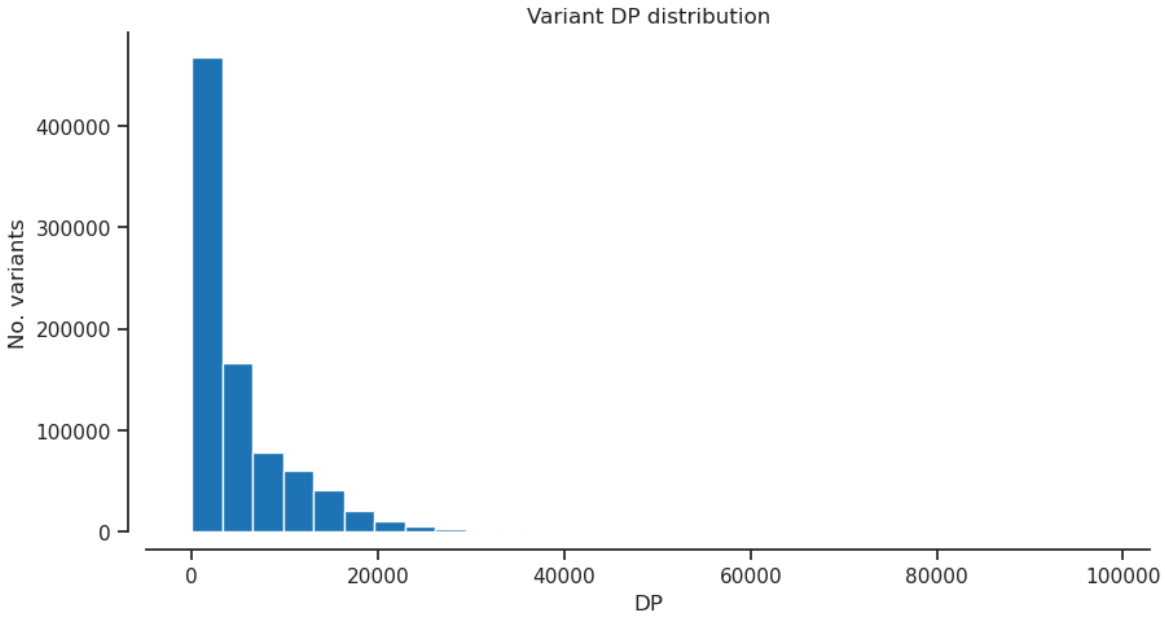


In [38]:
```
plot_hist('ReadPosRankSum','notsnp') # Z-score from Wilcoxon rank sum tes
```

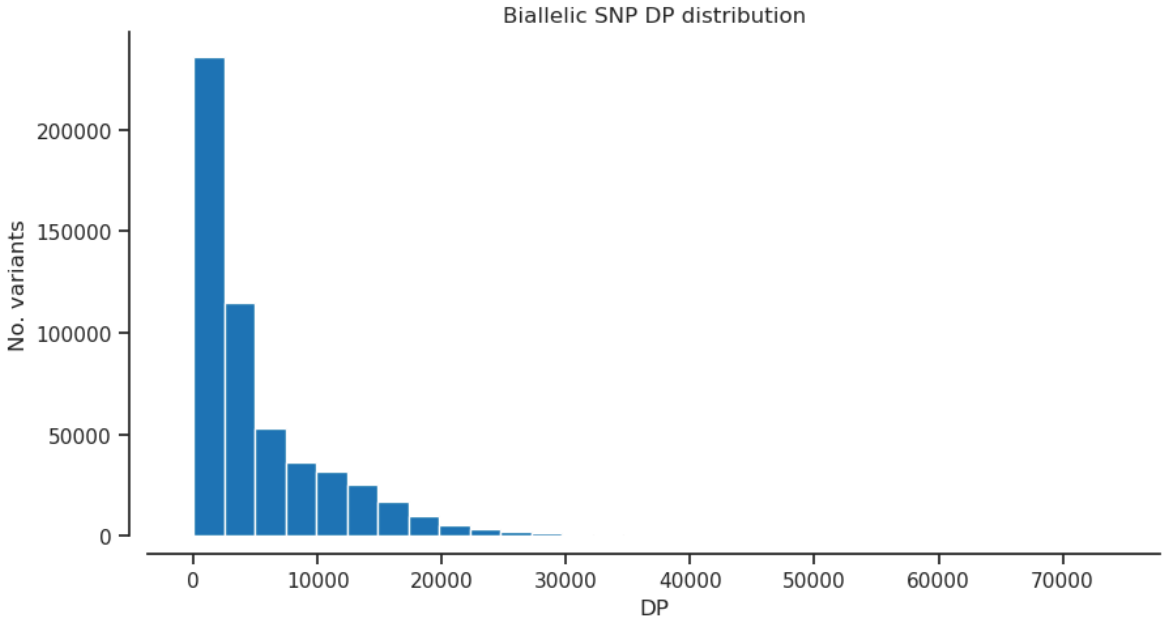


DP - Approximate read depth

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

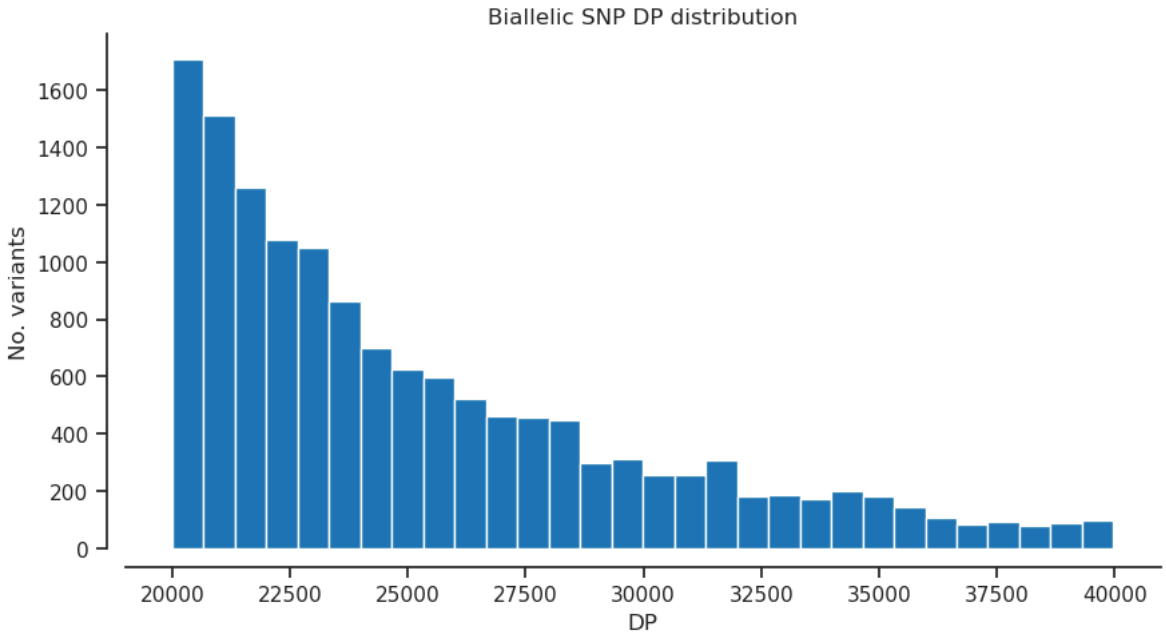


In [40]:
```
plot_hist('DP','biallelic')
```

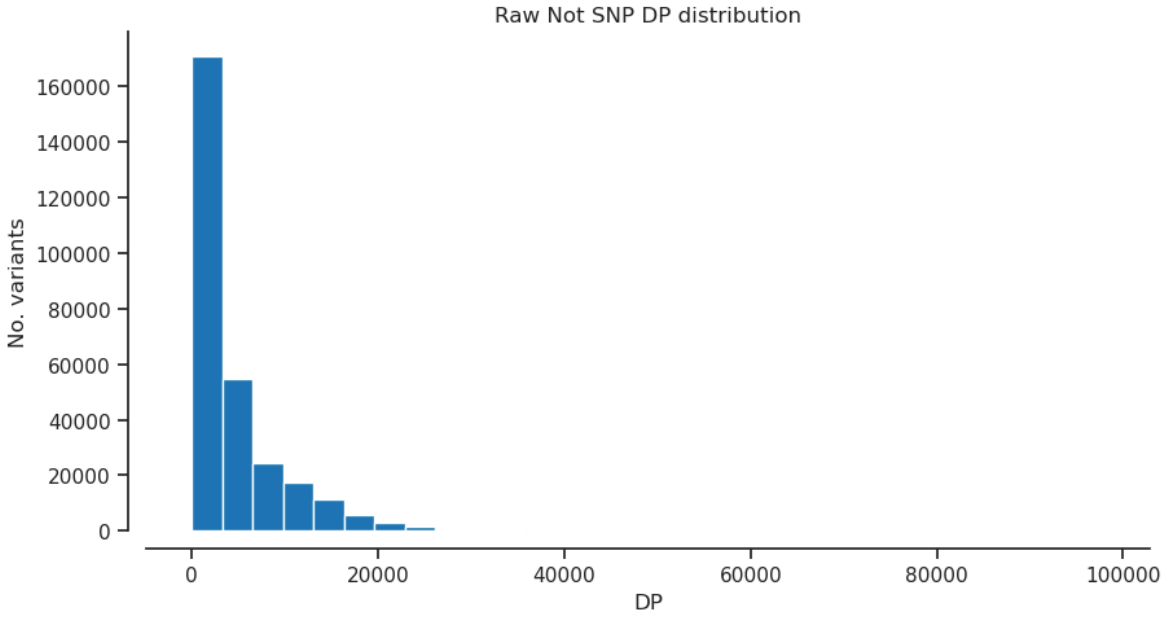


In [41]:
```
filter_expression = '(DP > 20000) & (DP < 40000)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

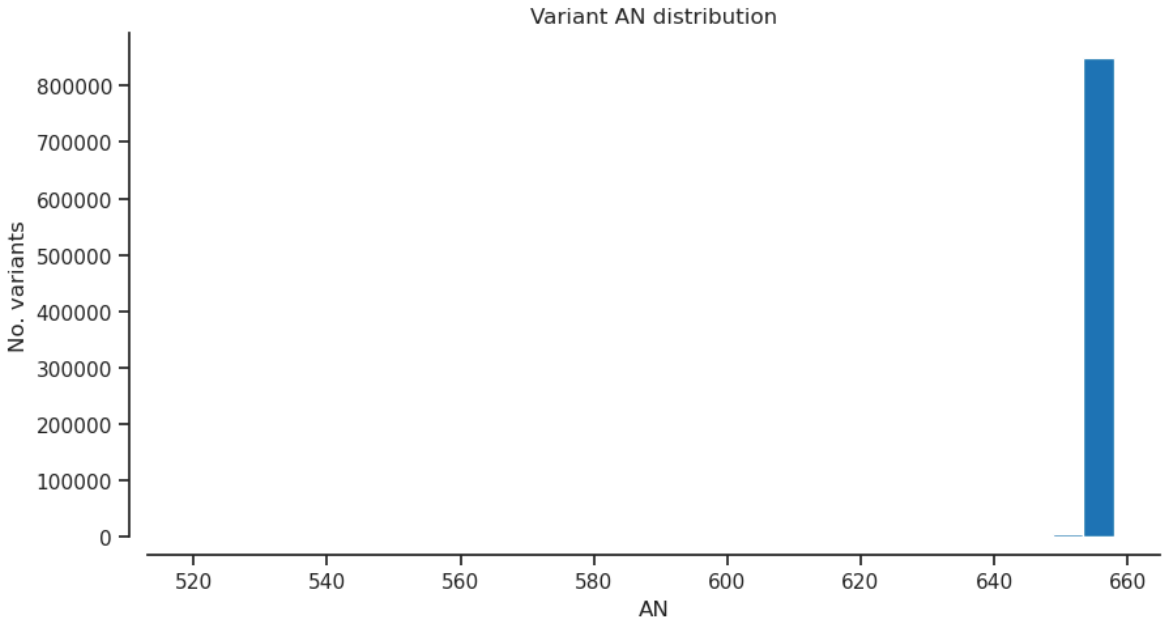


In [43]:
```
plot_hist('DP','notsnp')
```

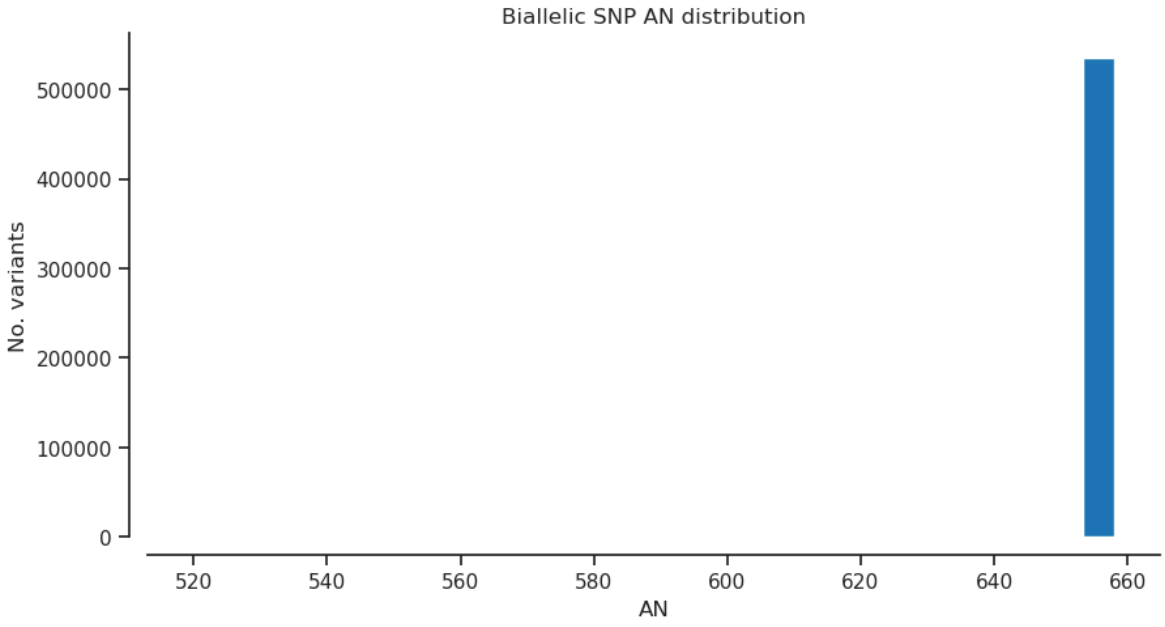


AN - Total number of alleles in called genotypes

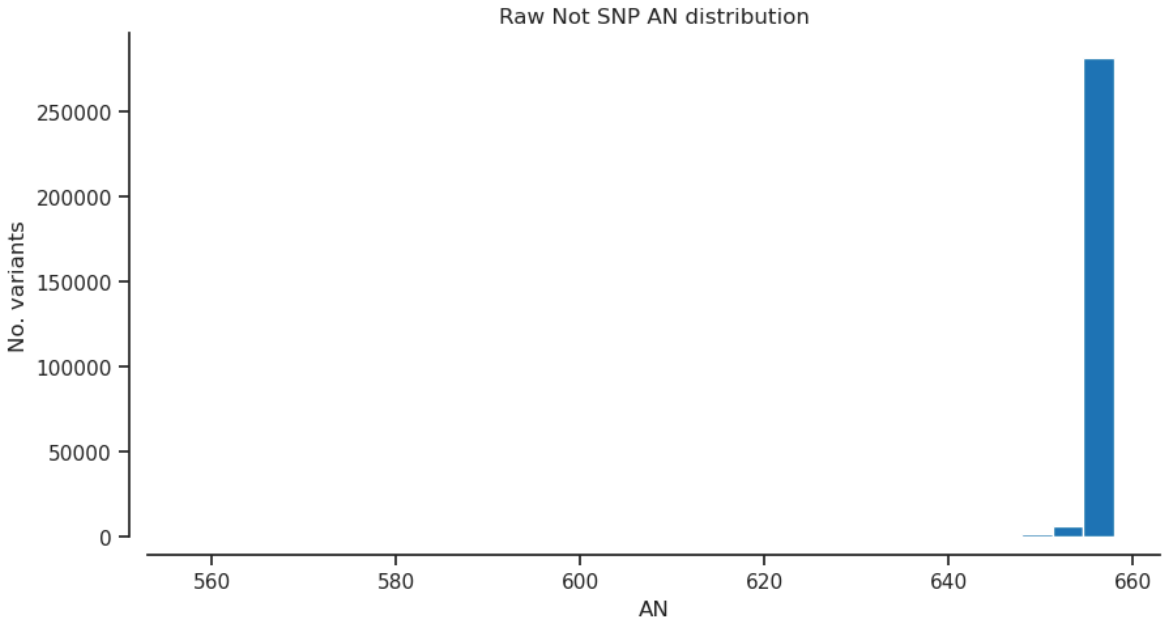
In [44]:
```
plot_hist('AN','var') # Total number of alleles in called genotypes
```



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]:
```
473343
```

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=(859843, 329, 2) dtype=int8 chunks=(65536, 64, 2) nbytes=539.6M cbytes=23.3M cratio=23.2 compression=gzip compression_opts=1 values=h5py._hl.dataset.Dataset>

	0	1	2	3	4	...	324	325	326	327	328
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
...						...					
859840	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
859841	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
859842	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=(473343, 329, 2) dtype=int8 chunks=(1849, 329, 2) nbytes=297.0M cbytes=22.9M cratio=13.0 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3	4	...	324	325	326	327	328
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
...						...					
473340	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
473341	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
473342	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=(473343, 4) dtype=int32 chunks=(29584, 4) nbytes=7.2M cbytes=1.1M cratio=6.7 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3
0	650	8	0	0
1	650	8	0	0
2	656	2	0	0
...		...		
473340	653	5	0	0
473341	657	1	0	0
473342	657	1	0	0

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

Out[52]: <AlleleCountsArray shape=(473343, 4) dtype=int32>

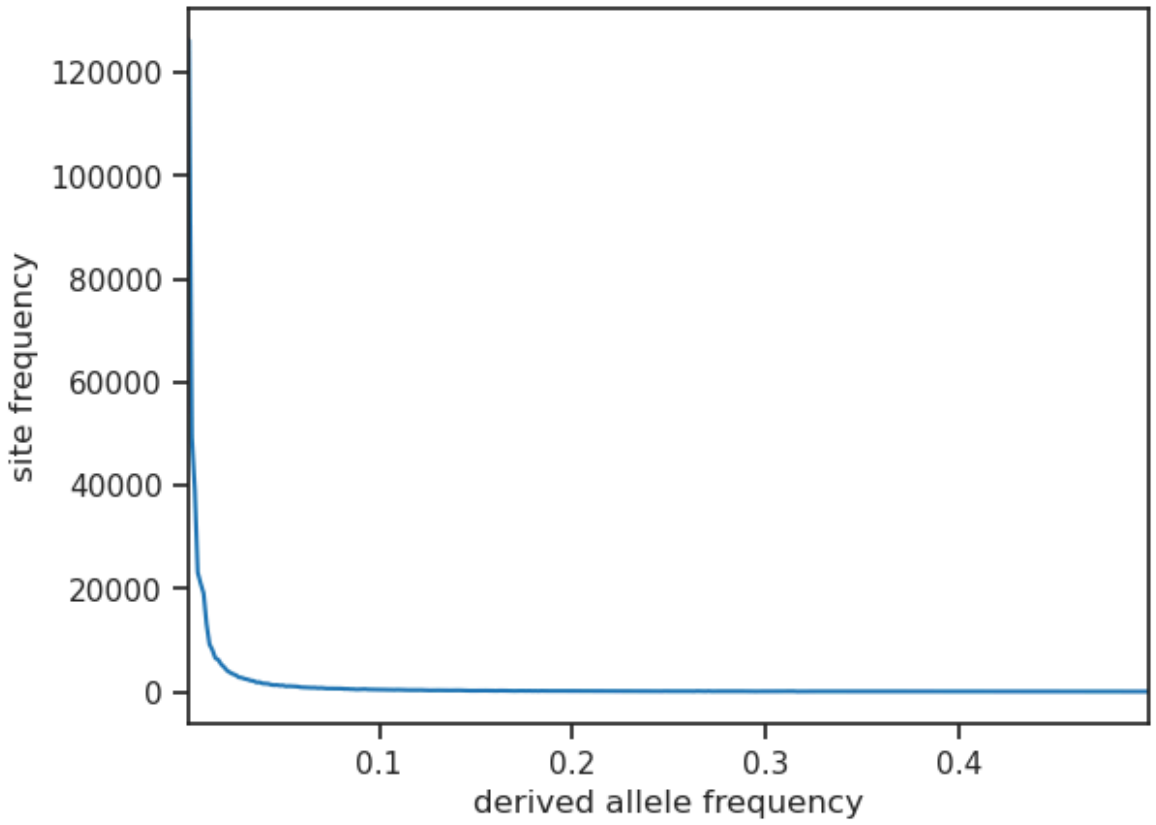
	0	1	2	3
0	650	8	0	0
1	650	8	0	0
2	656	2	0	0
...		...		
473340	653	5	0	0
473341	657	1	0	0
473342	657	1	0	0

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([[650,   8],
       [650,   8],
       [656,   2],
       ...,
       [653,   5],
       [657,   1],
       [657,   1]], 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=(473343,) dtype=bool chunks=(236672,)
   nbytes=462.2K cbytes=96.7K cratio=4.8
   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=(445189, 329, 2) dtype=int8 chunks=(1740, 329, 2) nbytes=279.4M cbytes=20.5M cratio=13.6 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3	4	...	324	325	326	327	328
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
...						...					
445186	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
445187	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
445188	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]: 445189

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'AUT00006-001',
 b'AUT00006-002',
 b'AUT00006-003',
 b'AUT00006-004',
 b'AUT00006-005',
 b'AUT00006-006',
 b'AUT00006-007',
 b'AUT00006-008',
 b'AUT00006-009',
 b'AUT00006-010',
 b'AUT00006-011',
 b'AUT00006-012',
 b'AUT00006-013',
 b'AUT00006-014',
 b'AUT00006-015',
 b'AUT00006-016',
 b'AUT00006-017',
 b'AUT00006-018',
 b'AUT00006-019',
 b'AUT00006-020',
 b'AUT00006-021',
 b'AUT00006-022',
 b'AUT00006-023',
 b'AUT00006-024',
 b'AUT00006-025',
 b'DEU00145-001',
 b'DEU00145-002',
 b'DEU00145-003',
 b'DEU00145-004',
 b'DEU00145-005',
 b'DEU00145-006',
 b'DEU00145-007',
 b'DEU00145-008',
 b'DEU00145-009',
 b'DEU00145-010',
 b'DEU00145-011',
 b'DEU00145-012',
 b'DEU00145-013',
 b'DEU00145-014',
 b'DEU00145-015',
 b'DEU00145-016',
 b'DEU00145-017',
 b'DEU00145-018',
 b'DEU00145-019',
 b'DEU00145-020',
 b'DEU00145-021',
 b'DEU00145-022',
 b'DEU00145-023',
 b'DEU00145-024',
 b'DEU00145-025',
 b'ESP00274-001',
 b'ESP00274-002',
 b'ESP00274-003',
 b'ESP00274-004',
 b'ESP00274-005',
 b'ESP00274-006',
 b'ESP00274-007',
 b'ESP00274-008',
 b'ESP00274-009',
 b'ESP00274-010',
```

```
b'ESP00274-011',
b'ESP00274-012',
b'ESP00274-013',
b'ESP00274-014',
b'ESP00274-015',
b'ESP00274-016',
b'ESP00274-017',
b'ESP00274-018',
b'ESP00274-019',
b'ESP00274-020',
b'ESP00274-021',
b'ESP00274-022',
b'ESP00274-023',
b'ESP00274-024',
b'ESP00274-025',
b'ESP00387-001',
b'ESP00387-002',
b'ESP00387-003',
b'ESP00387-004',
b'ESP00387-005',
b'ESP00387-006',
b'ESP00387-007',
b'ESP00387-008',
b'ESP00387-009',
b'ESP00387-010',
b'ESP00387-011',
b'ESP00387-012',
b'ESP00387-013',
b'ESP00387-014',
b'ESP00387-015',
b'ESP00387-016',
b'ESP00387-017',
b'ESP00387-018',
b'ESP00387-019',
b'ESP00387-020',
b'ESP00387-021',
b'ESP00387-022',
b'ESP00387-023',
b'ESP00387-024',
b'ESP00387-025',
b'FRA00052-001',
b'FRA00052-002',
b'FRA00052-003',
b'FRA00052-004',
b'FRA00052-005',
b'FRA00052-006',
b'FRA00052-007',
b'FRA00052-008',
b'FRA00052-009',
b'FRA00052-010',
b'FRA00052-011',
b'FRA00052-012',
b'FRA00052-013',
b'FRA00052-014',
b'FRA00052-015',
b'FRA00052-016',
b'FRA00052-017',
b'FRA00052-018',
b'FRA00052-019',
b'FRA00052-020',
```

```
b'FRA00052-021',
b'FRA00052-022',
b'FRA00052-023',
b'FRA00052-024',
b'FRA00052-025',
b'FRA00070-004',
b'FRA00070-005',
b'FRA00070-006',
b'FRA00070-011',
b'FRA00070-012',
b'FRA00070-013',
b'FRA00070-014',
b'FRA00070-015',
b'FRA00070-017',
b'FRA00070-018',
b'FRA00070-019',
b'FRA00070-020',
b'FRA00070-021',
b'FRA00070-022',
b'FRA00070-023',
b'FRA00070-024',
b'FRA00070-025',
b'GBR00014-001',
b'GBR00014-002',
b'GBR00014-003',
b'GBR00014-004',
b'GBR00014-005',
b'GBR00014-006',
b'GBR00014-007',
b'GBR00014-010',
b'GBR00014-011',
b'GBR00014-012',
b'GBR00014-013',
b'GBR00014-014',
b'GBR00014-015',
b'GBR00014-017',
b'GBR00014-019',
b'GBR00014-020',
b'GBR00014-021',
b'GBR00014-023',
b'GBR00014-024',
b'GBR00014-025',
b'GBR00014-026',
b'GBR00014-027',
b'GBR00014-028',
b'GBR00014-029',
b'GBR00014-030',
b'GBR00015-201',
b'GBR00015-202',
b'GBR00015-203',
b'GBR00015-204',
b'GBR00015-205',
b'GBR00015-206',
b'GBR00015-207',
b'GBR00015-208',
b'GBR00015-209',
b'GBR00015-210',
b'GBR00015-211',
b'GBR00015-212',
b'GBR00015-213',
```

```
b'GBR00015-214',
b'GBR00015-215',
b'GBR00015-216',
b'GBR00015-217',
b'GBR00015-218',
b'GBR00015-219',
b'GBR00015-220',
b'GBR00015-221',
b'GBR00015-222',
b'GBR00015-223',
b'GBR00015-224',
b'GBR00015-225',
b'ROU00452-001',
b'ROU00452-002',
b'ROU00452-003',
b'ROU00452-004',
b'ROU00452-005',
b'ROU00452-006',
b'ROU00452-007',
b'ROU00452-008',
b'ROU00452-009',
b'ROU00452-010',
b'ROU00452-011',
b'ROU00452-012',
b'ROU00452-013',
b'ROU00452-014',
b'ROU00452-015',
b'ROU00452-016',
b'ROU00452-017',
b'ROU00452-018',
b'ROU00452-019',
b'ROU00452-020',
b'ROU00452-021',
b'ROU00452-022',
b'ROU00452-023',
b'ROU00452-024',
b'ROU00452-025',
b'SVN00010-001',
b'SVN00010-002',
b'SVN00010-003',
b'SVN00010-004',
b'SVN00010-005',
b'SVN00010-006',
b'SVN00010-007',
b'SVN00010-008',
b'SVN00010-009',
b'SVN00010-010',
b'SVN00010-011',
b'SVN00010-012',
b'SVN00010-013',
b'SVN00010-014',
b'SVN00010-015',
b'SVN00010-016',
b'SVN00010-017',
b'SVN00010-018',
b'SVN00010-019',
b'SVN00010-020',
b'SVN00010-021',
b'SVN00010-022',
b'SVN00010-023',
```

```
b'SVN00010-024',
b'SVN00010-025',
b'SVN00032-001',
b'SVN00032-002',
b'SVN00032-003',
b'SVN00032-004',
b'SVN00032-005',
b'SVN00032-006',
b'SVN00032-007',
b'SVN00032-008',
b'SVN00032-009',
b'SVN00032-010',
b'SVN00032-011',
b'SVN00032-012',
b'SVN00032-013',
b'SVN00032-014',
b'SVN00032-015',
b'SVN00032-016',
b'SVN00032-017',
b'SVN00032-018',
b'SVN00032-019',
b'SVN00032-020',
b'SVN00032-021',
b'SVN00032-022',
b'SVN00032-023',
b'SVN00032-024',
b'SVN00032-025',
b'TUR00024-001',
b'TUR00024-002',
b'TUR00024-003',
b'TUR00024-004',
b'TUR00024-005',
b'TUR00024-006',
b'TUR00024-007',
b'TUR00024-008',
b'TUR00024-009',
b'TUR00024-010',
b'TUR00024-011',
b'TUR00024-012',
b'TUR00024-013',
b'TUR00024-014',
b'TUR00024-015',
b'TUR00024-016',
b'TUR00024-017',
b'TUR00024-018',
b'TUR00024-019',
b'TUR00024-020',
b'TUR00024-021',
b'TUR00024-022',
b'TUR00024-023',
b'TUR00024-024',
b'TUR00024-025',
b'TUR00111-001',
b'TUR00111-002',
b'TUR00111-003',
b'TUR00111-004',
b'TUR00111-005',
b'TUR00111-006',
b'TUR00111-007',
b'TUR00111-008',
```

```
 b'TUR00111-009',
 b'TUR00111-010',
 b'TUR00111-011',
 b'TUR00111-012',
 b'TUR00114-001',
 b'TUR00114-002',
 b'TUR00114-003',
 b'TUR00114-004',
 b'TUR00114-005',
 b'TUR00114-006',
 b'TUR00114-007',
 b'TUR00114-008',
 b'TUR00114-009',
 b'TUR00114-010',
 b'TUR00114-011',
 b'TUR00114-012',
 b'TUR00114-013',
 b'TUR00114-014',
 b'TUR00114-015',
 b'TUR00114-016',
 b'TUR00114-017',
 b'TUR00114-018',
 b'TUR00114-019',
 b'TUR00114-020',
 b'TUR00114-021',
 b'TUR00114-022',
 b'TUR00114-023',
 b'TUR00114-024',
 b'TUR00114-025']
```

In [62]:
```
samples_fn = '~/scratch/data/Qpetraea/Quercus_petraea_sample_list_scikit-
samples = pandas.read_csv(samples_fn, sep='\t')
samples
```

Out[62]:

	ID	Population
0	AUT00006-001	AUT00006
1	AUT00006-002	AUT00006
2	AUT00006-003	AUT00006
3	AUT00006-004	AUT00006
4	AUT00006-005	AUT00006
...	...	...
324	TUR00114-021	TUR00114
325	TUR00114-022	TUR00114
326	TUR00114-023	TUR00114
327	TUR00114-024	TUR00114
328	TUR00114-025	TUR00114

329 rows × 2 columns

In [63]:
```
samples.Population.value_counts()
```

Out[63]:
```
Population
AUT00006    25
DEU00145    25
ESP00274    25
ESP00387    25
FRA00052    25
GBR00014    25
SVN00032    25
GBR00015    25
ROU00452    25
SVN00010    25
TUR00114    25
TUR00024    25
FRA00070    17
TUR00111    12
Name: count, dtype: int64
```

In [64]:
```
populations = samples.Population.unique()
populations
###This identifiers come from the metadata file
```

Out[64]:
```
array(['AUT00006', 'DEU00145', 'ESP00274', 'ESP00387', 'FRA00052',
       'FRA00070', 'GBR00014', 'GBR00015', 'ROU00452', 'SVN00010',
       'SVN00032', 'TUR00024', 'TUR00111', 'TUR00114'], dtype=object)
```

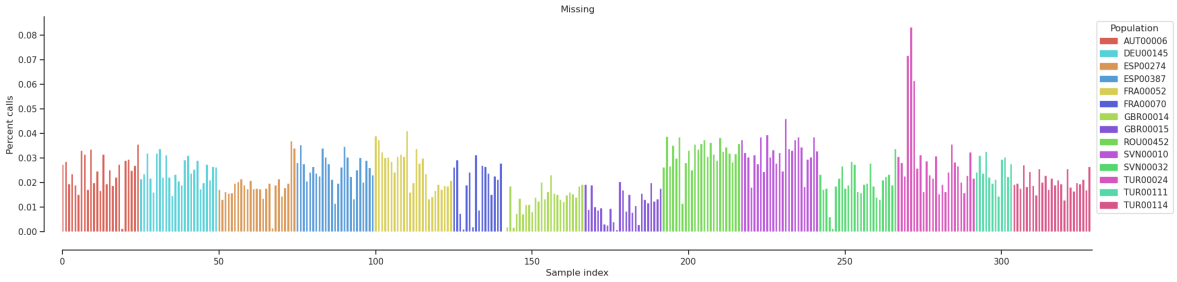
Gt frequency function

In [66]:
```
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", 14)
    pop2color = {'AUT00006': palette[0],
                 'DEU00145': palette[7],
                 'ESP00274': palette[1],
                 'ESP00387': palette[8],
                 'FRA00052': palette[2],
                 'FRA00070': palette[9],
                 'GBR00014': palette[3],
                 'GBR00015': palette[10],
                 'ROU00452': palette[4],
                 'SVN00010': palette[11],
                 'SVN00032': palette[5],
                 'TUR00024': palette[12],
                 'TUR00111': palette[6],
                 'TUR00114': palette[13]}
    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[7]),
               mpl.patches.Patch(color=palette[1]),
               mpl.patches.Patch(color=palette[8]),
               mpl.patches.Patch(color=palette[2]),
               mpl.patches.Patch(color=palette[9]),
```

```
                mpl.patches.Patch(color=palette[3]),
                mpl.patches.Patch(color=palette[10]),
                mpl.patches.Patch(color=palette[4]),
                mpl.patches.Patch(color=palette[11]),
                mpl.patches.Patch(color=palette[5]),
                mpl.patches.Patch(color=palette[12]),
                mpl.patches.Patch(color=palette[6]),
                mpl.patches.Patch(color=palette[13])]
    ax.legend(handles=handles, labels=['AUT00006', 'DEU00145', 'ESP00274'
       'GBR00014', 'GBR00015', 'ROU00452', 'SVN00010',
       'SVN00032', 'TUR00024', 'TUR00111', 'TUR00114'], title='Population
              bbox_to_anchor=(1, 1), loc='upper left')
```

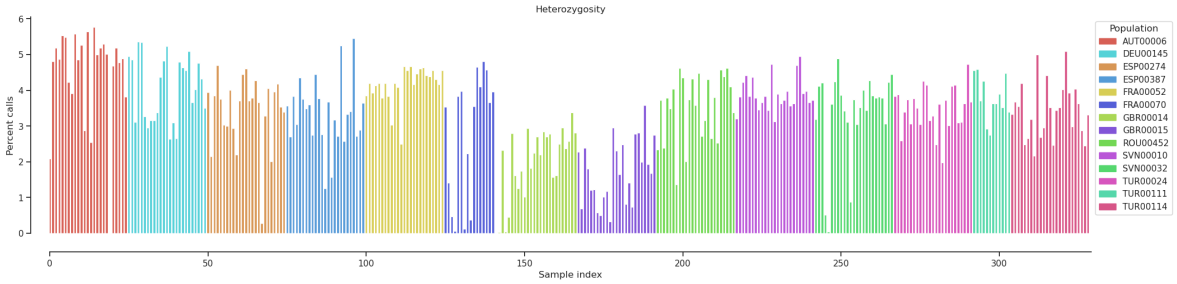
Plot missing

In [67]:
```
plot_genotype_frequency(pc_missing, 'Missing')
```



Plot heterozygosity

In [68]:
```
plot_genotype_frequency(pc_het, 'Heterozygosity')
```



PCA

In [69]:
```
palette = sns.color_palette("hls",14)
pop_colours = {
                'AUT00006': palette[0],
                 'DEU00145': palette[7],
                 'ESP00274': palette[1],
                 'ESP00387': palette[8],
                 'FRA00052': palette[2],
                 'FRA00070': palette[9],
                 'GBR00014': palette[3],
                 'GBR00015': palette[10],
                 'ROU00452': palette[4],
                 'SVN00010': palette[11],
                 'SVN00032': palette[5],
                 'TUR00024': palette[12],
```

```
                'TUR00111': palette[6],
                'TUR00114': palette[13]
}
```

In [70]:
```
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 [71]:
```
ac2 = gt_biallelic.count_alleles()
ac2
```

Out[71]:
<AlleleCountsChunkedArray shape=(445189, 2) dtype=int32 chunks=(55649, 2) nbytes=3.4M cbytes=821.3K cratio=4.2 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1
0	650	8
1	650	8
2	656	2
...	...	
445186	653	5
445187	657	1
445188	657	1

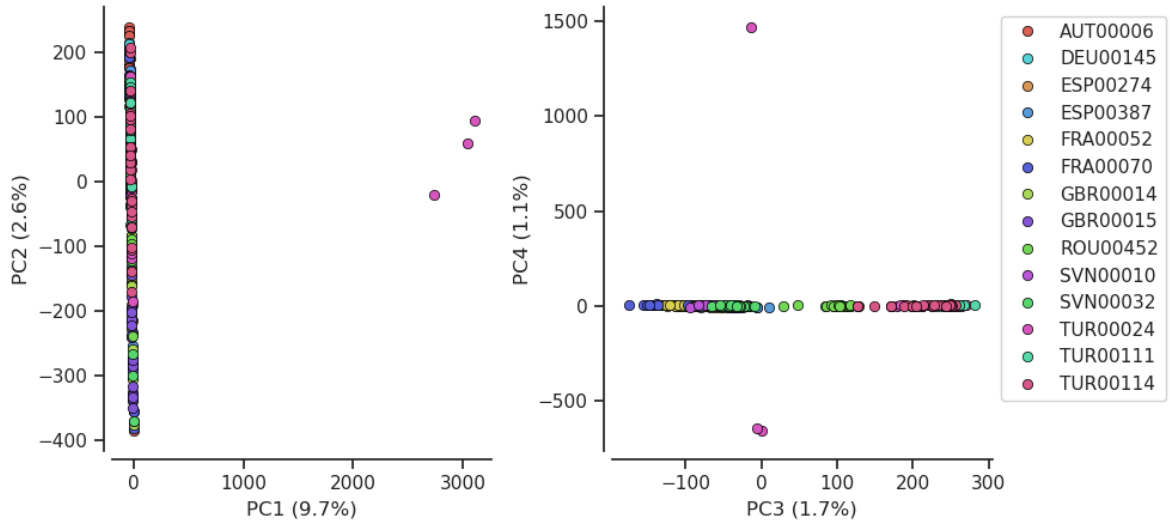
In [72]:
```
flt = (ac2[:, :2].min(axis=1) > 1)
gf = gt_biallelic.compress(flt, axis=0)
gn = gf.to_n_alt()
gn
```

Out[72]:
```
<ChunkedArrayWrapper shape=(318973, 329) dtype=int8 chunks=(2492, 329)
   nbytes=100.1M cbytes=14.1M cratio=7.1
   compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shu
ffle': 1, 'blocksize': 0}
   values=zarr.core.Array>
```

In [73]:
```
coords1, model1 = allel.pca(gn, n_components=10, scaler='patterson')
```

In [74]:
```
fig_pca(coords1, model1, 'Figure 1. Conventional PCA.')
```

Figure 1. Conventional PCA.



In [75]:
```
outliers = coords1[:,0]>2000
samples[outliers]
```

Out[75]:

	ID	Population
270	TUR00024-004	TUR00024
271	TUR00024-005	TUR00024
272	TUR00024-006	TUR00024

In [ ]: