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/Qilex/raw
```

Get data

In [3]:
```
callset_var_fn = '/users/mcevoysu/scratch/output/scikit-allel/Qilex/raw_S
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=(386595,) 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=66.0M cbytes=14.7M cratio=4.5 values=h5py._hl.group.Group>

	AC	AF	ALT	AN	BaseQRankSum	CHROM	DP	END	E
0	[3 -1 -1]	[0.00463 nan nan]	[b'*' b'' b'']	642	nan	b'chr01'	16516	-1	
1	[3 -1 -1]	[0.00463 nan nan]	[b'*' b'' b'']	642	nan	b'chr01'	16519	-1	
2	[3 -1 -1]	[0.00463 nan nan]	[b'*' b'' b'']	642	-0.782	b'chr01'	16518	-1	
...									
386592	[2 -1 -1]	[0.003086 nan nan]	[b'T' b'' b'']	642	nan	b'unanchored'	10	-1	
386593	[2 -1 -1]	[0.003086 nan nan]	[b'A' b'' b'']	642	nan	b'unanchored'	12	-1	
386594	[2 -1 -1]	[0.003086 nan nan]	[b'G' b'' b'']	642	nan	b'unanchored'	7	-1	

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

Out[7]: <VariantTable shape=(266535,) 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]	[0.001543 nan nan]	[b'T' b'' b'']	642	-0.983	b'chr01'	16462	-1	
1	[2 -1 -1]	[0.003086 nan nan]	[b'A' b'' b'']	642	-0.319	b'chr01'	16442	-1	
2	[1 -1 -1]	[0.001543 nan nan]	[b'G' b'' b'']	642	-0.842	b'chr01'	16426	-1	
...									
266532	[2 -1 -1]	[0.003086 nan nan]	[b'T' b'' b'']	642	nan	b'unanchored'	10	-1	
266533	[2 -1 -1]	[0.003086 nan nan]	[b'A' b'' b'']	642	nan	b'unanchored'	12	-1	
266534	[2 -1 -1]	[0.003086 nan nan]	[b'G' b'' b'']	642	nan	b'unanchored'	7	-1	

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

Out[8]: <VariantTable shape=(120060,) dtype=(numpy.record, [('AC', '<i4', (3,)), ('AF', '<f4', (3,)), ('ALT', 'O', (3,)), ('AN', '<i4'), ('BaseQRankSum', '<f4'), ('CHROM', 'O'), ('DP', '<i4'), ('END', '<i4'), ('ExcessHet', '<f4'), ('FILTER_LowQual', '?'), ('FILTER_PASS', '?'), ('FS', '<f4'), ('ID', 'O'), ('InbreedingCoeff', '<f4'), ('MLEAC', '<i4', (3,)), ('MLEAF', '<f4', (3,)), ('MQ', '<f4'), ('MQRankSum', '<f4'), ('POS', '<i4'), ('QD', '<f4'), ('QUAL', '<f4'), ('RAW_MQandDP', '<i4', (2,)), ('REF', 'O'), ('ReadPosRankSum', '<f4'), ('SOR', '<f4'), ('altlen', '<i4', (3,)), ('is_snp', '?'), ('numalt', '<i4')])>

	AC	AF	ALT	AN	BaseQRankSum	CHROM	DP	END	Ex
0	[3 -1 -1]	[0.00463 nan nan]	[b'*' b'' b'']	642	nan	b'chr01'	16516	-1	
1	[3 -1 -1]	[0.00463 nan nan]	[b'*' b'' b'']	642	nan	b'chr01'	16519	-1	
2	[3 -1 -1]	[0.00463 nan nan]	[b'*' b'' b'']	642	-0.782	b'chr01'	16518	-1	
...									
120057	[2 -1 -1]	[0.003086 nan nan]	[b'*' b'' b'']	642	nan	b'unanchored'	38	-1	
120058	[2 -1 -1]	[0.003086 nan nan]	[b'*' b'' b'']	642	nan	b'unanchored'	38	-1	
120059	[2 -1 -1]	[0.003086 nan nan]	[b'*' b'' b'']	642	nan	b'unanchored'	38	-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, 251724,  14332,    479])
```

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

Out[12]:
```
14811
```

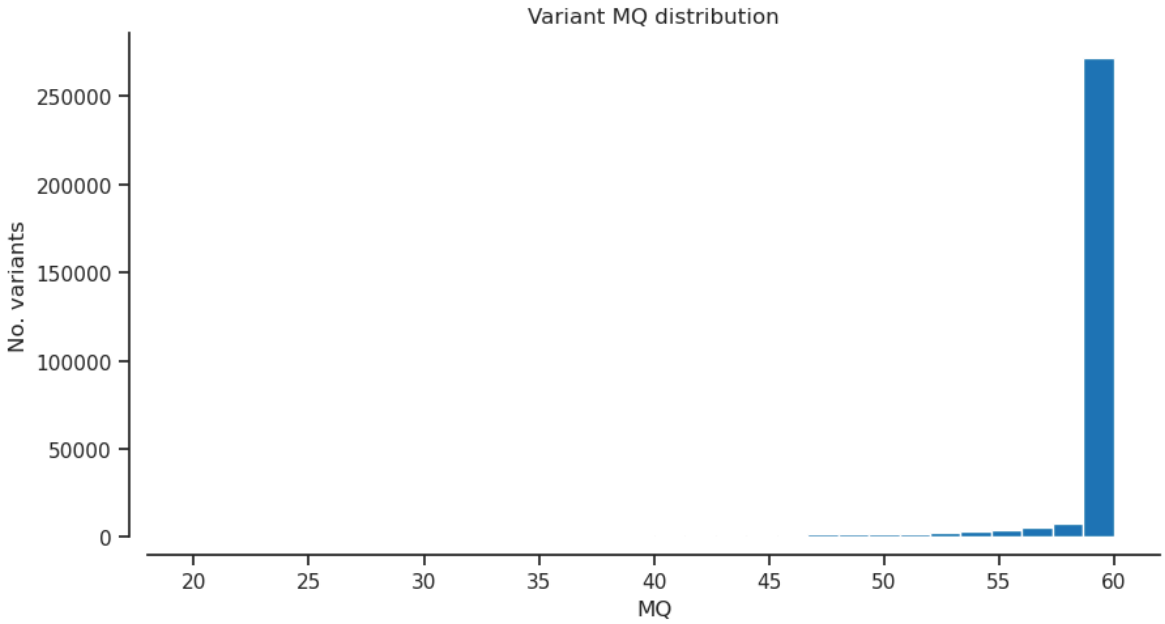
In [13]:
```
filter_expression = '(numalt == 1)'
biallelic_np = rawsnps.query(filter_expression)[:]
biallelic_np
```

Out[13]: <VariantTable shape=(251724,) dtype=(numpy.record, [('AC', '<i4', (3,)), ('AF', '<f4', (3,)), ('ALT', 'O', (3,)), ('AN', '<i4'), ('BaseQRankSum', '<f4'), ('CHROM', 'O'), ('DP', '<i4'), ('END', '<i4'), ('ExcessHet', '<f4'), ('FILTER_LowQual', '?'), ('FILTER_PASS', '?'), ('FS', '<f4'), ('ID', 'O'), ('InbreedingCoeff', '<f4'), ('MLEAC', '<i4', (3,)), ('MLEAF', '<f4', (3,)), ('MQ', '<f4'), ('MQRankSum', '<f4'), ('POS', '<i4'), ('QD', '<f4'), ('QUAL', '<f4'), ('RAW_MQandDP', '<i4', (2,)), ('REF', 'O'), ('ReadPosRankSum', '<f4'), ('SOR', '<f4'), ('altlen', '<i4', (3,)), ('is_snp', '?'), ('numalt', '<i4')])>

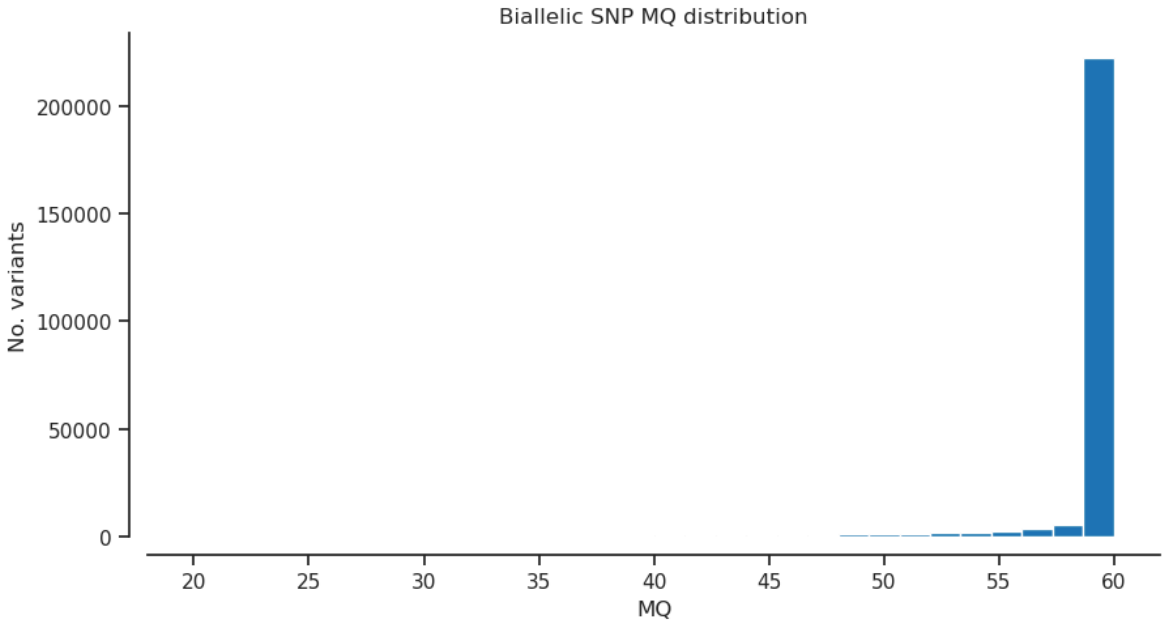
	AC	AF	ALT	AN	BaseQRankSum	CHROM	DP	END	Ex
0	[1 -1 -1]	[0.001543 nan nan]	[b'T' b'' b'']	642	-0.983	b'chr01'	16462	-1	
1	[2 -1 -1]	[0.003086 nan nan]	[b'A' b'' b'']	642	-0.319	b'chr01'	16442	-1	
2	[1 -1 -1]	[0.001543 nan nan]	[b'G' b'' b'']	642	-0.842	b'chr01'	16426	-1	
...									
251721	[2 -1 -1]	[0.003086 nan nan]	[b'T' b'' b'']	642	nan	b'unanchored'	10	-1	
251722	[2 -1 -1]	[0.003086 nan nan]	[b'A' b'' b'']	642	nan	b'unanchored'	12	-1	
251723	[2 -1 -1]	[0.003086 nan nan]	[b'G' b'' b'']	642	nan	b'unanchored'	7	-1	

MQ - RMS mapping quality

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

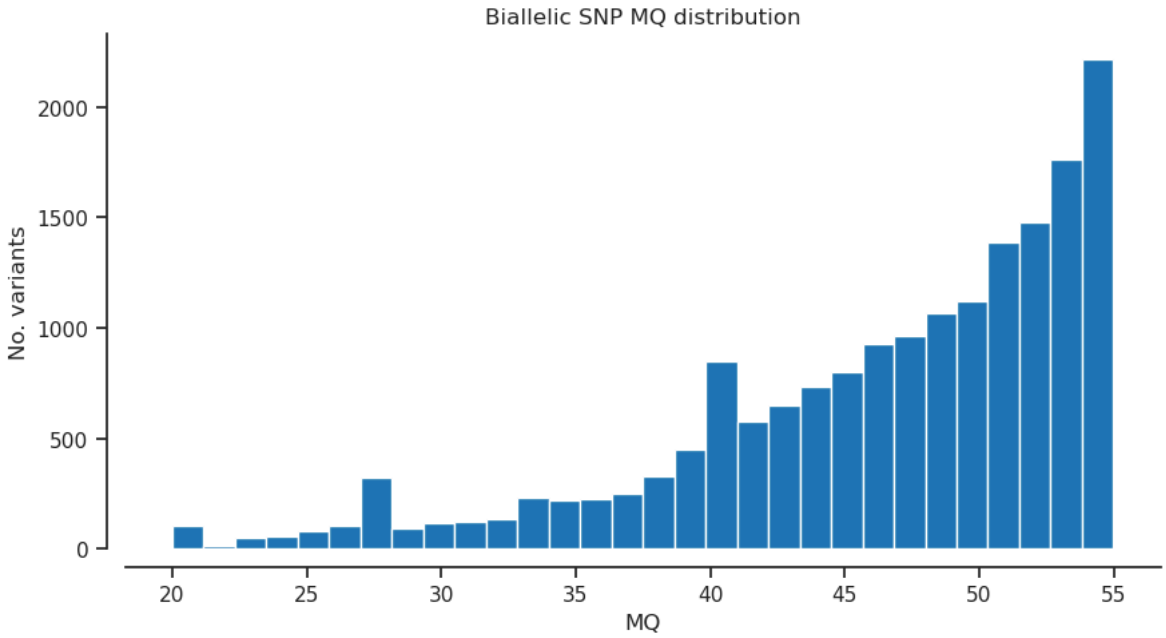


In [15]:
```
plot_hist('MQ','biallelic') # RMS mapping quality
```



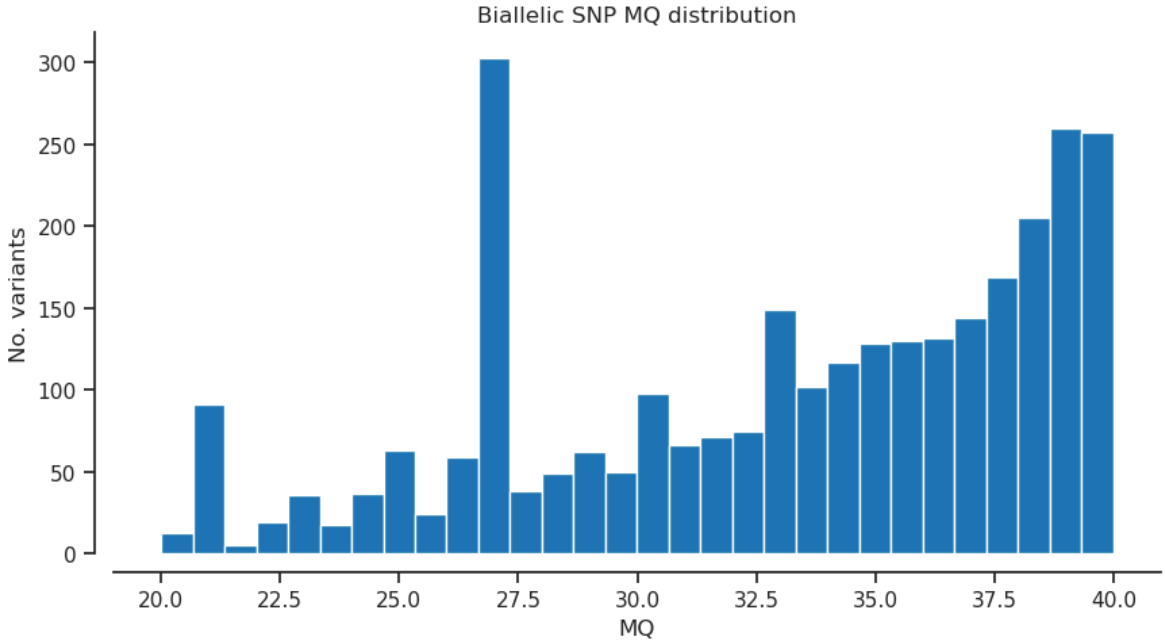
In [16]:
```
filter_expression = '(MQ < 55)'
bi_selection = biallelic_np.query(filter_expression)[:]
#np.count_nonzero(var_selection)
```

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

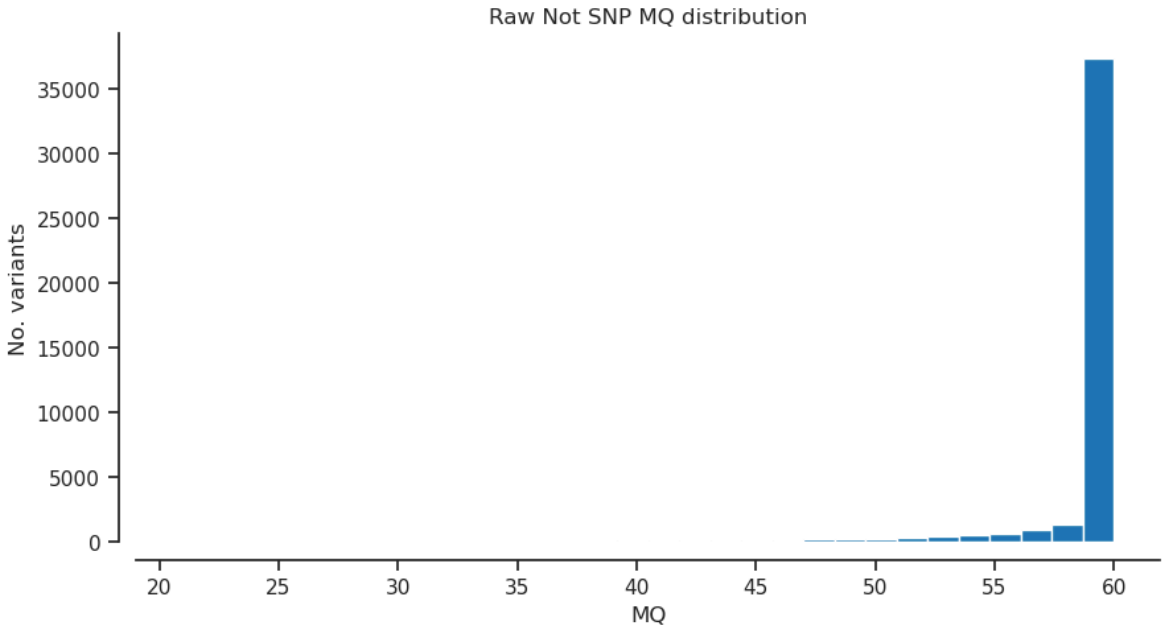


In [18]:
```
filter_expression = '(MQ < 40)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

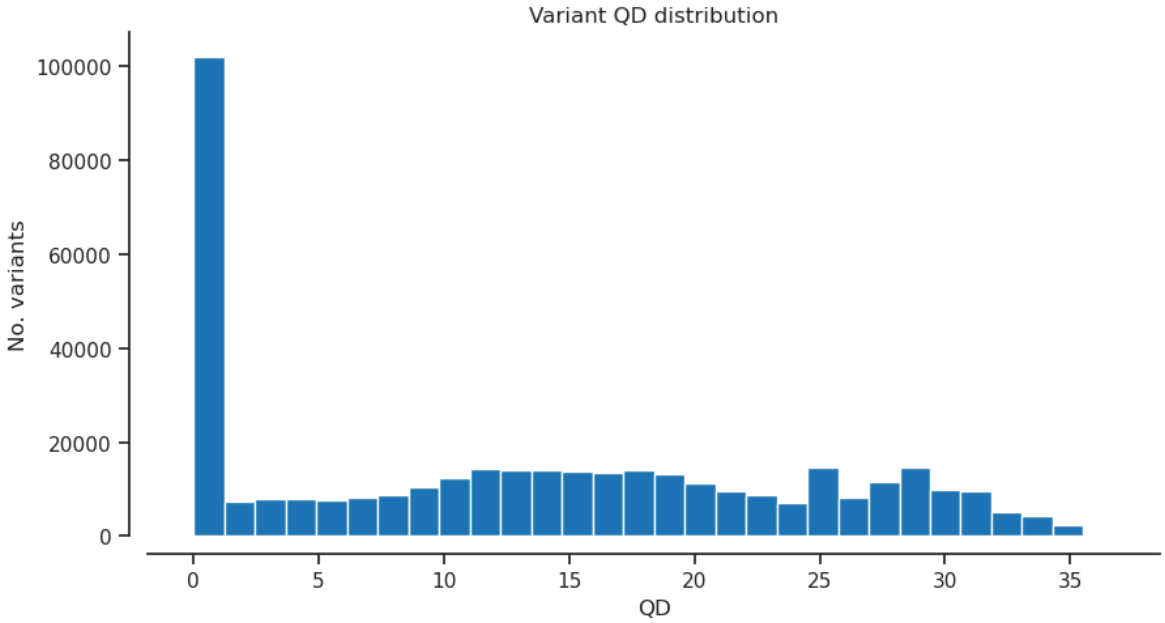


In [20]:
```
plot_hist('MQ','notsnp')
```

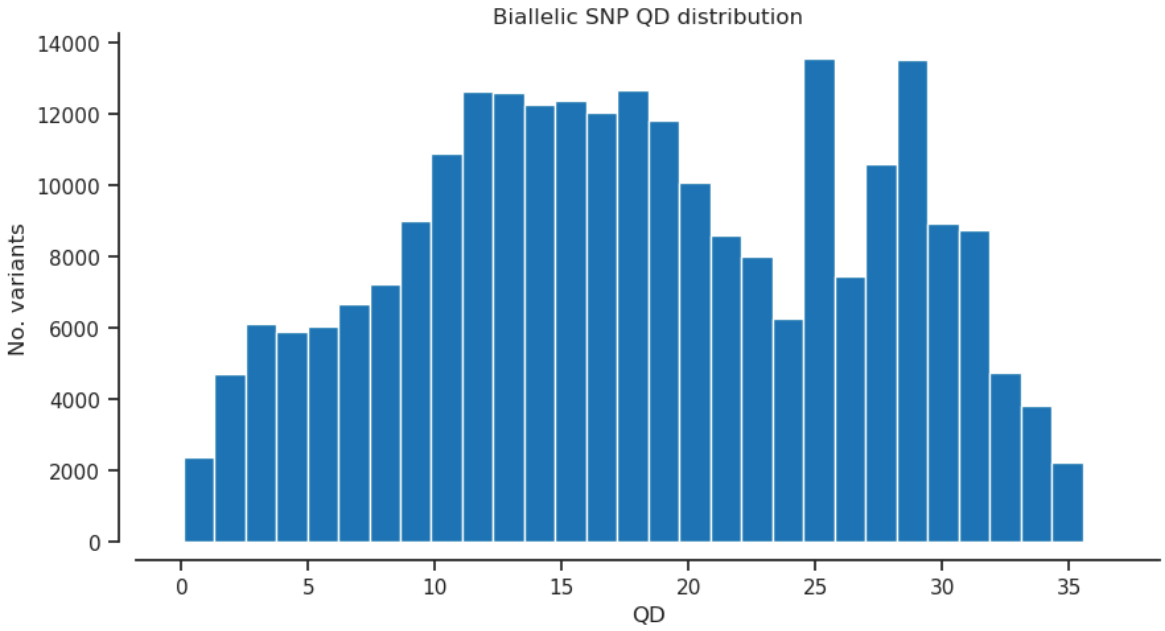


QD - Variant Confidence/Quality by Depth

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

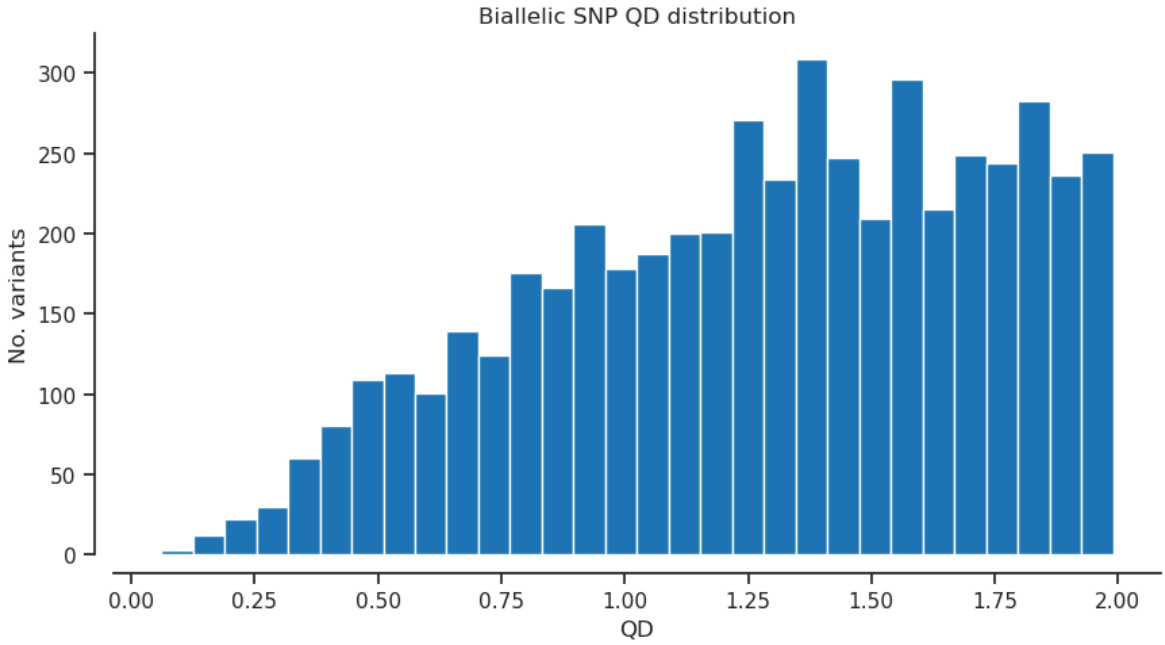


In [22]:
```
plot_hist('QD','biallelic') # Variant Confidence/Quality by Depth
```

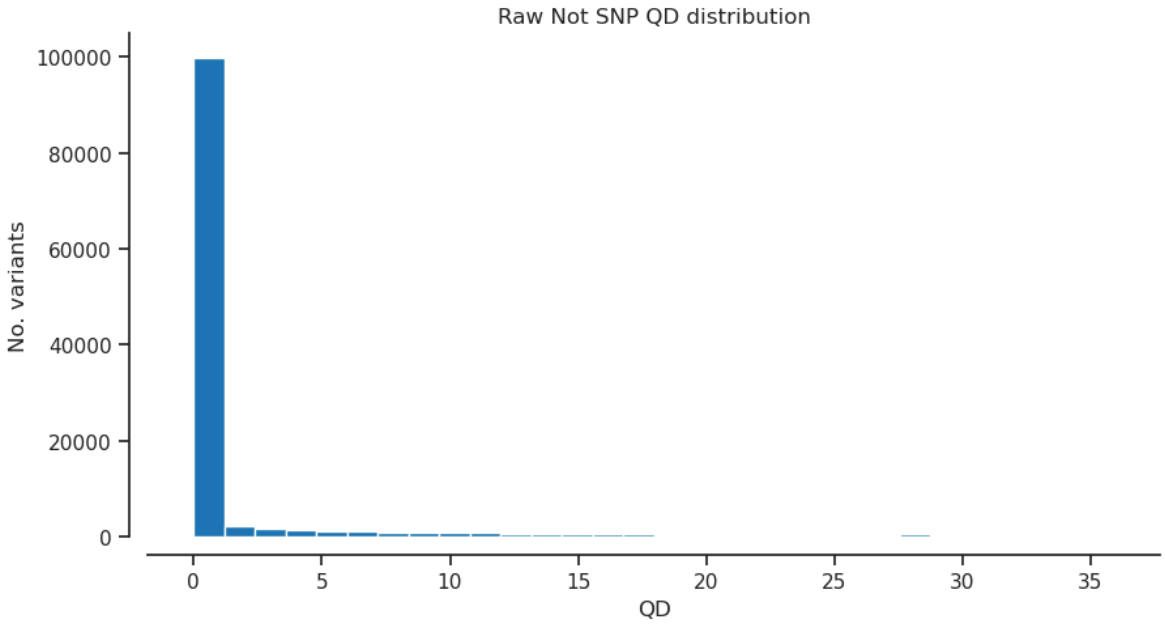


In [23]:
```
filter_expression = '(QD < 2)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

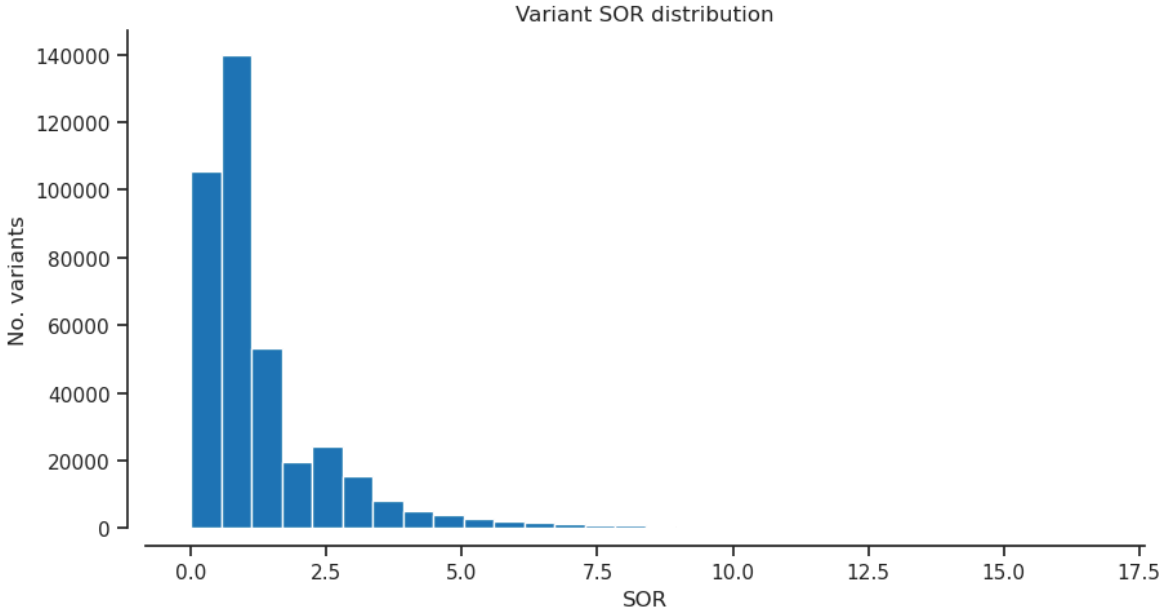


In [25]:
```
plot_hist('QD','notsnp') # Variant Confidence/Quality by Depth
```

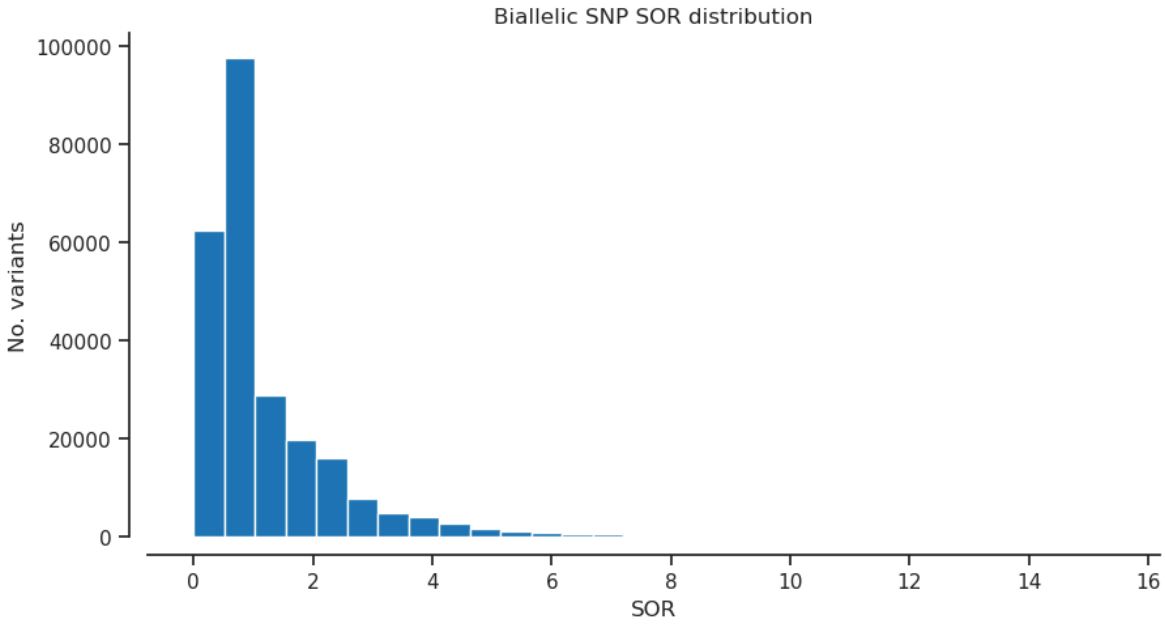


SOR - Symmetric Odds Ratio of 2x2 contingency table to detect strand bias

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

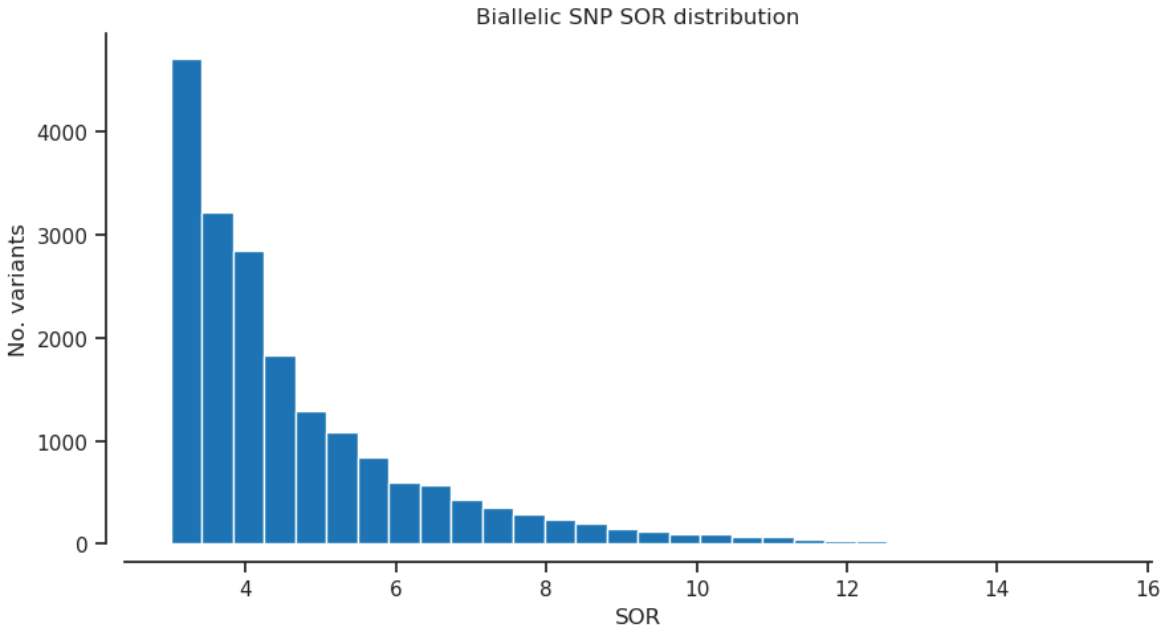


In [27]:
```
plot_hist('SOR','biallelic') # Symmetric Odds Ratio of 2x2 contingency ta
```

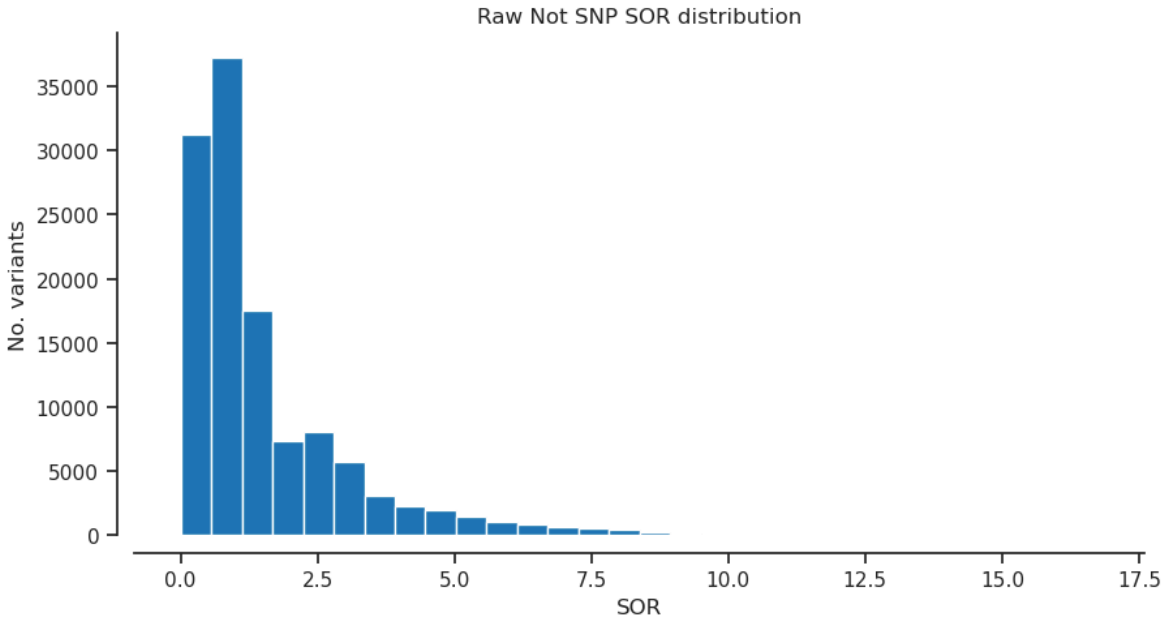


In [28]:
```
filter_expression = '(SOR > 3)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

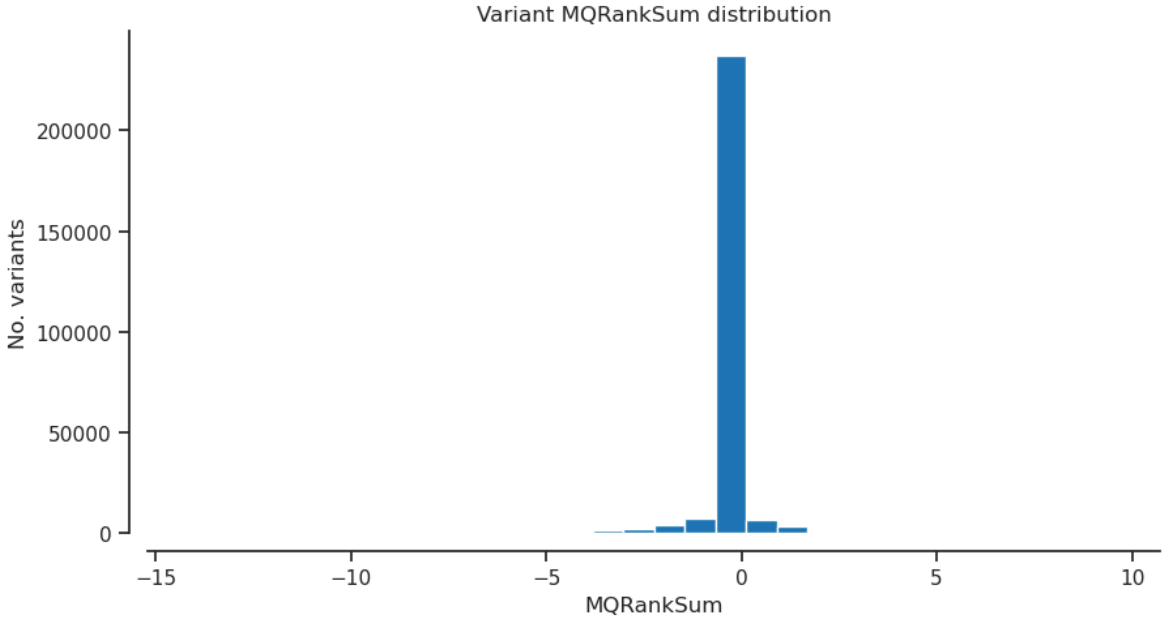


In [30]:
```
plot_hist('SOR','notsnp') # Symmetric Odds Ratio of 2x2 contingency table
```

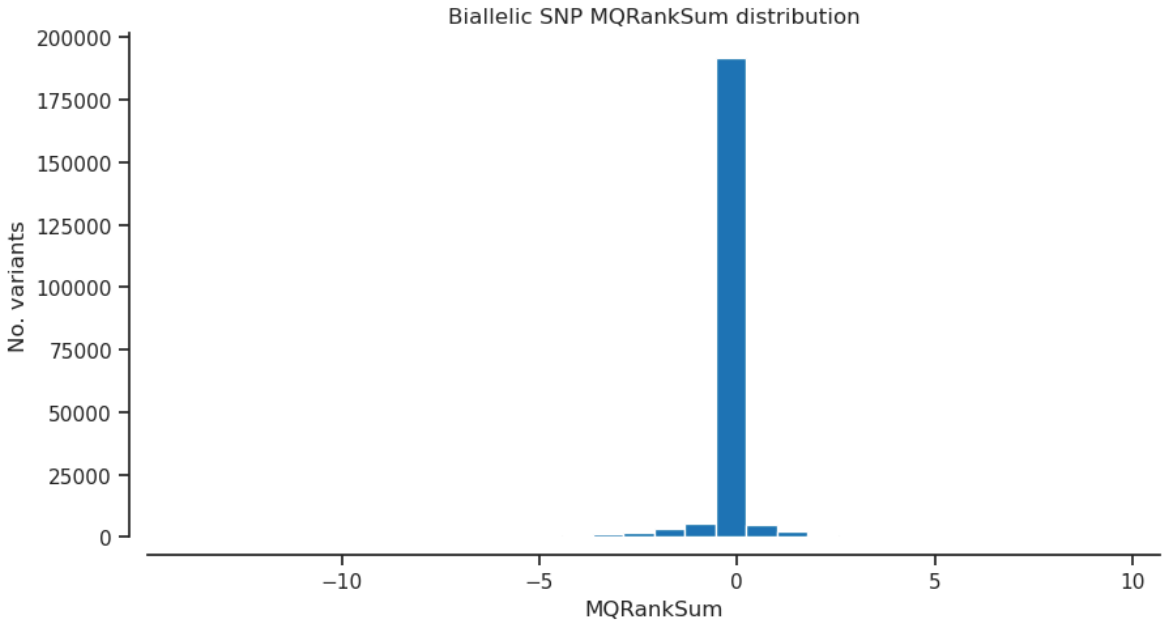


MQRankSum - Z-score From Wilcoxon rank sum test of Alt vs. Ref read mapping qualities

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

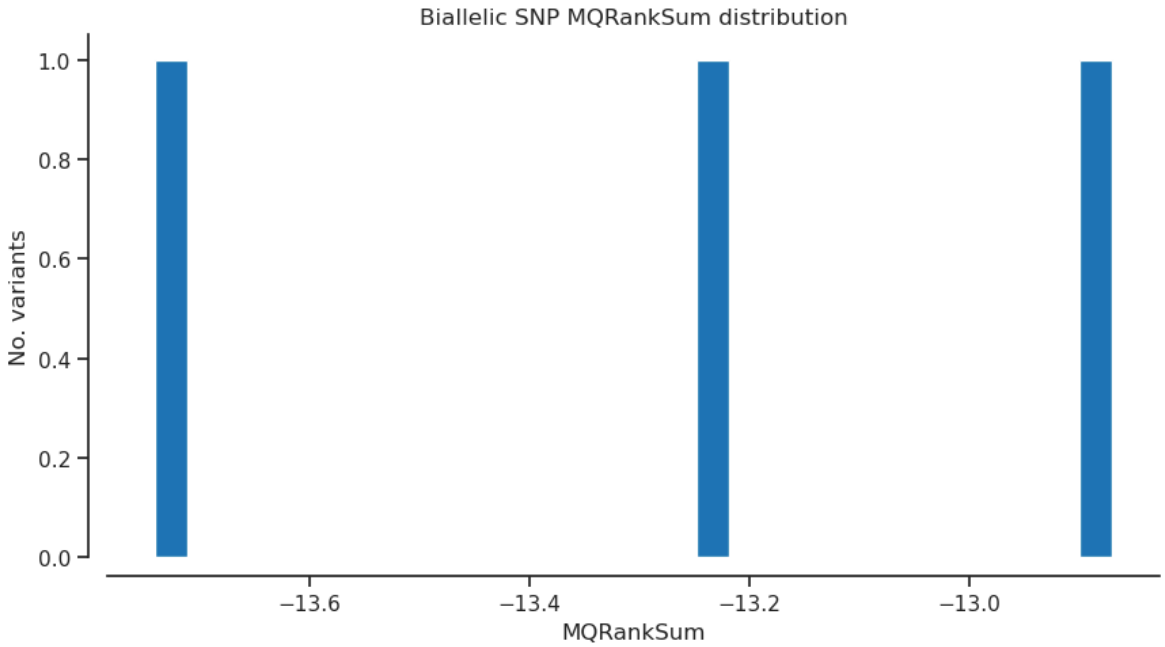


In [32]:
```
plot_hist('MQRankSum','biallelic') # Z-score From Wilcoxon rank sum test
```

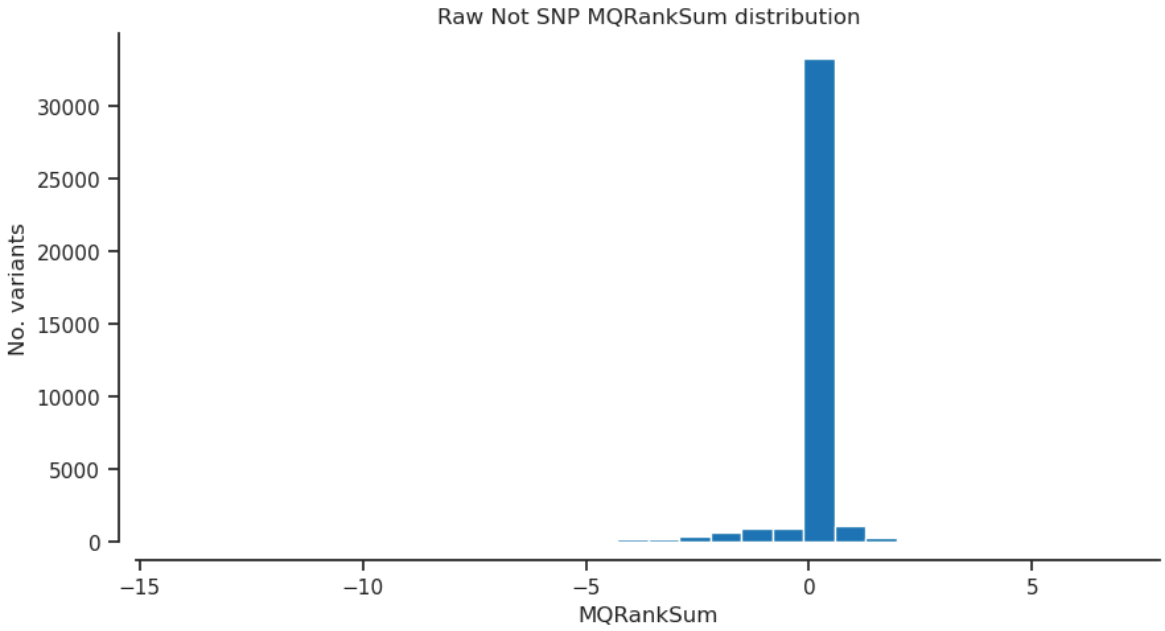


In [33]:
```
filter_expression = '(MQRankSum < -12.5)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

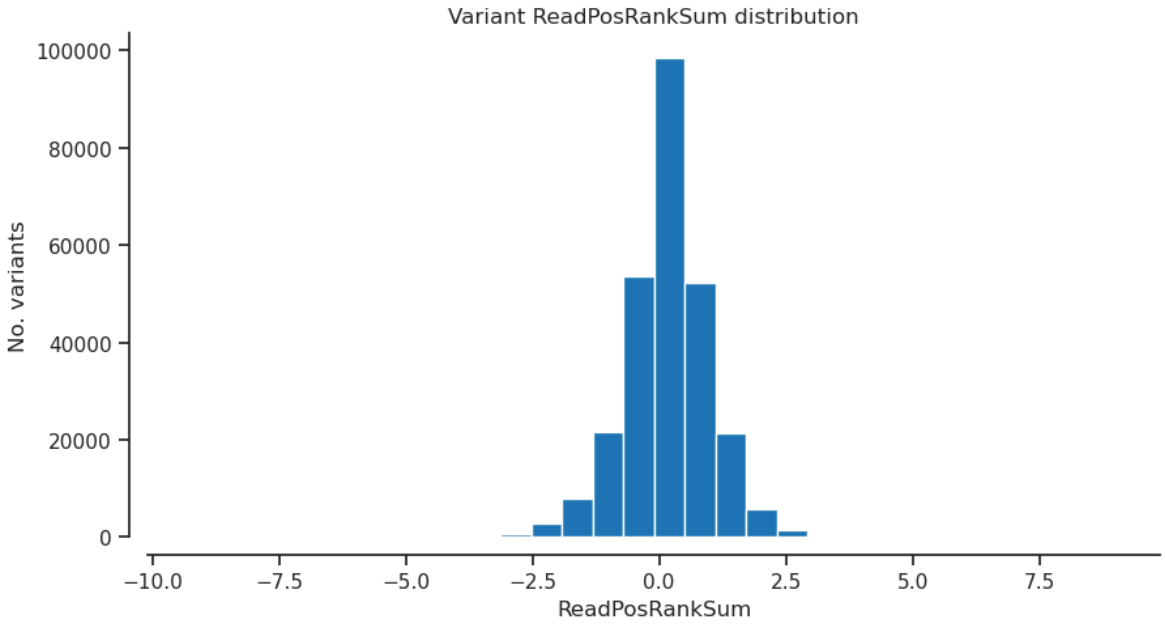


In [35]:
```
plot_hist('MQRankSum','notsnp') # Z-score From Wilcoxon rank sum test of
```

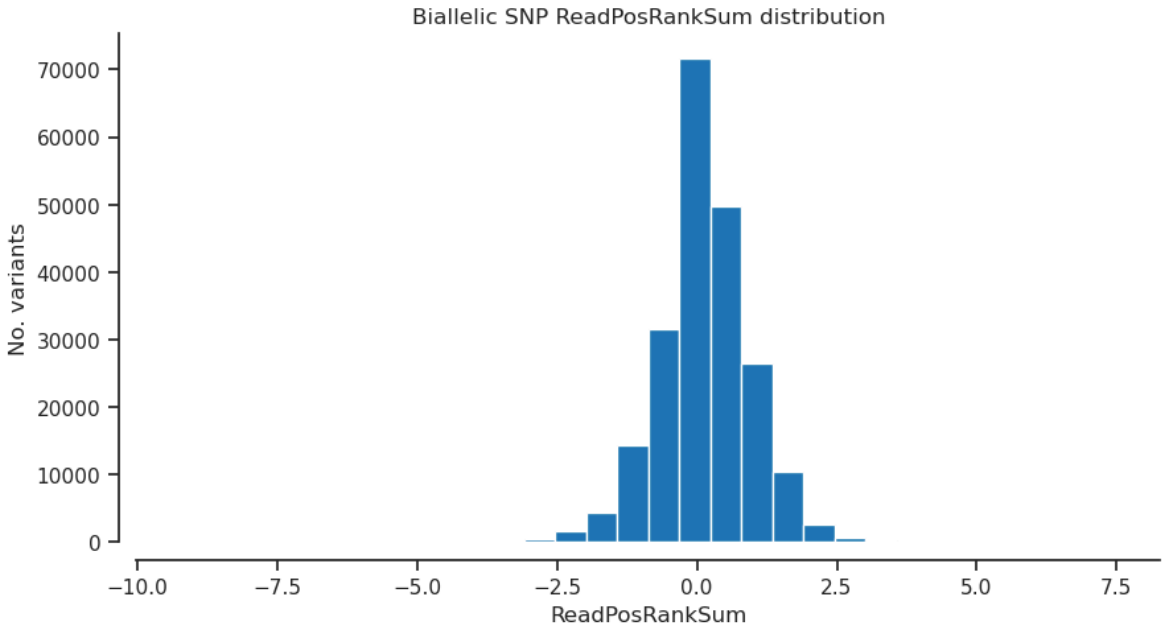


ReadPosRankSum - Z-score from Wilcoxon rank sum test of Alt vs. Ref read position bias

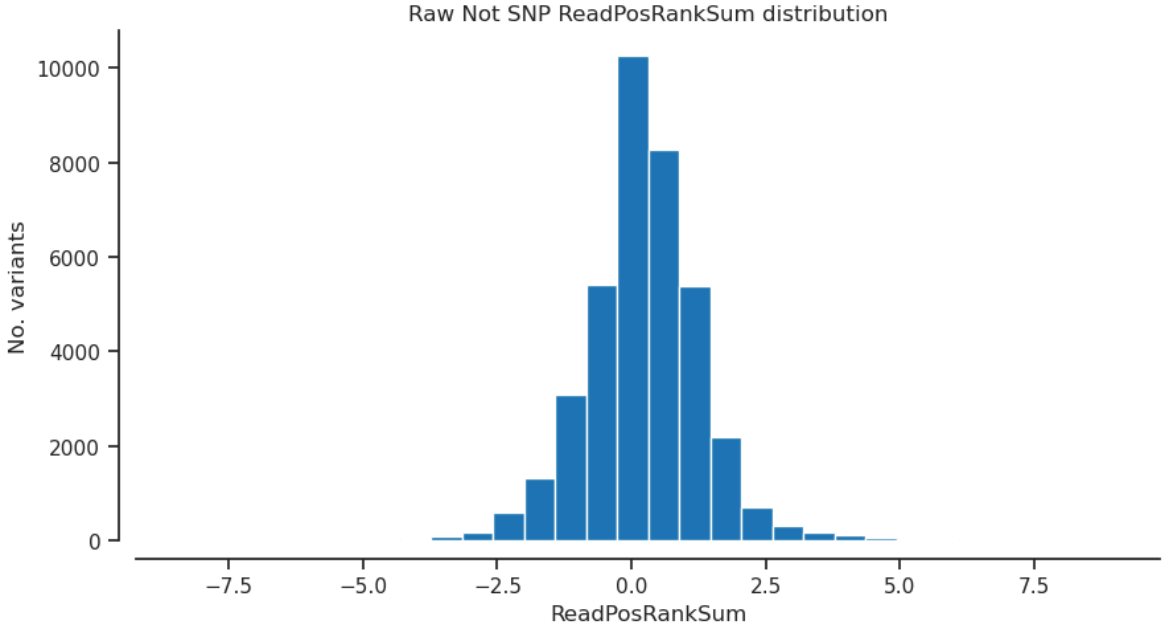
In [36]:
```
plot_hist('ReadPosRankSum','var') # Z-score from Wilcoxon rank sum test o
```



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

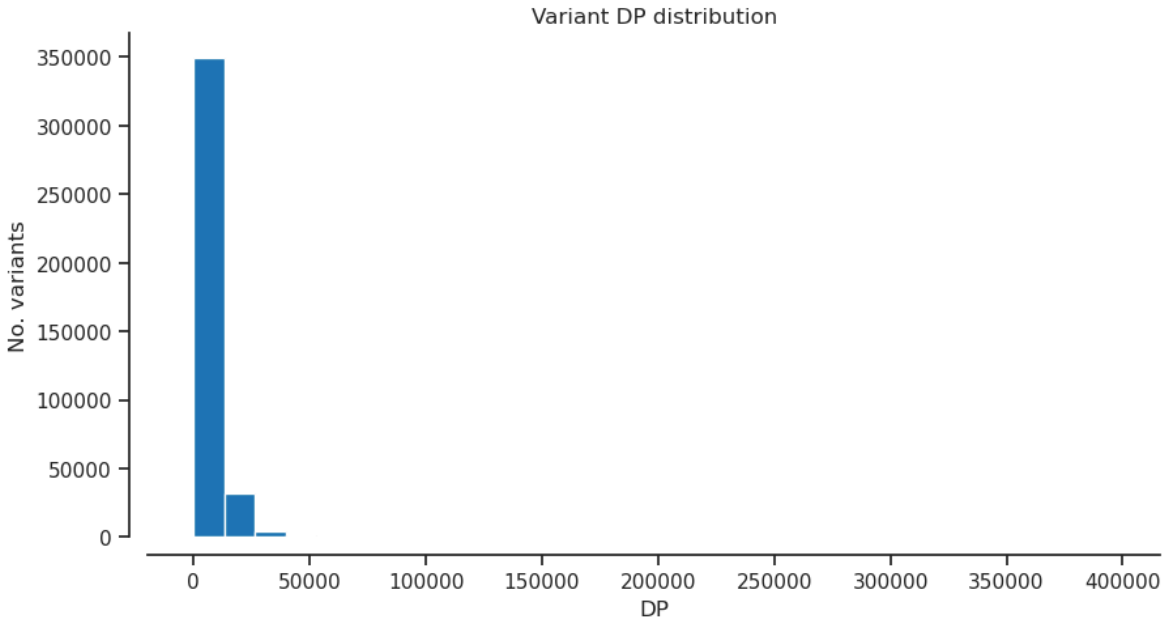


In [38]:
```
plot_hist('ReadPosRankSum','notsnp') # Z-score from Wilcoxon rank sum tes
```

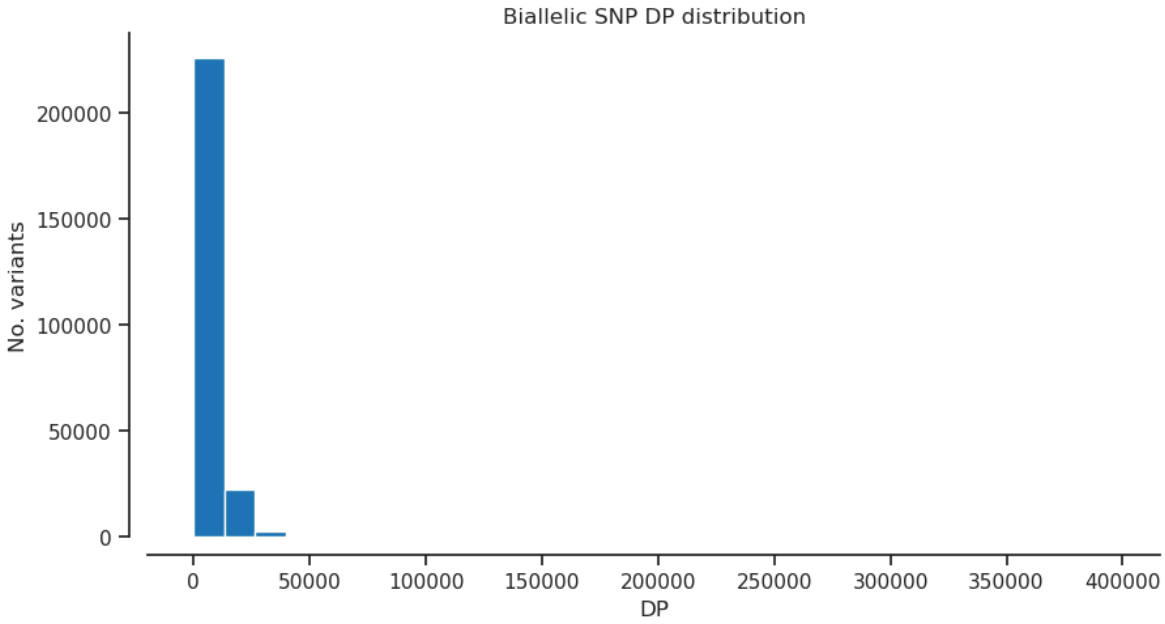


DP - Approximate read depth

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

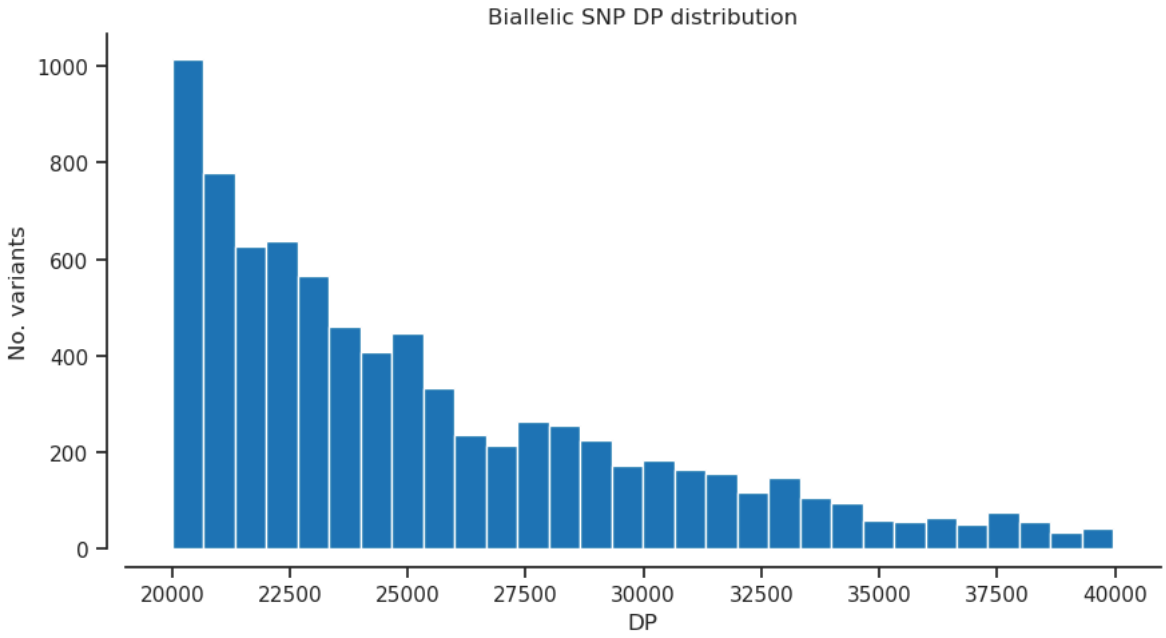


In [40]:
```
plot_hist('DP','biallelic')
```

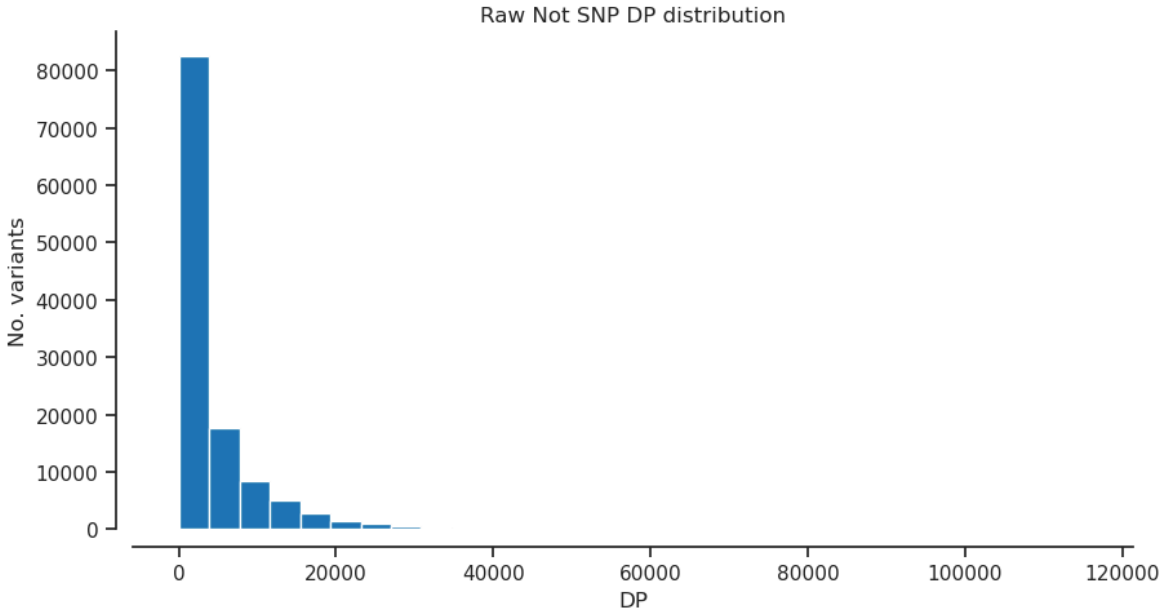


In [41]:
```
filter_expression = '(DP > 20000) & (DP < 40000)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

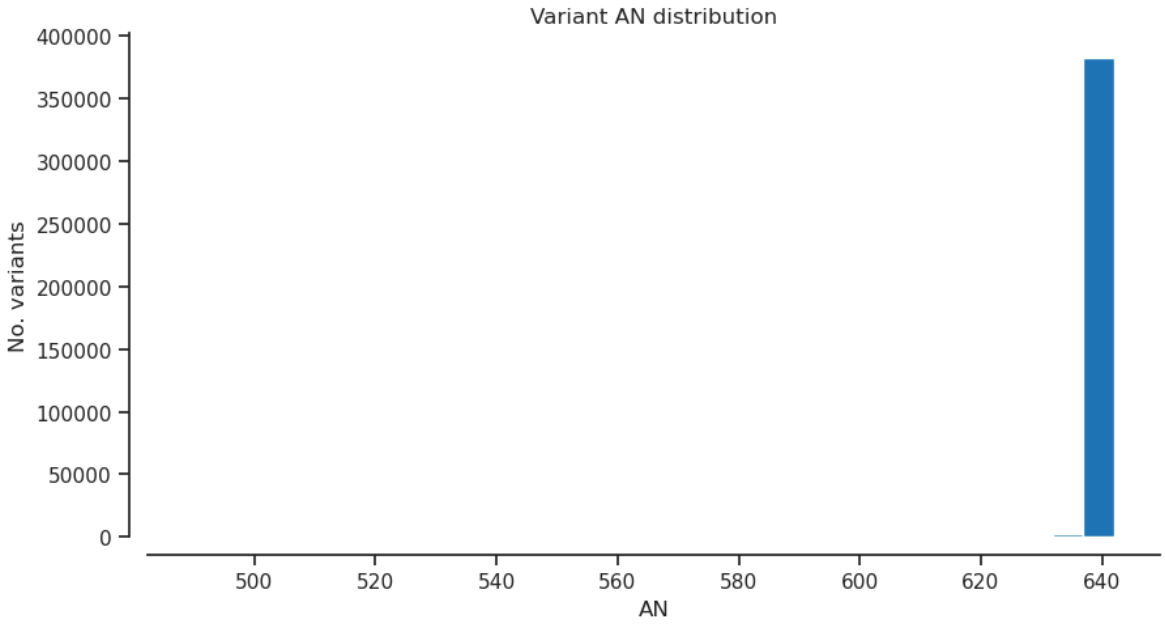


In [43]:
```
plot_hist('DP','notsnp')
```

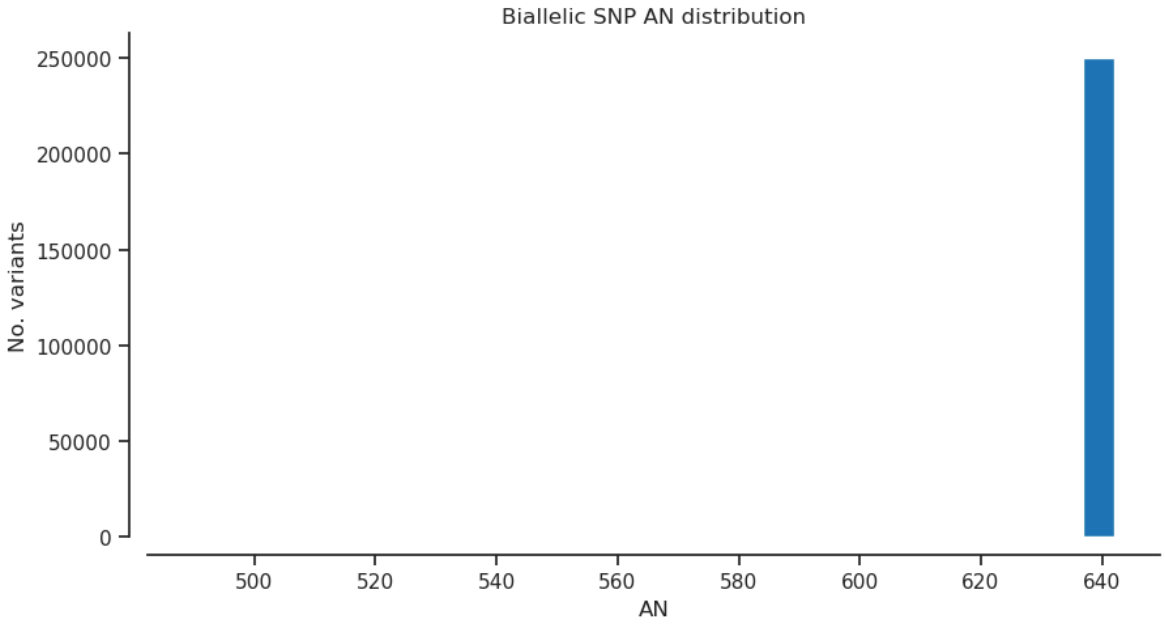


AN - Total number of alleles in called genotypes

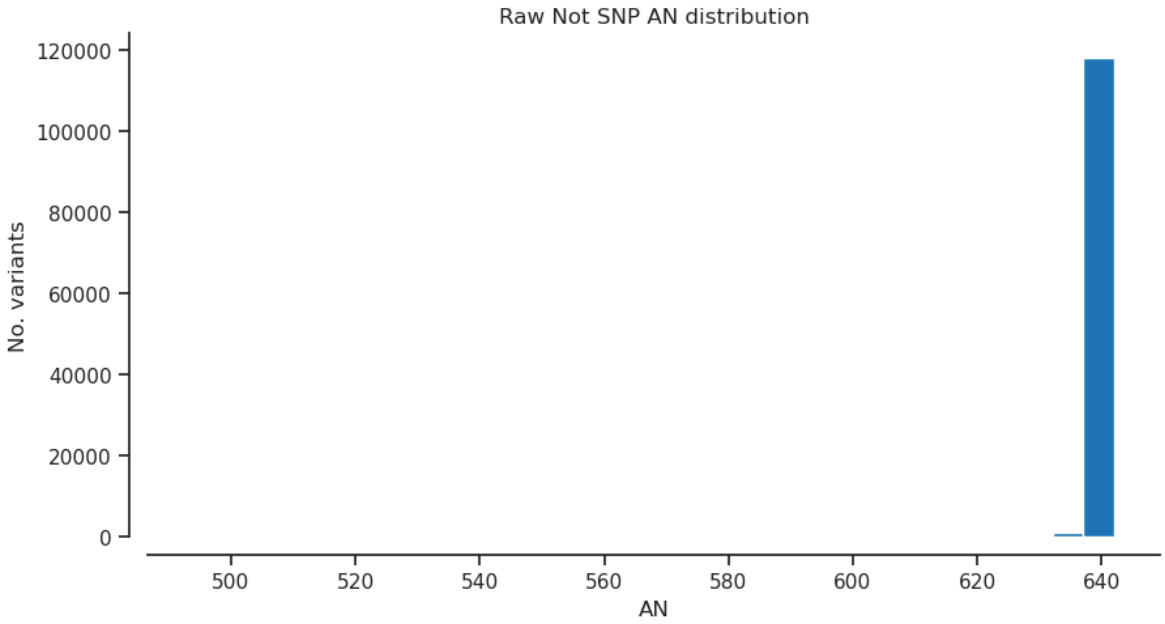
In [44]:
```
plot_hist('AN','var') # Total number of alleles in called genotypes
```



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]:
```
222029
```

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=(386595, 321, 2) dtype=int8 chunks=(65536, 64, 2) nbytes=236.7M cbytes=10.4M cratio=22.7 compression=gzip compression_opts=1 values=h5py._hl.dataset.Dataset>

	0	1	2	3	4	...	316	317	318	319	320
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
...						...					
386592	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
386593	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
386594	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=(222029, 321, 2) dtype=int8 chunks=(1735, 321, 2) nbytes=135.9M cbytes=10.7M cratio=12.7 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3	4	...	316	317	318	319	320
0	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
1	0/1	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
...						...					
222026	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
222027	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
222028	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=(222029, 4) dtype=int32 chunks=(27754, 4) nbytes=3.4M cbytes=523.2K cratio=6.6 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3
0	641	1	0	0
1	640	2	0	0
2	641	1	0	0
...		...		
222026	641	1	0	0
222027	641	1	0	0
222028	641	1	0	0

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

Out[52]: <AlleleCountsArray shape=(222029, 4) dtype=int32>

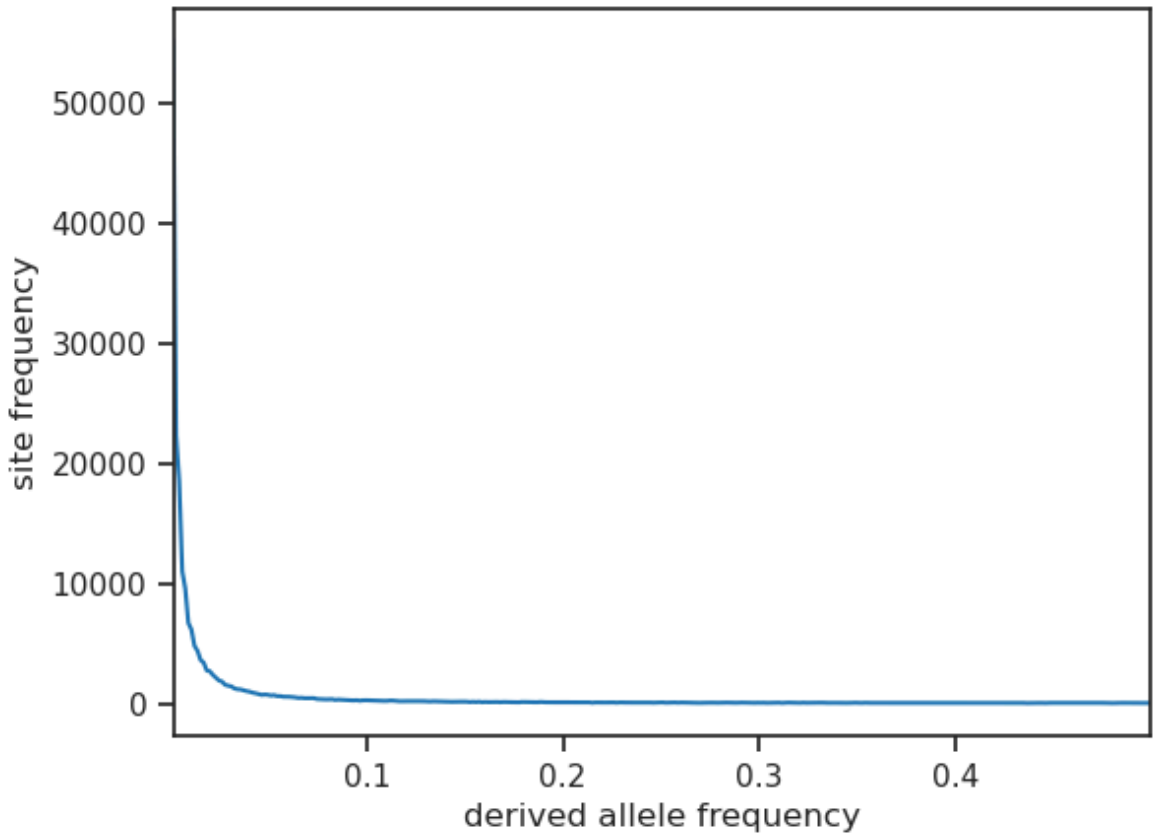
	0	1	2	3
0	641	1	0	0
1	640	2	0	0
2	641	1	0	0
...		...		
222026	641	1	0	0
222027	641	1	0	0
222028	641	1	0	0

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([[641,   1],
       [640,   2],
       [641,   1],
       ...,
       [641,   1],
       [641,   1],
       [641,   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=(222029,) dtype=bool chunks=(222029,)
   nbytes=216.8K cbytes=47.3K cratio=4.6
   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=(207741, 321, 2) dtype=int8 chunks=(1623, 321, 2) nbytes=127.2M cbytes=9.4M cratio=13.6 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3	4	...	316	317	318	319	320
0	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
1	0/1	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
...						...					
207738	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
207739	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
207740	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]: 207741

In [66]:
```
pc_missing = gt_biallelic.count_missing(axis=0)[:] * 100 / n_variants
pc_het = gt_biallelic.count_het(axis=0)[:] * 100 / n_variants
```

Samples

In [58]:
```
samples_var = callset_var['samples']
samples_var = list(samples_var)
samples_var
```

Out[58]:
```
[b'ESP00053-001',
 b'ESP00053-002',
 b'ESP00053-003',
 b'ESP00053-004',
 b'ESP00053-005',
 b'ESP00053-006',
 b'ESP00053-007',
 b'ESP00053-008',
 b'ESP00053-009',
 b'ESP00053-010',
 b'ESP00053-011',
 b'ESP00053-012',
 b'ESP00053-013',
 b'ESP00053-014',
 b'ESP00053-015',
 b'ESP00053-016',
 b'ESP00053-017',
 b'ESP00053-018',
 b'ESP00053-019',
 b'ESP00053-020',
 b'ESP00053-021',
 b'ESP00053-022',
 b'ESP00053-023',
 b'ESP00053-024',
 b'ESP00053-025',
 b'ESP00058-001',
 b'ESP00058-002',
 b'ESP00058-003',
 b'ESP00058-004',
 b'ESP00058-005',
 b'ESP00058-006',
 b'ESP00058-007',
 b'ESP00058-008',
 b'ESP00058-009',
 b'ESP00058-010',
 b'ESP00058-011',
 b'ESP00058-012',
 b'ESP00058-013',
 b'ESP00058-014',
 b'ESP00058-015',
 b'ESP00058-016',
 b'ESP00058-017',
 b'ESP00058-018',
 b'ESP00058-019',
 b'ESP00058-020',
 b'ESP00058-021',
 b'ESP00058-022',
 b'ESP00058-023',
 b'ESP00058-024',
 b'ESP00058-025',
 b'ESP00061-001',
 b'ESP00061-002',
 b'ESP00061-003',
 b'ESP00061-004',
 b'ESP00061-005',
 b'ESP00061-006',
 b'ESP00061-007',
 b'ESP00061-008',
 b'ESP00061-009',
 b'ESP00061-010',
```

```
b'ESP00061-011',
b'ESP00061-012',
b'ESP00061-013',
b'ESP00061-014',
b'ESP00061-015',
b'ESP00061-016',
b'ESP00061-017',
b'ESP00061-018',
b'ESP00061-019',
b'ESP00061-020',
b'ESP00061-021',
b'ESP00061-022',
b'ESP00061-023',
b'ESP00061-024',
b'ESP00061-025',
b'ESP00140-001',
b'ESP00140-002',
b'ESP00140-003',
b'ESP00140-004',
b'ESP00140-005',
b'ESP00140-006',
b'ESP00140-007',
b'ESP00140-008',
b'ESP00140-009',
b'ESP00140-010',
b'ESP00140-011',
b'ESP00140-012',
b'ESP00140-013',
b'ESP00140-014',
b'ESP00140-015',
b'ESP00140-016',
b'ESP00140-017',
b'ESP00140-018',
b'ESP00140-019',
b'ESP00140-020',
b'ESP00140-021',
b'ESP00140-022',
b'ESP00140-023',
b'ESP00140-024',
b'ESP00140-025',
b'ESP00152-001',
b'ESP00152-002',
b'ESP00152-003',
b'ESP00152-004',
b'ESP00152-005',
b'ESP00152-006',
b'ESP00152-007',
b'ESP00152-008',
b'ESP00152-009',
b'ESP00152-010',
b'ESP00152-011',
b'ESP00152-012',
b'ESP00152-013',
b'ESP00152-014',
b'ESP00152-015',
b'ESP00152-016',
b'ESP00152-017',
b'ESP00152-018',
b'ESP00152-019',
b'ESP00152-020',
```

```
b'ESP00152-021',
b'ESP00152-022',
b'ESP00152-023',
b'ESP00152-024',
b'ESP00152-025',
b'ESP00155-001',
b'ESP00155-002',
b'ESP00155-003',
b'ESP00155-004',
b'ESP00155-005',
b'ESP00155-006',
b'ESP00155-007',
b'ESP00155-008',
b'ESP00155-009',
b'ESP00155-010',
b'ESP00155-011',
b'ESP00155-012',
b'ESP00155-013',
b'ESP00155-014',
b'ESP00155-015',
b'ESP00155-016',
b'ESP00155-017',
b'ESP00155-018',
b'ESP00155-019',
b'ESP00155-020',
b'ESP00155-021',
b'ESP00155-022',
b'ESP00155-023',
b'ESP00155-024',
b'ESP00155-025',
b'ESP00221-001',
b'ESP00221-002',
b'ESP00221-003',
b'ESP00221-004',
b'ESP00221-005',
b'ESP00221-006',
b'ESP00221-007',
b'ESP00221-008',
b'ESP00221-009',
b'ESP00221-010',
b'ESP00221-011',
b'ESP00221-012',
b'ESP00221-013',
b'ESP00221-014',
b'ESP00221-015',
b'ESP00221-016',
b'ESP00221-017',
b'ESP00221-018',
b'ESP00221-019',
b'ESP00221-020',
b'ESP00221-021',
b'ESP00221-022',
b'ESP00221-023',
b'ESP00221-024',
b'ESP00221-025',
b'ESP00290-001',
b'ESP00290-002',
b'ESP00290-003',
b'ESP00290-004',
b'ESP00290-005',
```

```
b'ESP00290-006',
b'ESP00290-007',
b'ESP00290-008',
b'ESP00290-009',
b'ESP00290-010',
b'ESP00290-011',
b'ESP00290-012',
b'ESP00290-013',
b'ESP00290-014',
b'ESP00290-015',
b'ESP00290-016',
b'ESP00290-017',
b'ESP00290-018',
b'ESP00290-019',
b'ESP00290-020',
b'ESP00290-021',
b'ESP00290-022',
b'ESP00290-023',
b'ESP00290-024',
b'ESP00290-025',
b'ESP00300-001',
b'ESP00300-002',
b'ESP00300-003',
b'ESP00300-004',
b'ESP00300-005',
b'ESP00300-006',
b'ESP00300-007',
b'ESP00300-008',
b'ESP00300-009',
b'ESP00300-010',
b'ESP00300-011',
b'ESP00300-012',
b'ESP00300-013',
b'ESP00300-014',
b'ESP00300-015',
b'ESP00300-016',
b'ESP00300-017',
b'ESP00300-018',
b'ESP00300-019',
b'ESP00300-020',
b'ESP00300-021',
b'ESP00300-022',
b'ESP00300-023',
b'ESP00300-024',
b'ESP00300-025',
b'ESP00348-001',
b'ESP00348-002',
b'ESP00348-003',
b'ESP00348-004',
b'ESP00348-005',
b'ESP00348-006',
b'ESP00348-007',
b'ESP00348-008',
b'ESP00348-009',
b'ESP00348-010',
b'ESP00348-011',
b'ESP00348-012',
b'ESP00348-013',
b'ESP00348-014',
b'ESP00348-015',
```

```
b'ESP00348-016',
b'ESP00348-017',
b'ESP00348-018',
b'ESP00348-019',
b'ESP00348-020',
b'ESP00348-021',
b'ESP00348-022',
b'ESP00348-023',
b'ESP00348-024',
b'ESP00348-025',
b'ITA00043-001',
b'ITA00043-002',
b'ITA00043-003',
b'ITA00043-004',
b'ITA00043-005',
b'ITA00043-006',
b'ITA00043-007',
b'ITA00043-008',
b'ITA00043-009',
b'ITA00043-010',
b'ITA00043-011',
b'ITA00043-012',
b'ITA00043-013',
b'ITA00043-014',
b'ITA00043-015',
b'ITA00043-016',
b'ITA00043-017',
b'ITA00043-018',
b'ITA00043-019',
b'ITA00043-020',
b'ITA00043-021',
b'ITA00043-022',
b'ITA00043-023',
b'ITA00043-024',
b'ITA00043-025',
b'ITA00105-101',
b'ITA00105-102',
b'ITA00105-103',
b'ITA00105-104',
b'ITA00105-105',
b'ITA00105-106',
b'ITA00105-107',
b'ITA00105-108',
b'ITA00105-109',
b'ITA00105-110',
b'ITA00105-111',
b'ITA00105-112',
b'ITA00105-113',
b'ITA00105-114',
b'ITA00105-115',
b'ITA00105-116',
b'ITA00105-117',
b'ITA00105-118',
b'ITA00105-119',
b'ITA00105-120',
b'ITA00105-121',
b'ITA00105-122',
b'ITA00105-123',
b'ITA00105-124',
b'ITA00105-125',
```

```
 b'SVN00059-001',
 b'SVN00059-002',
 b'SVN00059-003',
 b'SVN00059-004',
 b'SVN00059-005',
 b'SVN00059-006',
 b'SVN00059-007',
 b'SVN00059-008',
 b'SVN00059-009',
 b'SVN00059-010',
 b'SVN00059-011',
 b'SVN00059-012',
 b'SVN00059-013',
 b'SVN00059-014',
 b'SVN00059-015',
 b'SVN00059-016',
 b'SVN00059-017',
 b'SVN00059-018',
 b'SVN00059-019',
 b'SVN00059-020',
 b'SVN00059-021']
```

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

Out[61]:

	ID	Population
0	ESP00053-001	ESP00053
1	ESP00053-002	ESP00053
2	ESP00053-003	ESP00053
3	ESP00053-004	ESP00053
4	ESP00053-005	ESP00053
...	...	...
316	SVN00059-017	SVN00059
317	SVN00059-018	SVN00059
318	SVN00059-019	SVN00059
319	SVN00059-020	SVN00059
320	SVN00059-021	SVN00059

321 rows × 2 columns

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

Out[62]:
```
Population
ESP00053    25
ESP00058    25
ESP00061    25
ESP00140    25
ESP00152    25
ESP00155    25
ESP00221    25
ESP00290    25
ESP00300    25
ESP00348    25
ITA00043    25
ITA00105    25
SVN00059    21
Name: count, dtype: int64
```

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

Out[63]:
```
array(['ESP00053', 'ESP00058', 'ESP00061', 'ESP00140', 'ESP00152',
       'ESP00155', 'ESP00221', 'ESP00290', 'ESP00300', 'ESP00348',
       'ITA00043', 'ITA00105', 'SVN00059'], dtype=object)
```

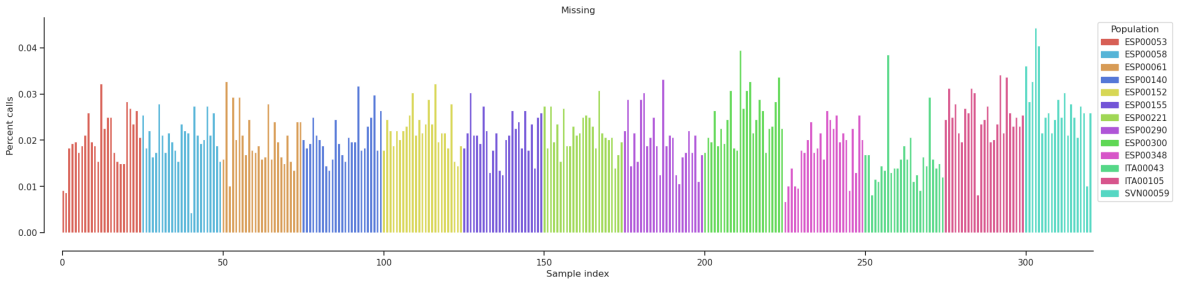
Gt frequency function

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", 13)
    pop2color = {'ESP00053': palette[0],
                 'ESP00058': palette[7],
                 'ESP00061': palette[1],
                 'ESP00140': palette[8],
                 'ESP00152': palette[2],
                 'ESP00155': palette[9],
                 'ESP00221': palette[3],
                 'ESP00290': palette[10],
                 'ESP00300': palette[4],
                 'ESP00348': palette[11],
                 'ITA00043': palette[5],
                 'ITA00105': palette[12],
                 'SVN00059': palette[6]}
    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])]
    ax.legend(handles=handles, labels=['ESP00053', 'ESP00058', 'ESP00061'
       'ESP00155', 'ESP00221', 'ESP00290', 'ESP00300', 'ESP00348',
     'ITA00043', 'ITA00105', 'SVN00059'], title='Population',
              bbox_to_anchor=(1, 1), loc='upper left')
```

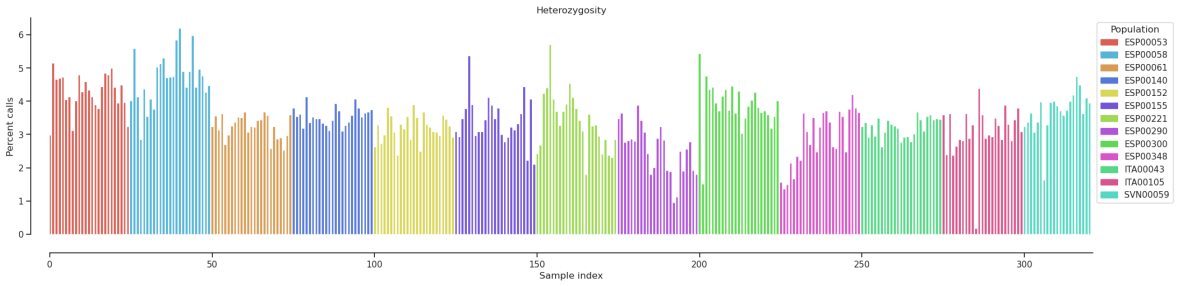
Plot missing

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



Plot heterozygosity

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



PCA

In [70]:
```
palette = sns.color_palette("hls",13)
pop_colours = {
                'ESP00053': palette[0],
                 'ESP00058': palette[7],
                 'ESP00061': palette[1],
                 'ESP00140': palette[8],
                 'ESP00152': palette[2],
                 'ESP00155': palette[9],
                 'ESP00221': palette[3],
                 'ESP00290': palette[10],
                 'ESP00300': palette[4],
                 'ESP00348': palette[11],
                 'ITA00043': palette[5],
                 'ITA00105': palette[12],
                 'SVN00059': palette[6]
}
```

In [71]:
```
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 [72]:
```
ac2 = gt_biallelic.count_alleles()
ac2
```

Out[72]: <AlleleCountsChunkedArray shape=(207741, 2) dtype=int32 chunks=(51936, 2) nbytes=1.6M cbytes=388.4K cratio=4.2 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1
0	641	1
1	640	2
2	641	1
...	...	
207738	641	1
207739	641	1
207740	641	1

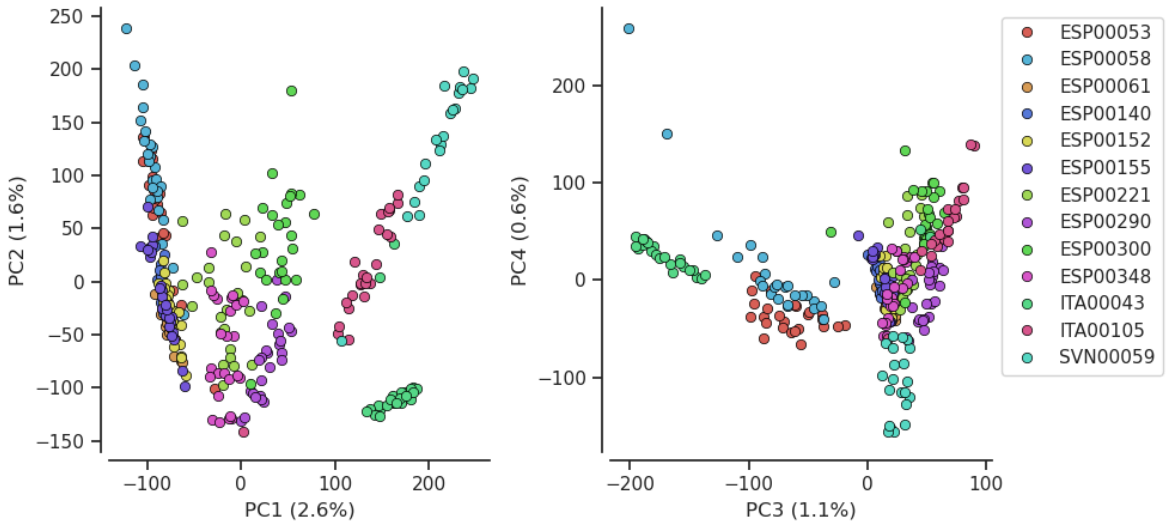
In [73]:
```
flt = (ac2[:, :2].min(axis=1) > 1)
gf = gt_biallelic.compress(flt, axis=0)
gn = gf.to_n_alt()
gn
```

Out[73]:
```
<ChunkedArrayWrapper shape=(152601, 321) dtype=int8 chunks=(2385, 321)
   nbytes=46.7M cbytes=6.5M cratio=7.2
   compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shu
ffle': 1, 'blocksize': 0}
   values=zarr.core.Array>
```

In [74]:
```
coords1, model1 = allel.pca(gn, n_components=10, scaler='patterson')
```

In [75]:
```
fig_pca(coords1, model1, 'Figure 1. Conventional PCA.')
```

Figure 1. Conventional PCA.



In [ ]:
```

```