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

In [33]:
```
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/Aalba/vcf_filtering/ra
```

Get data

In [34]:
```
callset_var_fn = '/users/mcevoysu/scratch/output/Aalba/scikit-allel/raw_S
callset_var = h5py.File(callset_var_fn, mode='r')
```

In [35]:
```
calldata_var = callset_var['calldata']
list(calldata_var)
```

Out[35]:
```
['AD', 'DP', 'GQ', 'GT', 'MIN_DP', 'PGT', 'PID', 'PL', 'PS', 'RGQ', 'S
B']
```

In [36]:
```
list(callset_var['variants'])
```

Out[36]:
```
['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 [37]:
```
variants = allel.VariantChunkedTable(callset_var['variants'])
variants
```

Out[37]: <VariantChunkedTable shape=(216087,) 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=36.9M cbytes=7.9M cratio=4.7 values=h5py._hl.group.Group>

	AC	AF	ALT	AN	BaseQRankSum	CHROM	DP	E
0	[2 -1 -1]	[0.002315 nan nan]	[b'C' b'' b'']	864	-1.039	b'aalba5_s00000025'	14531	
1	[1 -1 -1]	[0.001157 nan nan]	[b'*' b'' b'']	864	nan	b'aalba5_s00000025'	12677	
2	[2 -1 -1]	[0.002315 nan nan]	[b'C' b'' b'']	864	0.319	b'aalba5_s00000025'	12506	
...								
216084	[2 -1 -1]	[0.002315 nan nan]	[b'A' b'' b'']	864	0.0	b'aalba5_s00422950'	703	
216085	[2 -1 -1]	[0.002315 nan nan]	[b'G' b'' b'']	864	nan	b'aalba5_s00422950'	508	
216086	[2 -1 -1]	[0.002315 nan nan]	[b'C' b'' b'']	864	nan	b'aalba5_s00422950'	218	

In [38]:
```
variants_np = variants[:]
rawsnps = variants_np.query('(is_snp == True)')
rawsnps
```

Out[38]: <VariantTable shape=(141944,) 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	E
0	[2 -1 -1]	[0.002315 nan nan]	[b'C' b'' b'']	864	-1.039	b'aalba5_s00000025'	14531	
1	[2 -1 -1]	[0.002315 nan nan]	[b'C' b'' b'']	864	0.319	b'aalba5_s00000025'	12506	
2	[14 -1 -1]	[0.016 nan nan]	[b'T' b'' b'']	864	0.214	b'aalba5_s00000025'	12170	
...								
141941	[2 -1 -1]	[0.002315 nan nan]	[b'A' b'' b'']	864	0.0	b'aalba5_s00422950'	703	
141942	[2 -1 -1]	[0.002315 nan nan]	[b'G' b'' b'']	864	nan	b'aalba5_s00422950'	508	
141943	[2 -1 -1]	[0.002315 nan nan]	[b'C' b'' b'']	864	nan	b'aalba5_s00422950'	218	

In [39]:
```
notsnp = variants_np.query('(is_snp != True)')
notsnp
```

Out[39]: <VariantTable shape=(74143,) 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	E
0	[1 -1 -1]	[0.001157 nan nan]	[b'*' b'' b'']	864	nan	b'aalba5_s00000025'	12677	
1	[28 2 -1]	[0.032 0.002315 nan]	[b'C' b'*' b'']	864	0.423	b'aalba5_s00000025'	11564	
2	[1 -1 -1]	[0.001157 nan nan]	[b'*' b'' b'']	864	nan	b'aalba5_s00000025'	2513	
...								
74140	[260 -1 -1]	[0.301 nan nan]	[b'*' b'' b'']	864	nan	b'aalba5_s00422183'	995	
74141	[1 -1 -1]	[0.001157 nan nan]	[b'*' b'' b'']	864	nan	b'aalba5_s00422584'	2514	
74142	[1 -1 -1]	[0.001157 nan nan]	[b'*' b'' b'']	864	nan	b'aalba5_s00422950'	7438	

Plot function

In [40]:
```
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 [41]:
```
numalt = rawsnps['numalt']
np.max(numalt)
```

Out[41]:
```
3
```

In [42]:
```
count_numalt = np.bincount(numalt)
count_numalt
```

Out[42]:
```
array([     0, 138760,   3117,     67])
```

In [43]:
```
n_multiallelic = np.sum(count_numalt[2:])
n_multiallelic
```

Out[43]:
```
3184
```

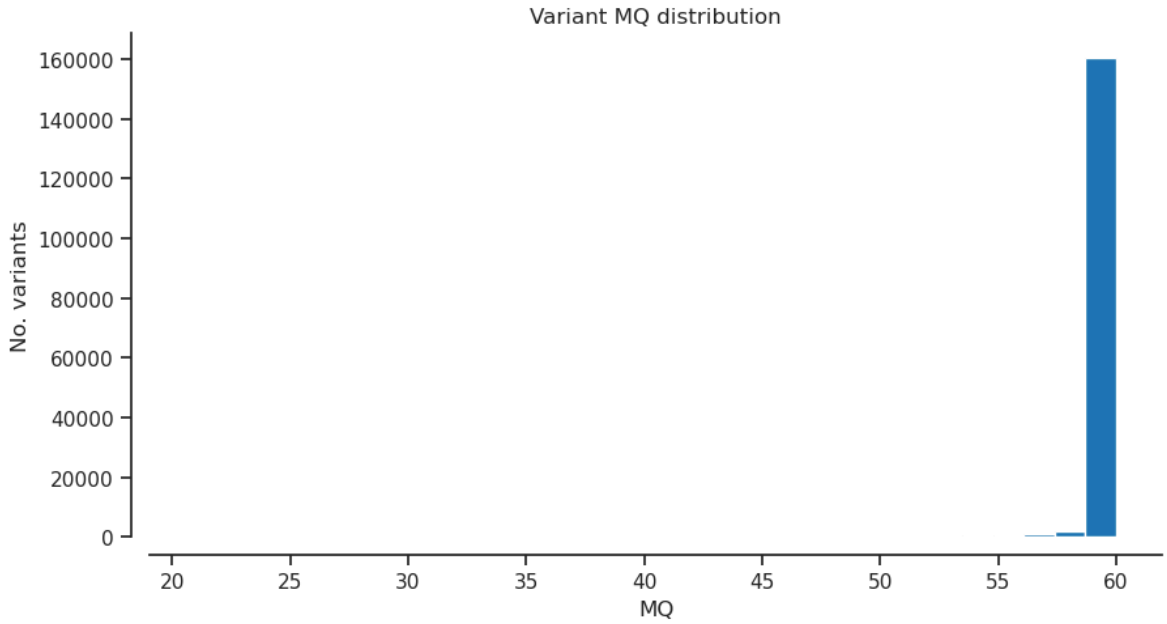
In [44]:
```
filter_expression = '(numalt == 1)'
biallelic_np = rawsnps.query(filter_expression)[:]
biallelic_np
```

Out[44]: <VariantTable shape=(138760,) 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')])>

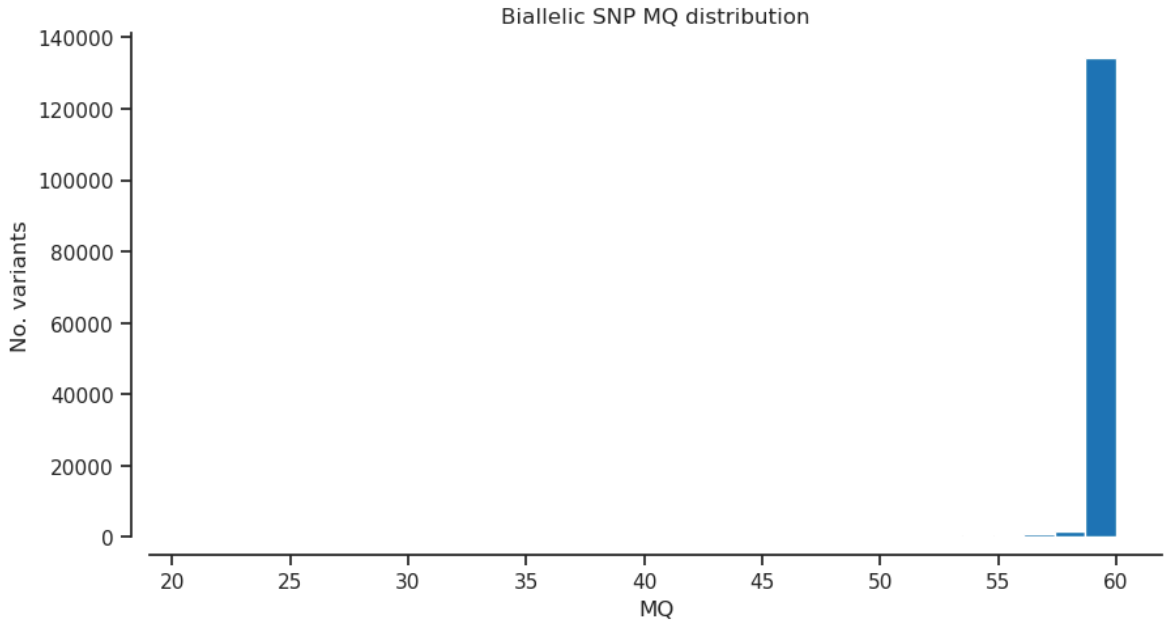
	AC	AF	ALT	AN	BaseQRankSum	CHROM	DP	E
0	[2 -1 -1]	[0.002315 nan nan]	[b'C' b'' b'']	864	-1.039	b'aalba5_s00000025'	14531	
1	[2 -1 -1]	[0.002315 nan nan]	[b'C' b'' b'']	864	0.319	b'aalba5_s00000025'	12506	
2	[14 -1 -1]	[0.016 nan nan]	[b'T' b'' b'']	864	0.214	b'aalba5_s00000025'	12170	
...								
138757	[2 -1 -1]	[0.002315 nan nan]	[b'A' b'' b'']	864	0.0	b'aalba5_s00422950'	703	
138758	[2 -1 -1]	[0.002315 nan nan]	[b'G' b'' b'']	864	nan	b'aalba5_s00422950'	508	
138759	[2 -1 -1]	[0.002315 nan nan]	[b'C' b'' b'']	864	nan	b'aalba5_s00422950'	218	

MQ - RMS mapping quality

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

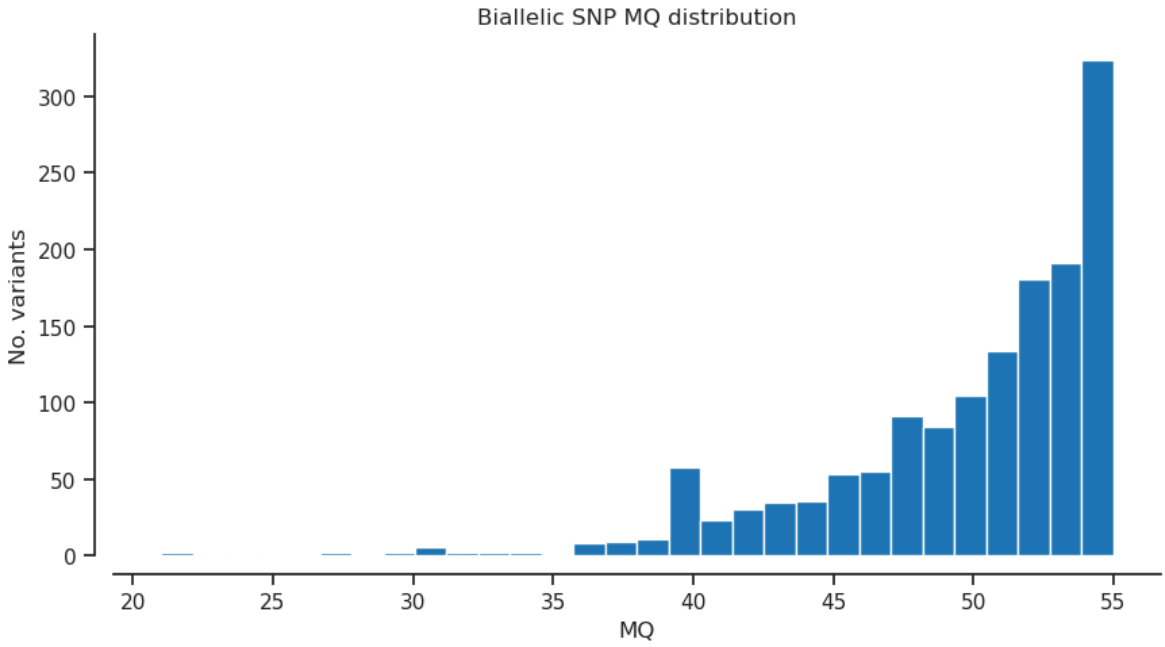


In [15]:
```
plot_hist('MQ','biallelic') # RMS mapping quality
```



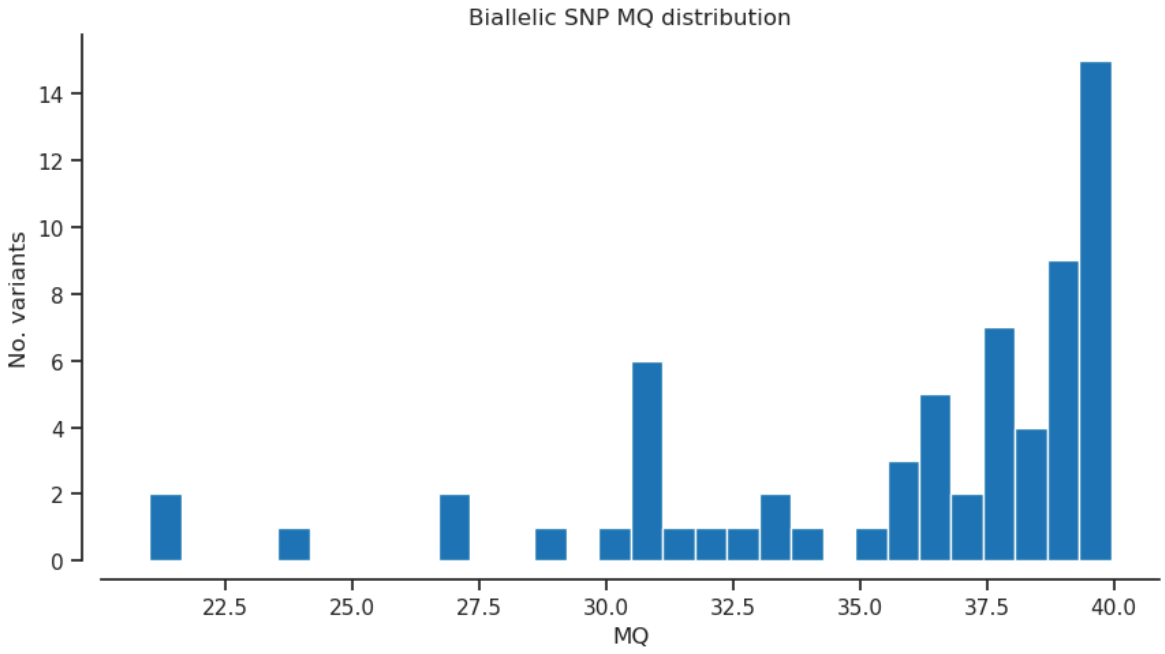
In [16]:
```
filter_expression = '(MQ < 55)'
bi_selection = biallelic_np.query(filter_expression)[:]
#np.count_nonzero(var_selection)
```

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

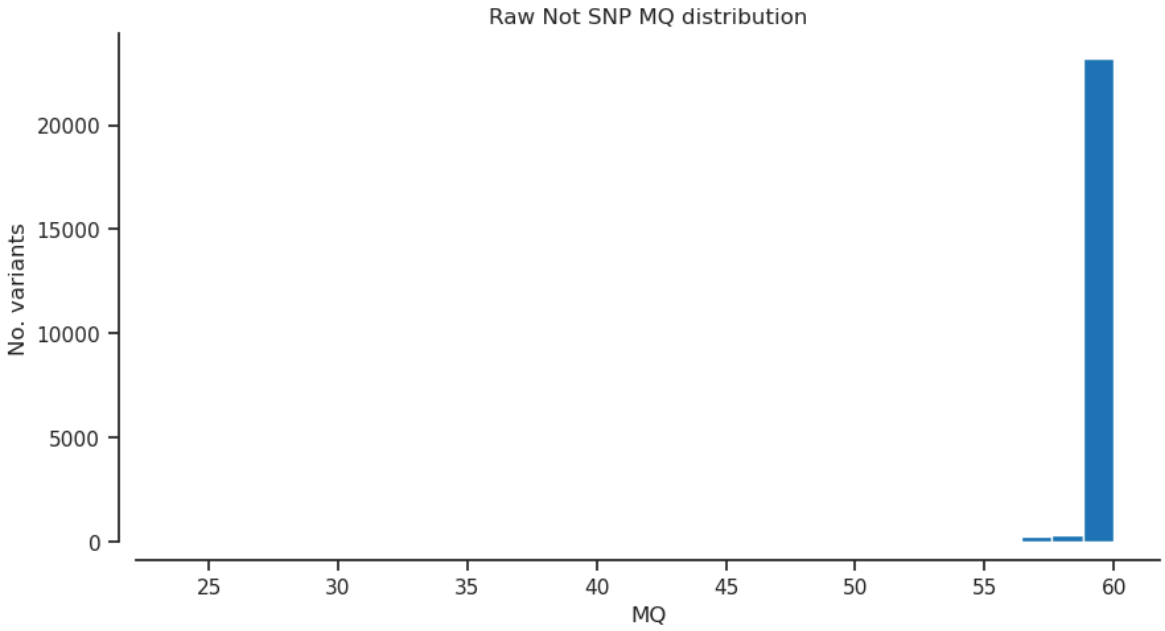


In [18]:
```
filter_expression = '(MQ < 40)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

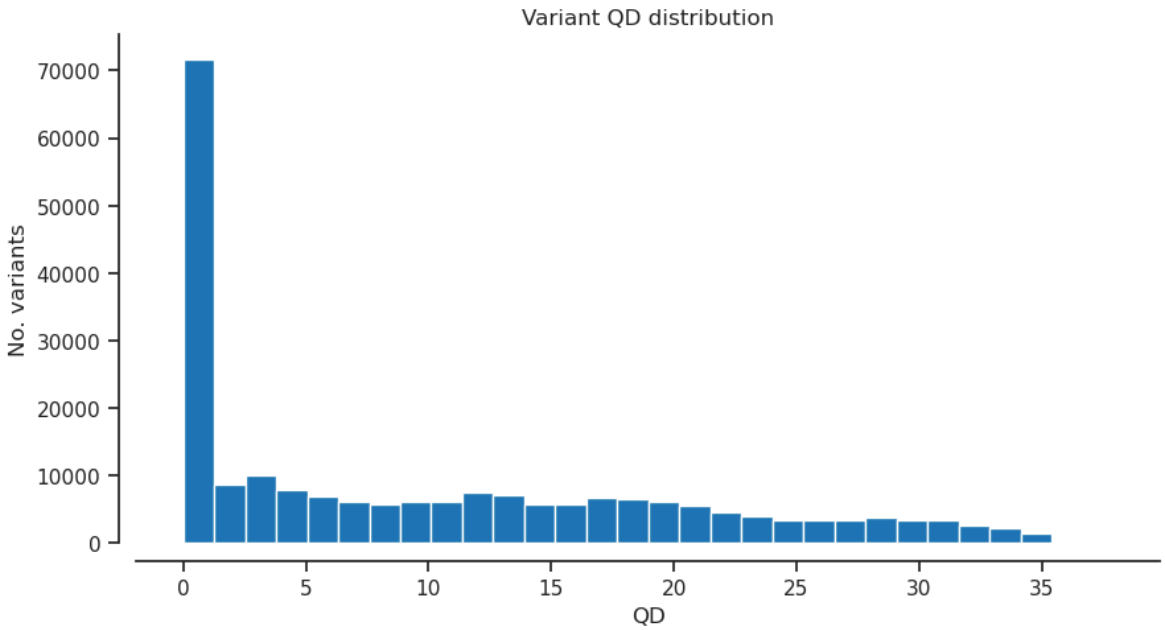


In [20]:
```
plot_hist('MQ','notsnp')
```

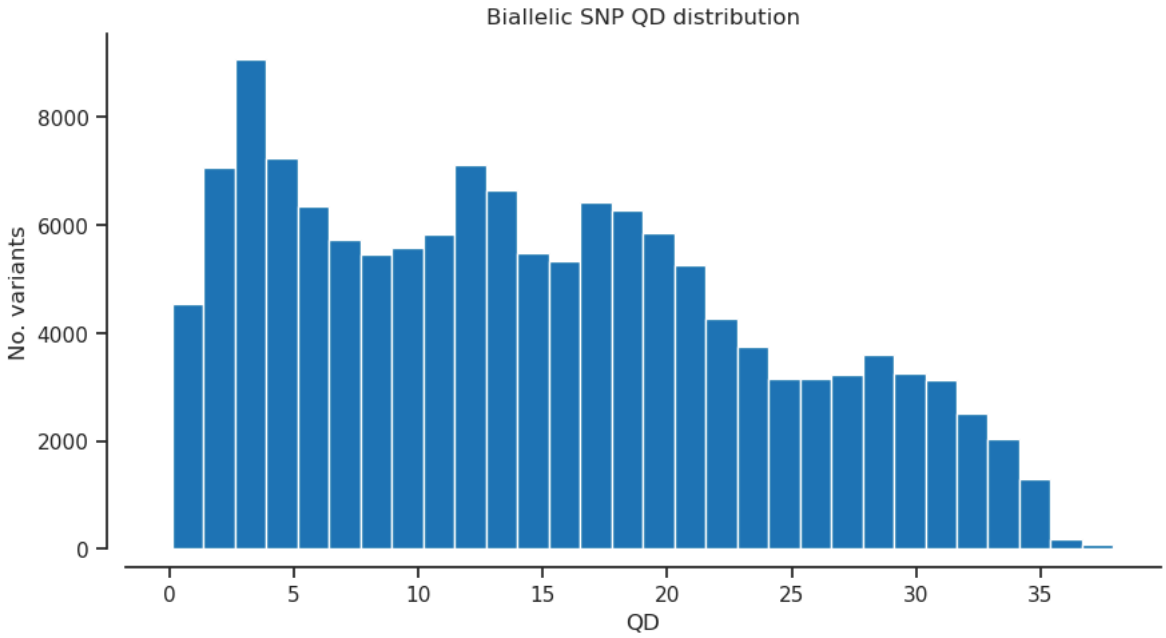


QD - Variant Confidence/Quality by Depth

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

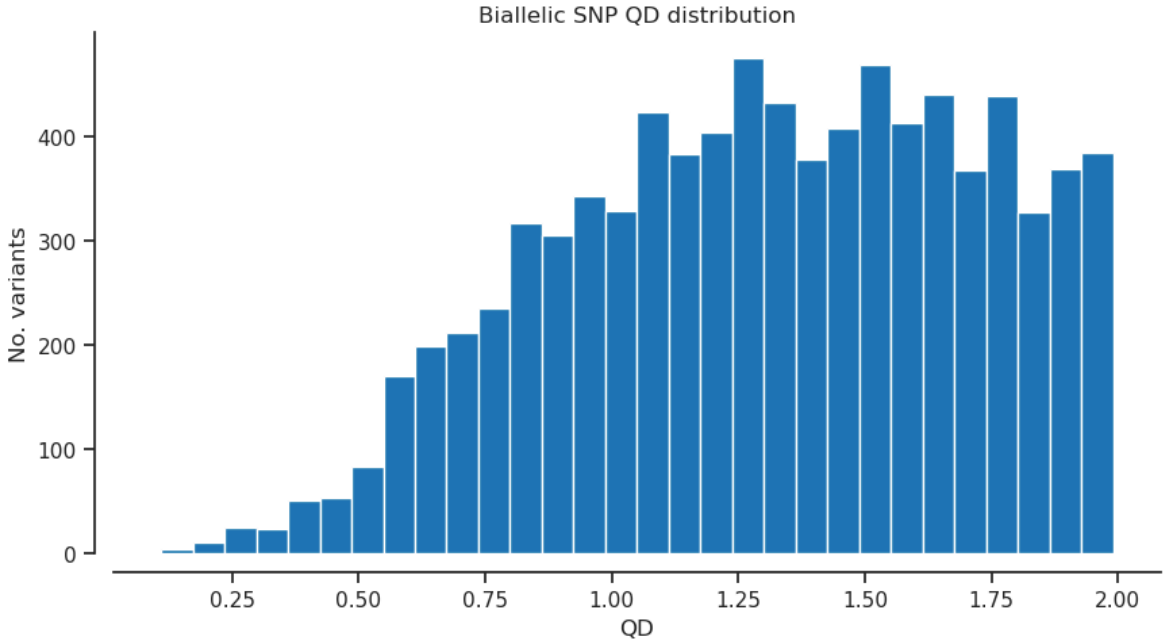


In [22]:
```
plot_hist('QD','biallelic') # Variant Confidence/Quality by Depth
```

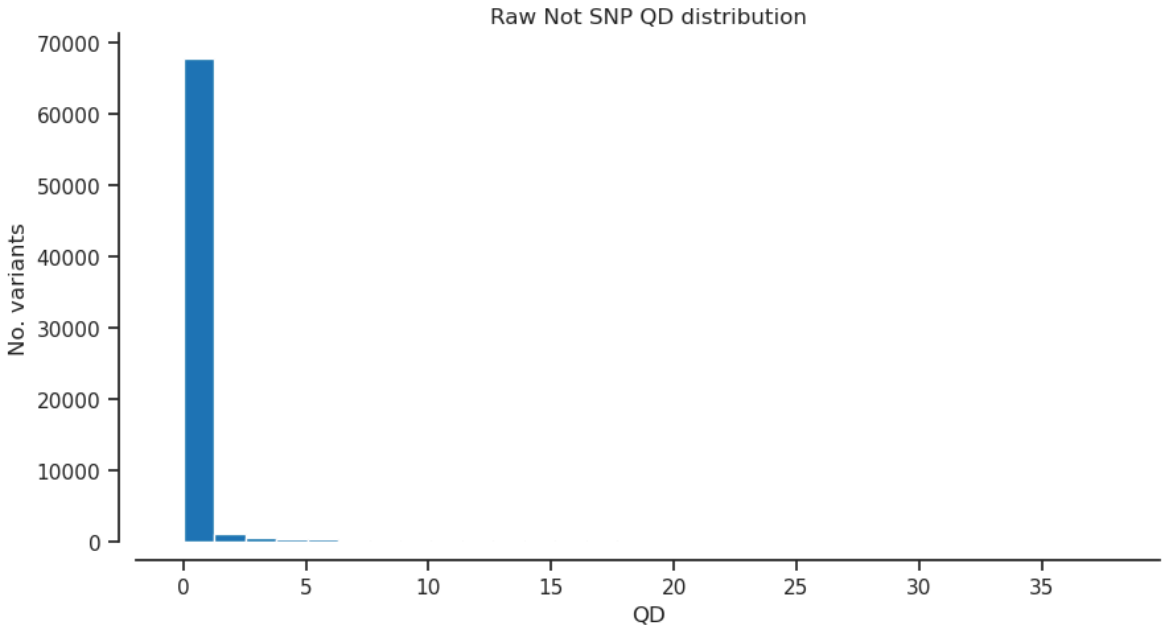


In [23]:
```
filter_expression = '(QD < 2)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

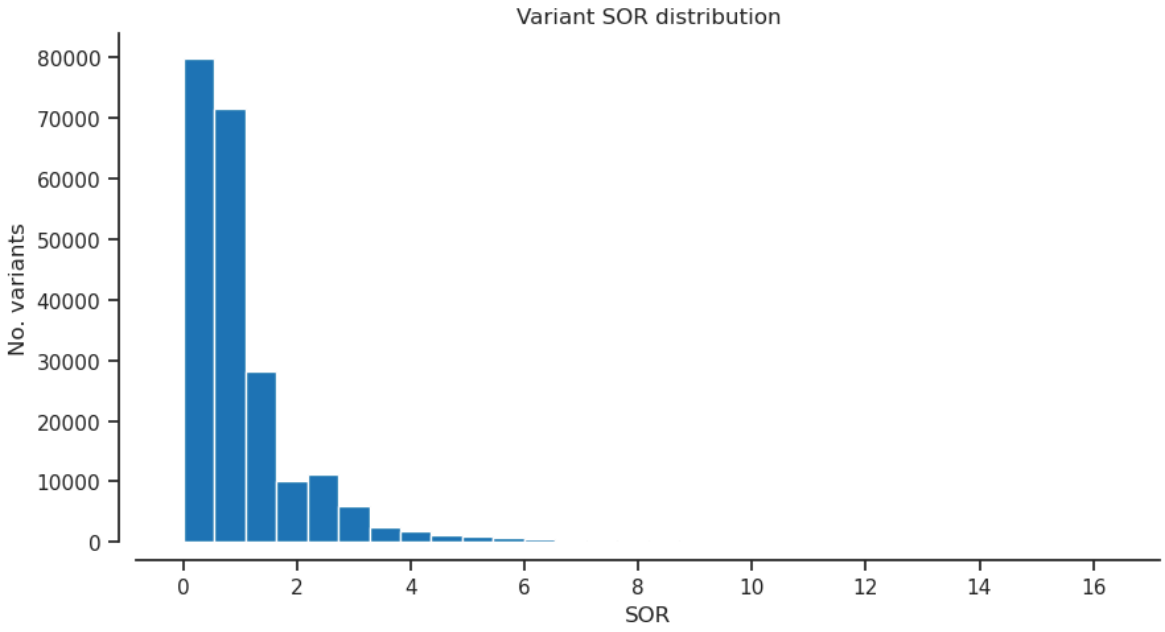


In [25]:
```
plot_hist('QD','notsnp') # Variant Confidence/Quality by Depth
```

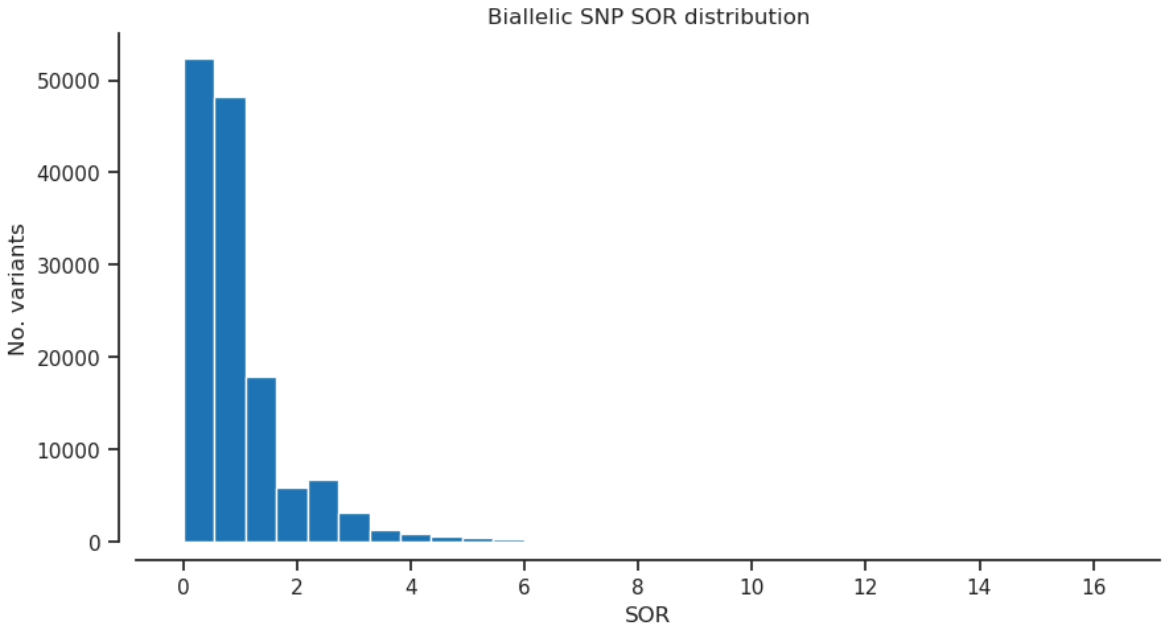


SOR - Symmetric Odds Ratio of 2x2 contingency table to detect strand bias

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

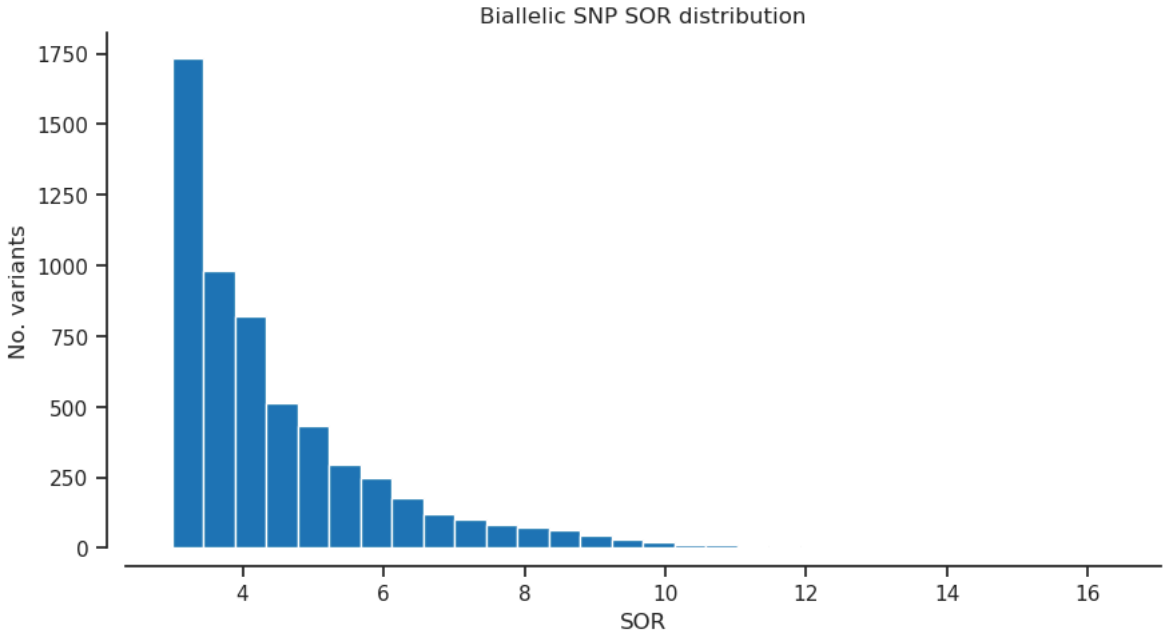


In [27]:
```
plot_hist('SOR','biallelic') # Symmetric Odds Ratio of 2x2 contingency ta
```

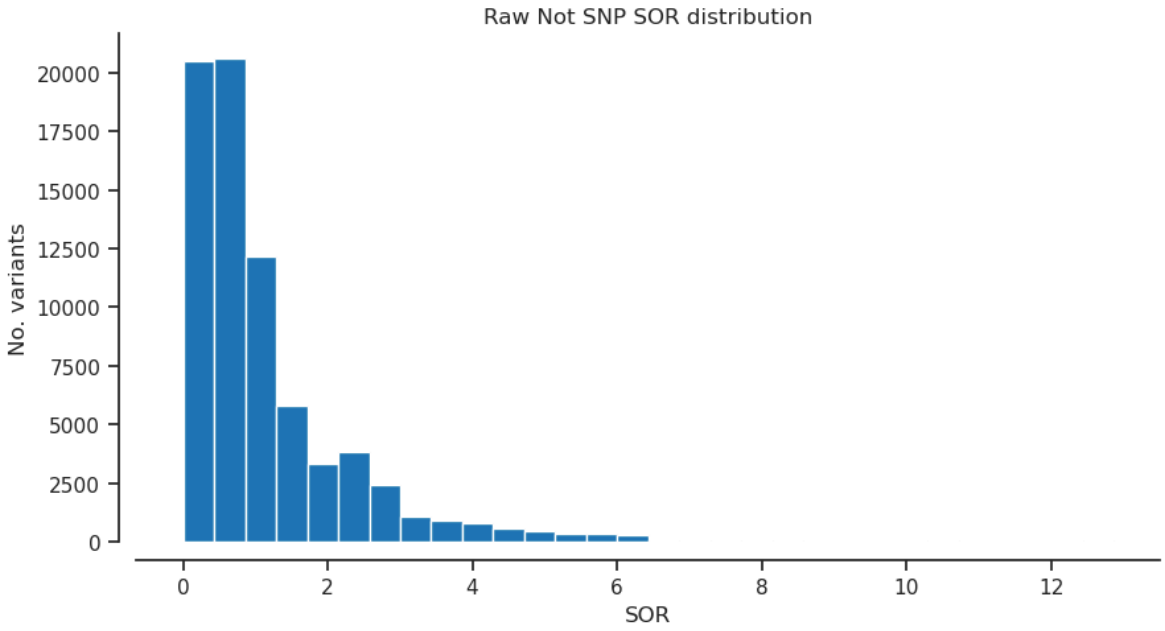


In [28]:
```
filter_expression = '(SOR > 3)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

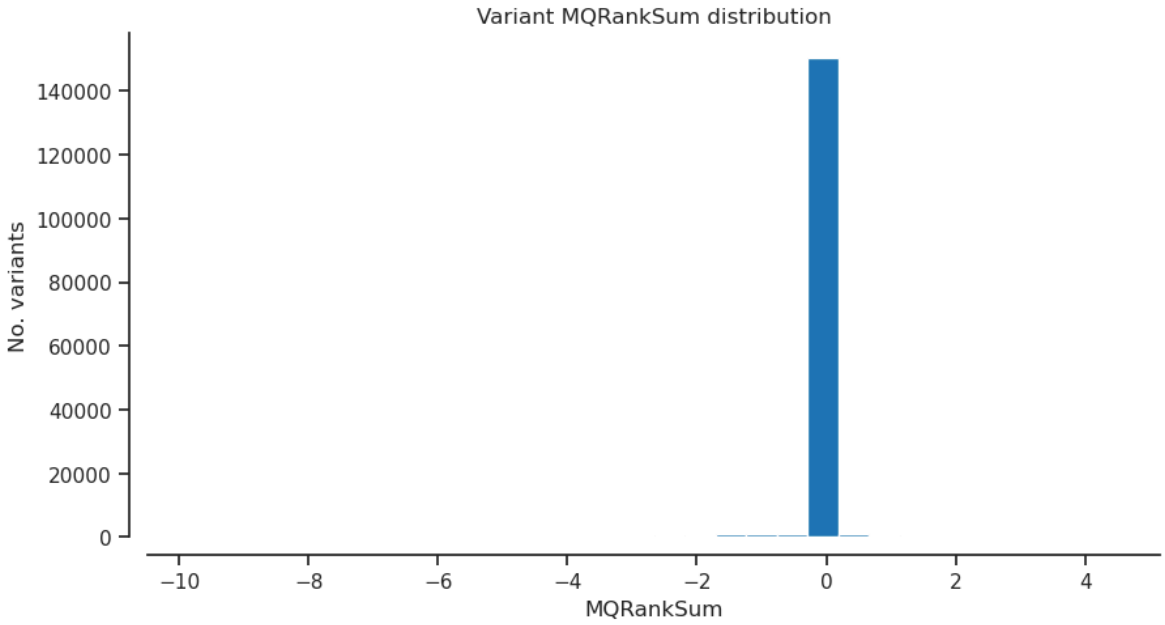


In [30]:
```
plot_hist('SOR','notsnp') # Symmetric Odds Ratio of 2x2 contingency table
```

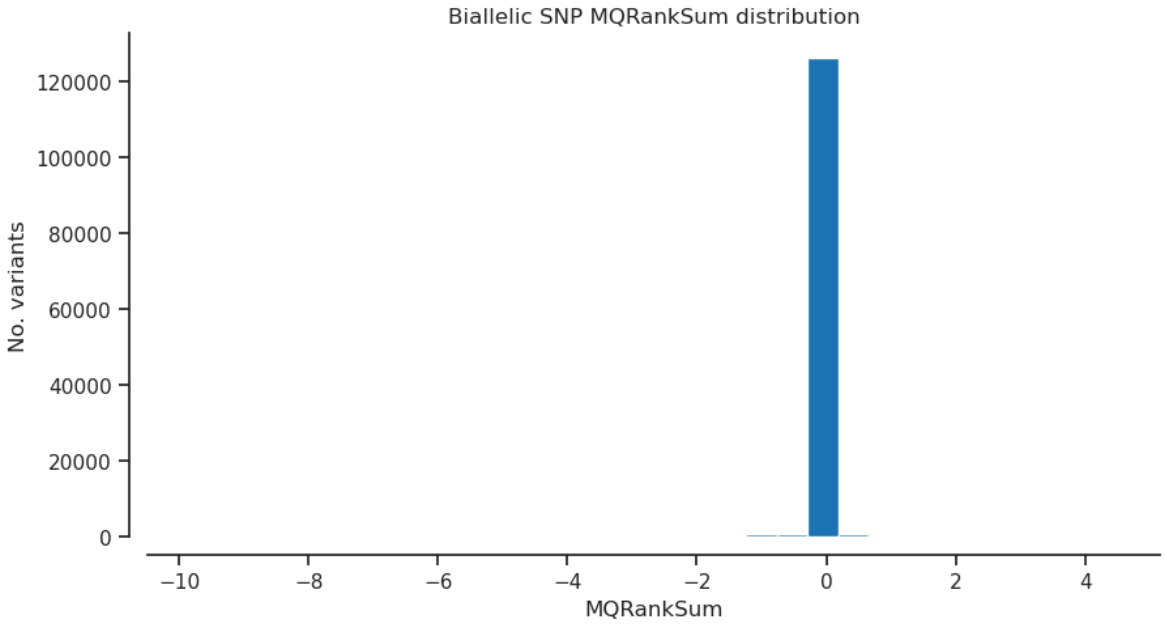


MQRankSum - Z-score From Wilcoxon rank sum test of Alt vs. Ref read mapping qualities

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

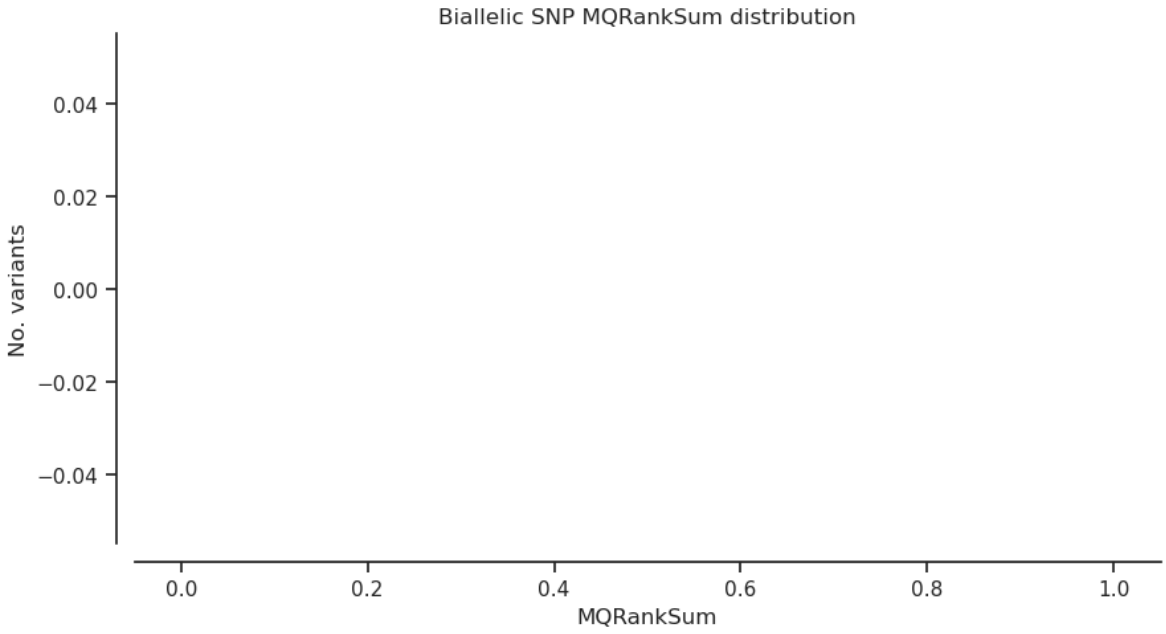


In [32]:
```
plot_hist('MQRankSum','biallelic') # Z-score From Wilcoxon rank sum test
```

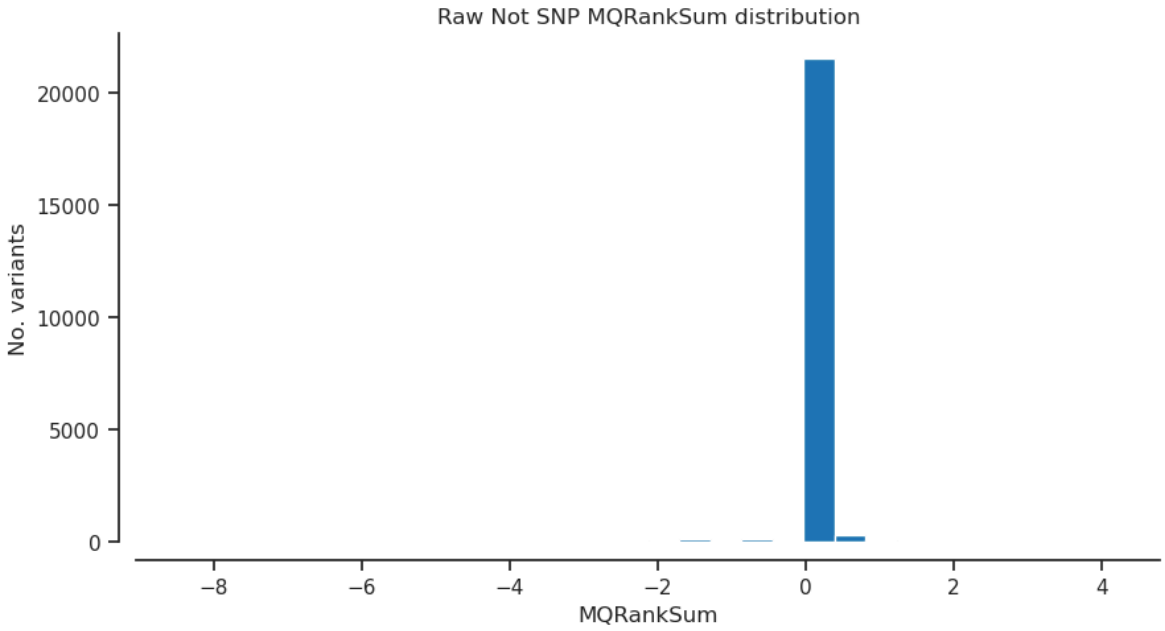


In [33]:
```
filter_expression = '(MQRankSum < -12.5)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

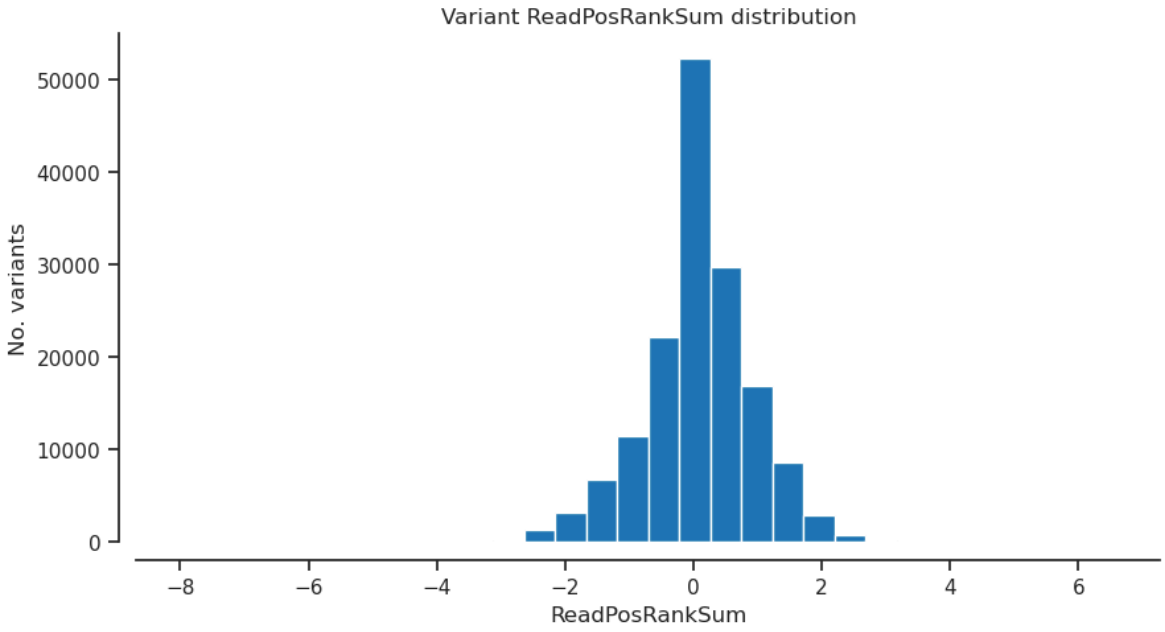


In [35]:
```
plot_hist('MQRankSum','notsnp') # Z-score From Wilcoxon rank sum test of
```

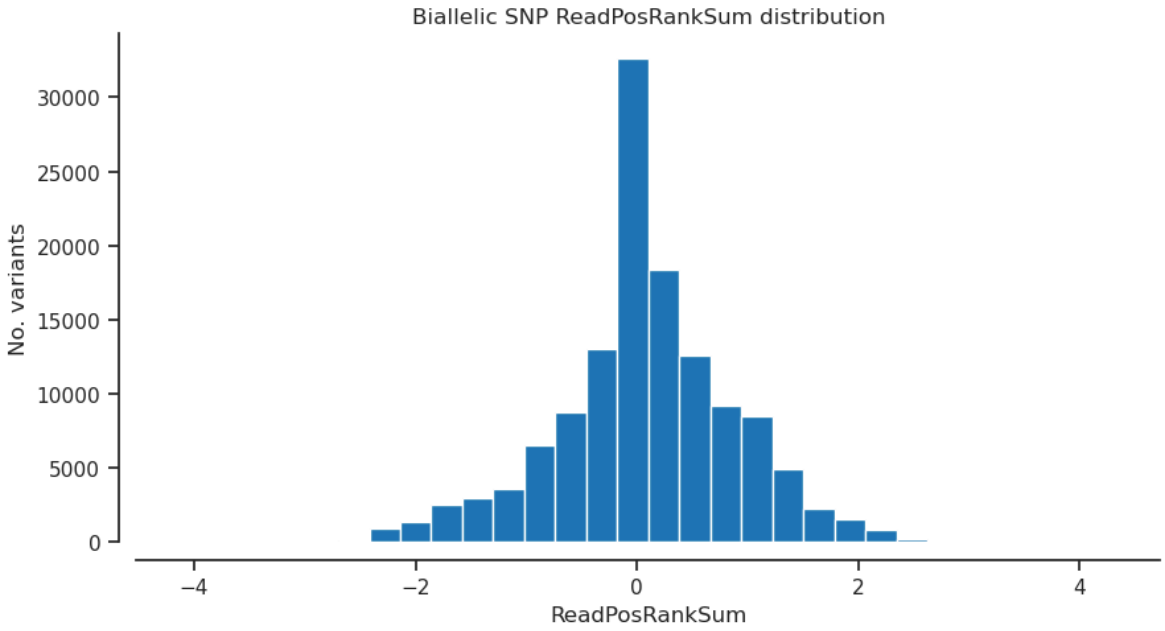


ReadPosRankSum - Z-score from Wilcoxon rank sum test of Alt vs. Ref read position bias

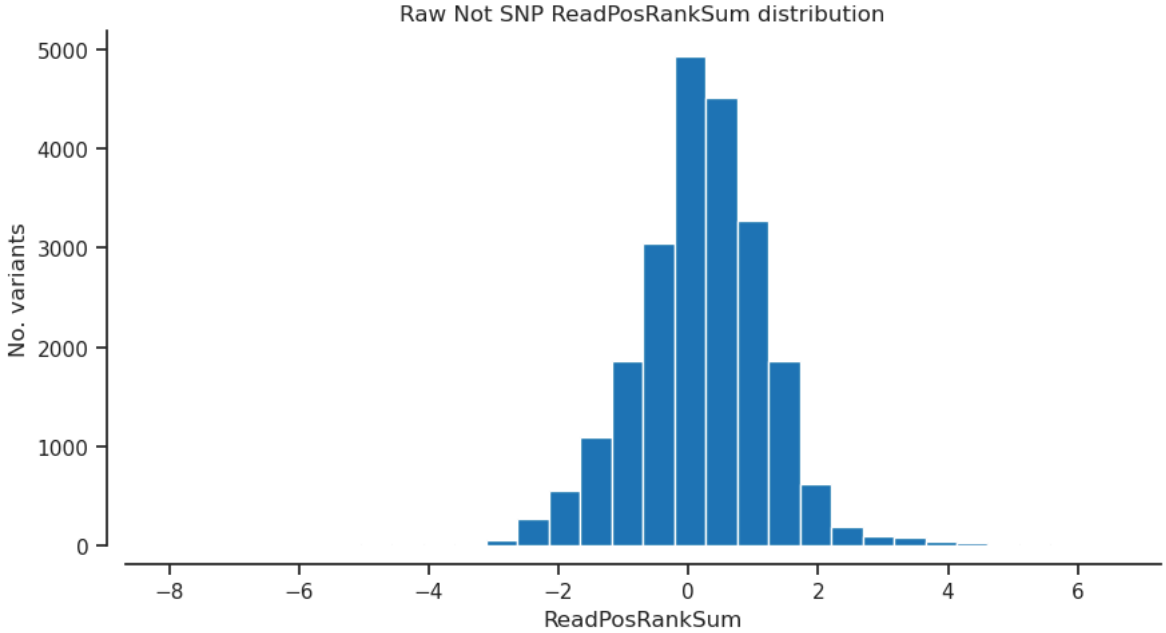
In [36]:
```
plot_hist('ReadPosRankSum','var') # Z-score from Wilcoxon rank sum test o
```



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

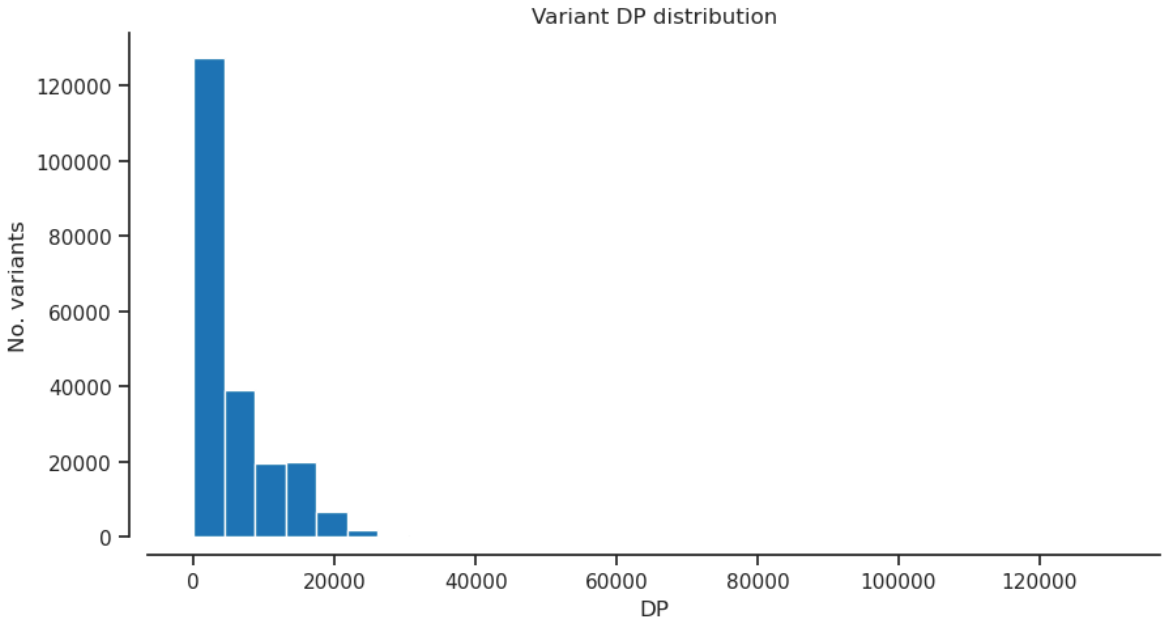


In [38]:
```
plot_hist('ReadPosRankSum','notsnp') # Z-score from Wilcoxon rank sum tes
```

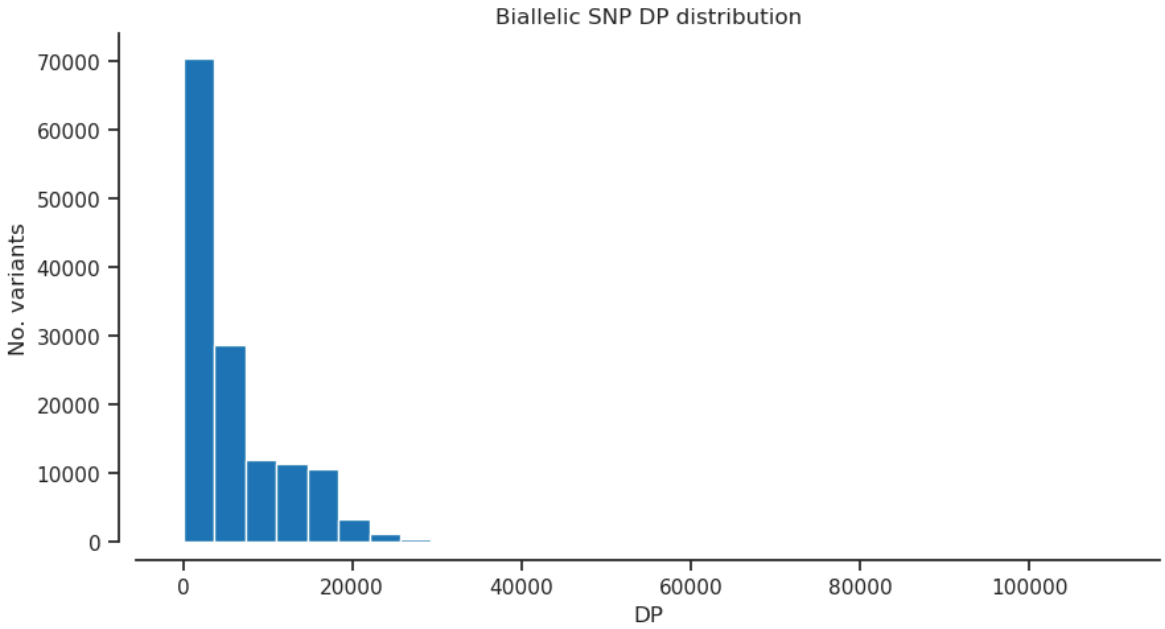


DP - Approximate read depth

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

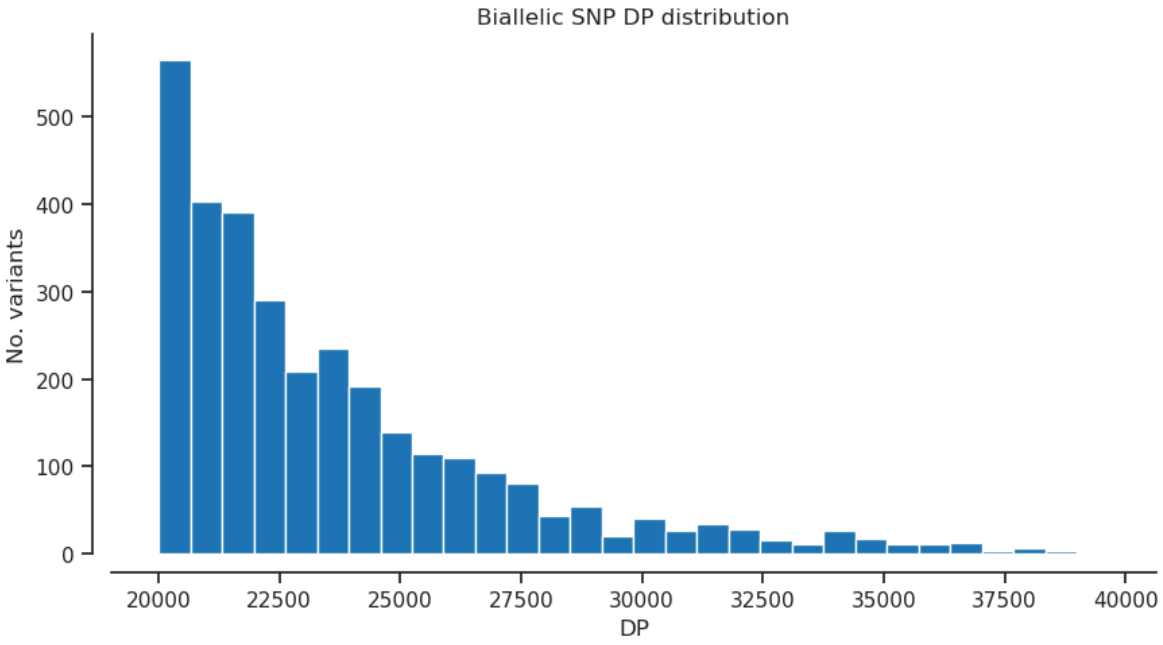


In [40]:
```
plot_hist('DP','biallelic')
```

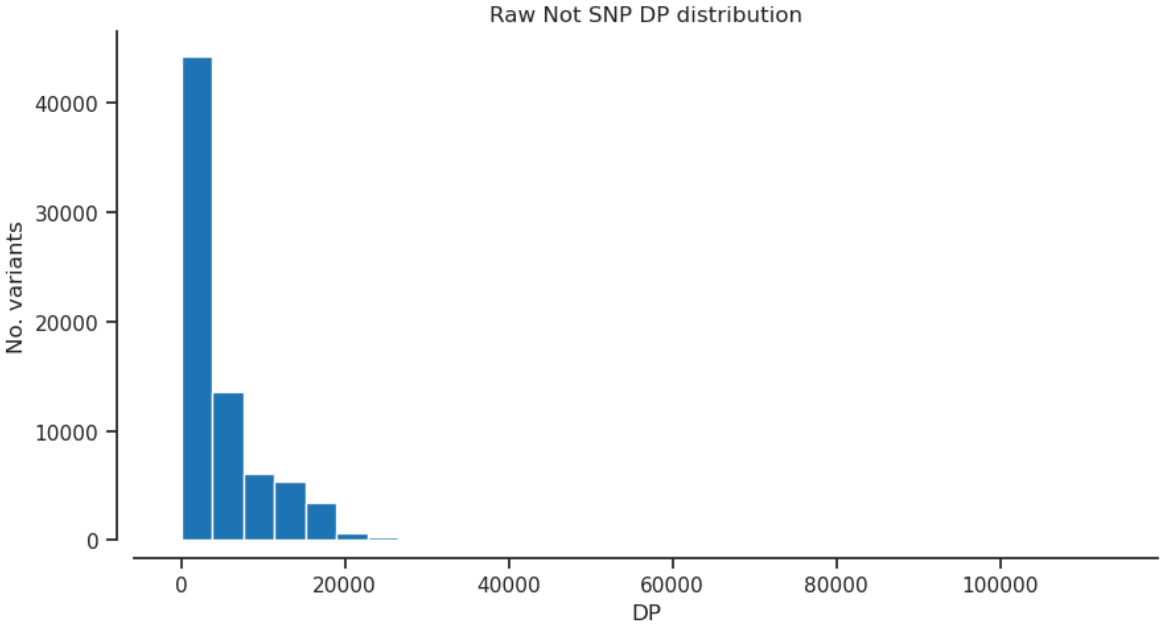


In [41]:
```
filter_expression = '(DP > 20000) & (DP < 40000)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

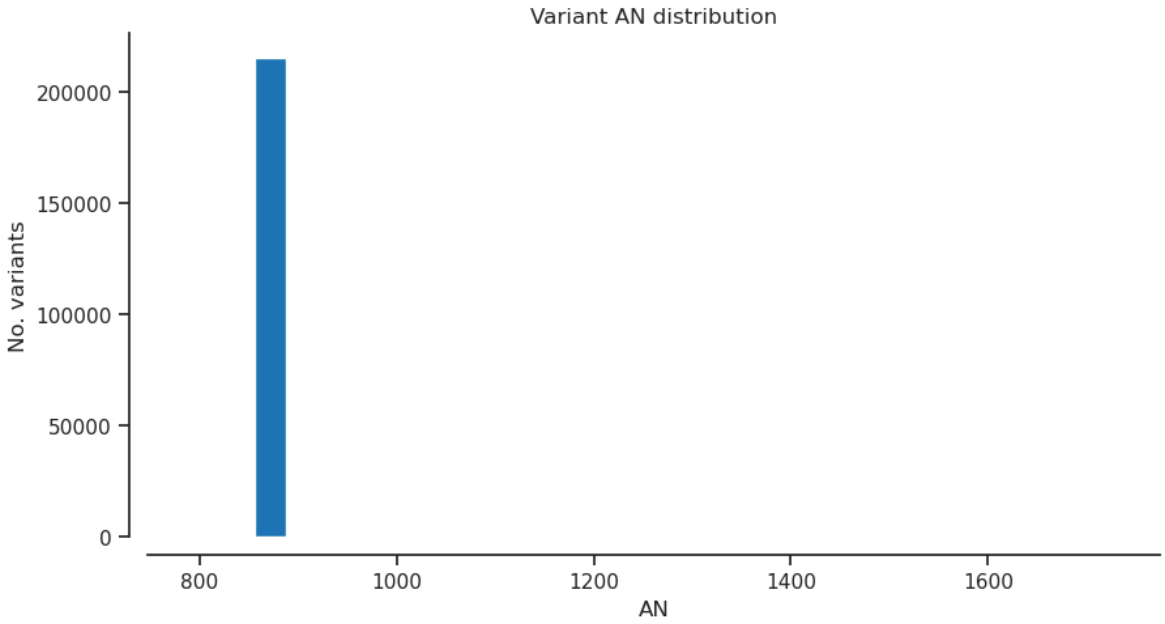


In [43]:
```
plot_hist('DP','notsnp')
```

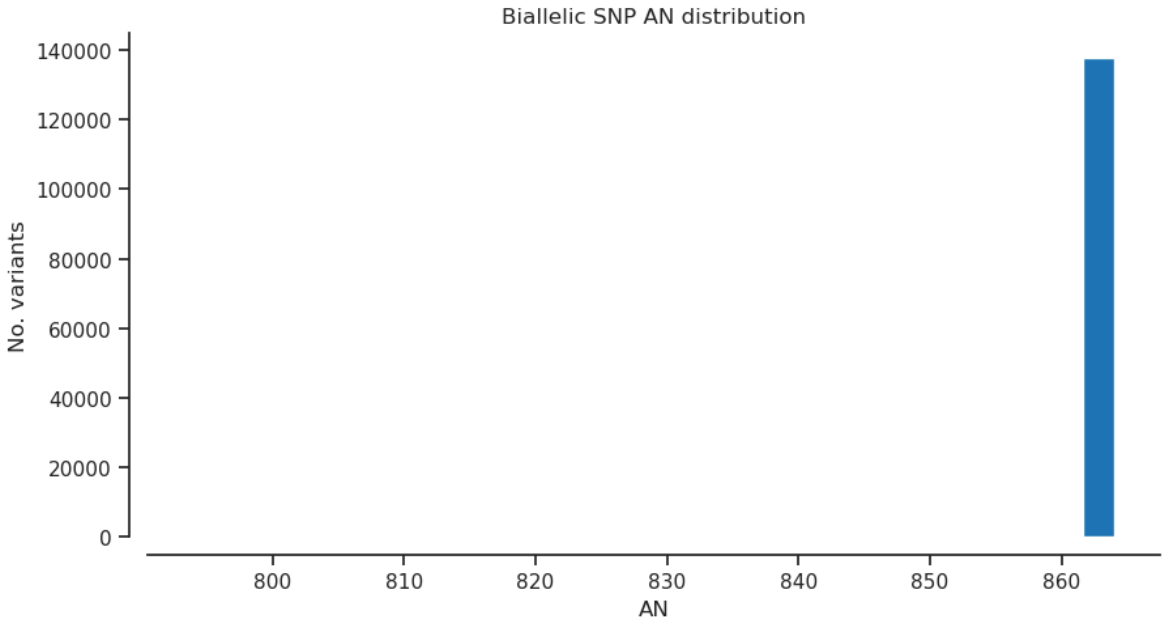


AN - Total number of alleles in called genotypes

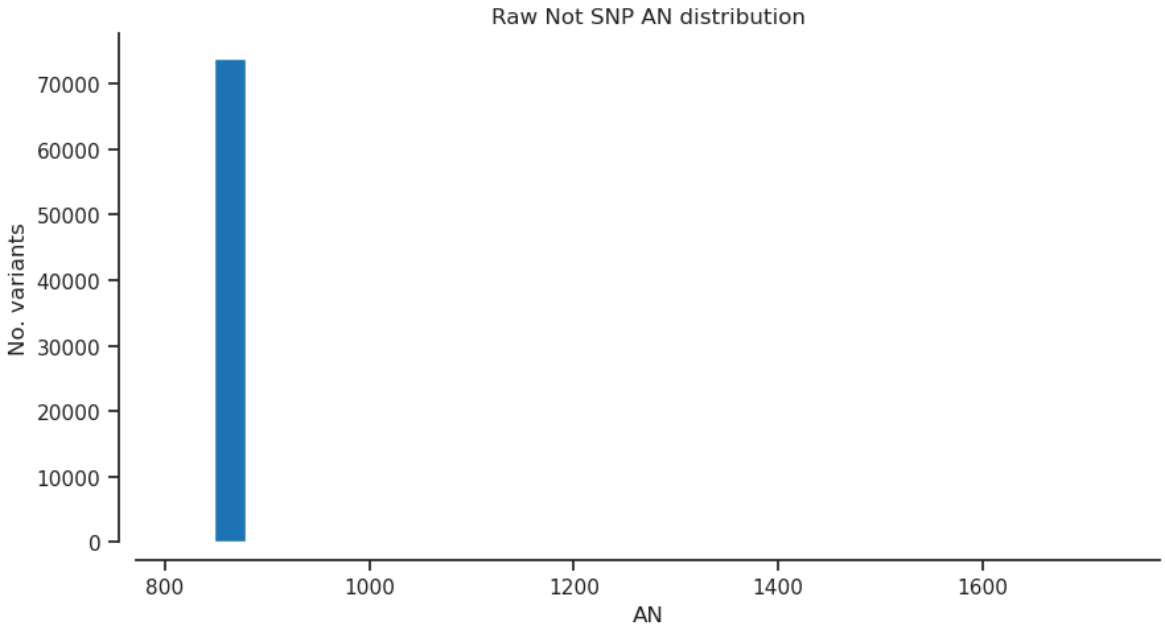
In [44]:
```
plot_hist('AN','var') # Total number of alleles in called genotypes
```



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 [45]:
```
# 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[45]:
```
125698
```

Genotype

In [46]:
```
calldata_var = callset_var['calldata']
list(calldata_var)
```

Out[46]:
```
['AD', 'DP', 'GQ', 'GT', 'MIN_DP', 'PGT', 'PID', 'PL', 'PS', 'RGQ', 'S
B']
```

In [47]:
```
genotypes_var = allel.GenotypeChunkedArray(calldata_var['GT'])
genotypes_var
```

Out[47]: <GenotypeChunkedArray shape=(216087, 432, 2) dtype=int8 chunks=(65536, 64, 2) nbytes=178.1M cbytes=7.5M cratio=23.6 compression=gzip compression_opts=1 values=h5py._hl.dataset.Dataset>

	0	1	2	3	4	...	427	428	429	430	431
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
...						...					
216084	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
216085	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
216086	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0

In [48]:
```
# using the selected filters set above
gt_filtered_snps = genotypes_var.subset(variant_selection)
gt_filtered_snps
```

Out[48]: <GenotypeChunkedArray shape=(125698, 432, 2) dtype=int8 chunks=(983, 432, 2) nbytes=103.6M cbytes=8.4M cratio=12.3 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3	4	...	427	428	429	430	431
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
...						...					
125695	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
125696	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
125697	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0

In [49]:
```
# grab the allele counts for the populations
ac = gt_filtered_snps.count_alleles()
ac
```

Out[49]: <AlleleCountsChunkedArray shape=(125698, 4) dtype=int32 chunks=(15713, 4) nbytes=1.9M cbytes=399.2K cratio=4.9 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3
0	862	2	0	0
1	850	14	0	0
2	858	6	0	0
...		...		
125695	863	1	0	0
125696	863	1	0	0
125697	862	2	0	0

In [50]:
```
ac[:]
```

Out[50]: <AlleleCountsArray shape=(125698, 4) dtype=int32>

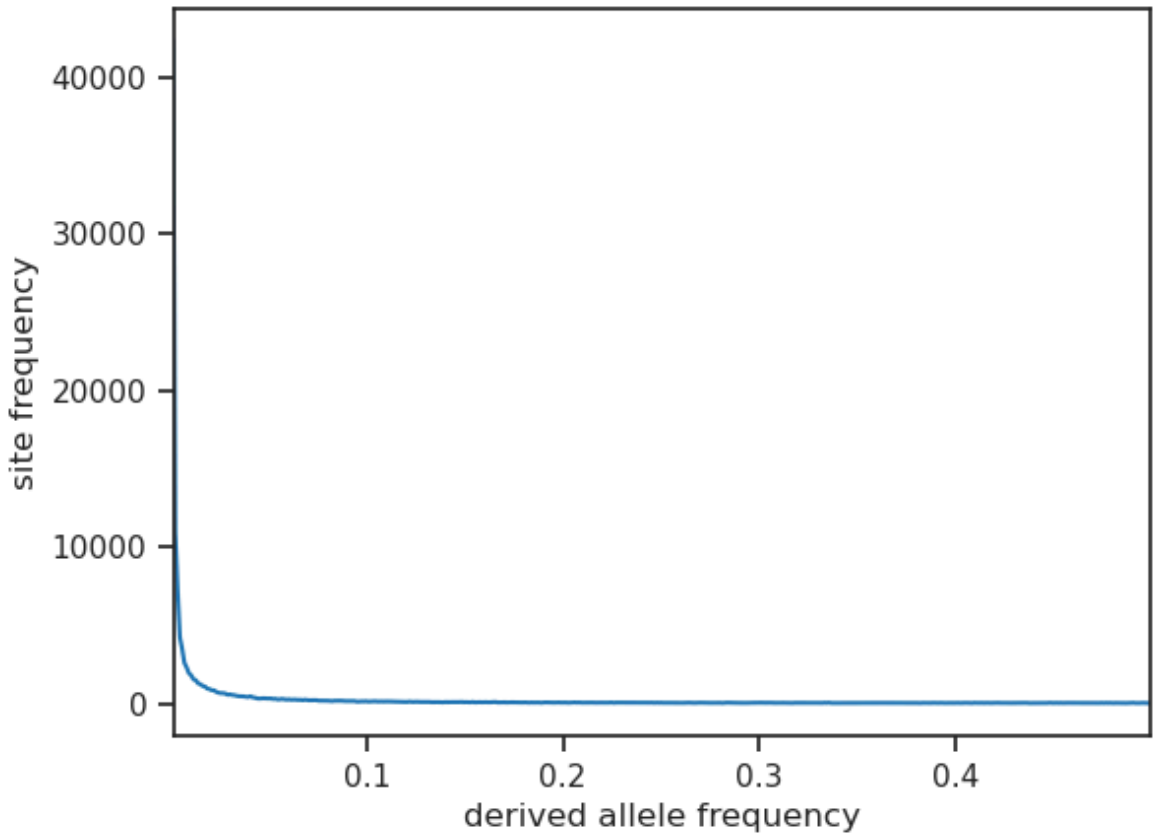
	0	1	2	3
0	862	2	0	0
1	850	14	0	0
2	858	6	0	0
...		...		
125695	863	1	0	0
125696	863	1	0	0
125697	862	2	0	0

In [51]:
```
# 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[51]:
```
array([[862,   2],
       [850,  14],
       [858,   6],
       ...,
       [863,   1],
       [863,   1],
       [862,   2]], dtype=int32)
```

In [52]:
```
# 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[52]:
```
<Axes: xlabel='derived allele frequency', ylabel='site frequency'>
```



In [53]:
```
biallelic = (ac.max_allele() == 1)
###This is the filter expression for biallelic sites
biallelic
```

Out[53]:
```
<ChunkedArrayWrapper shape=(125698,) dtype=bool chunks=(125698,)
   nbytes=122.8K cbytes=13.4K cratio=9.1
   compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shu
ffle': 1, 'blocksize': 0}
   values=zarr.core.Array>
```

In [54]:
```
# select only the biallelic variants
gt_biallelic = gt_filtered_snps.compress(biallelic)
gt_biallelic
```

Out[54]: <GenotypeChunkedArray shape=(122679, 432, 2) dtype=int8 chunks=(959, 432, 2) nbytes=101.1M cbytes=8.0M cratio=12.6 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3	4	...	427	428	429	430	431
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
...						...					
122676	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
122677	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
122678	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0

In [55]:
```
n_variants = len(gt_biallelic)
n_variants
```

Out[55]: 122679

In [56]:
```
pc_missing = gt_biallelic.count_missing(axis=0)[:] * 100 / n_variants
pc_het = gt_biallelic.count_het(axis=0)[:] * 100 / n_variants
```

Samples

In [57]:
```
samples_var = callset_var['samples']
samples_var = list(samples_var)
samples_var
```

Out[57]:
```
[b'1-ATAA51-001-AA01-A01',
 b'10-ATAA51-010-AA01-B02',
 b'100-ITAA57-025-AA02-D01',
 b'101-SIAA62-001-AA02-E01',
 b'102-SIAA62-002-AA02-F01',
 b'103-SIAA62-003-AA02-G01',
 b'104-SIAA62-004-AA02-H01',
 b'105-SIAA62-005-AA02-A02',
 b'106-SIAA62-006-AA02-B02',
 b'107-SIAA62-007-AA02-C02',
 b'108-SIAA62-008-AA02-D02',
 b'109-SIAA62-009-AA02-E02',
 b'11-ATAA51-011-AA01-C02',
 b'110-SIAA62-010-AA02-F02',
 b'111-SIAA62-011-AA02-G02',
 b'112-SIAA62-012-AA02-H02',
 b'113-SIAA62-013-AA02-A03',
 b'114-SIAA62-014-AA02-B03',
 b'115-SIAA62-015-AA02-C03',
 b'116-SIAA62-016-AA02-D03',
 b'117-SIAA62-017-AA02-E03',
 b'118-SIAA62-018-AA02-F03',
 b'119-SIAA62-019-AA02-G03',
 b'12-ATAA51-012-AA01-D02',
 b'120-SIAA62-020-AA02-H03',
 b'121-SIAA62-021-AA02-A04',
 b'122-SIAA62-022-AA02-B04',
 b'123-SIAA62-023-AA02-C04',
 b'124-SIAA62-024-AA02-D04',
 b'125-SIAA62-025-AA02-E04',
 b'126-SIAA63-001-AA02-F04',
 b'127-SIAA63-002-AA02-G04',
 b'128-SIAA63-003-AA02-H04',
 b'129-SIAA63-004-AA02-A05',
 b'13-ATAA51-013-AA01-E02',
 b'130-SIAA63-005-AA02-B05',
 b'131-SIAA63-006-AA02-C05',
 b'132-SIAA63-007-AA02-D05',
 b'133-SIAA63-008-AA02-E05',
 b'134-SIAA63-009-AA02-F05',
 b'135-SIAA63-010-AA02-G05',
 b'136-SIAA63-011-AA02-H05',
 b'137-SIAA63-012-AA02-A06',
 b'138-SIAA63-013-AA02-B06',
 b'139-SIAA63-014-AA02-C06',
 b'14-ATAA51-014-AA01-F02',
 b'140-SIAA63-015-AA02-D06',
 b'141-SIAA63-016-AA02-E06',
 b'142-SIAA63-017-AA02-F06',
 b'143-SIAA63-018-AA02-G06',
 b'144-SIAA63-019-AA02-H06',
 b'145-SIAA63-020-AA02-A07',
 b'146-SIAA63-021-AA02-B07',
 b'147-SIAA63-022-AA02-C07',
 b'148-SIAA63-023-AA02-D07',
 b'149-SIAA63-024-AA02-E07',
 b'15-ATAA51-015-AA01-G02',
 b'150-SIAA63-025-AA02-F07',
 b'151-ITAA64-001-AA02-G07',
 b'152-ITAA64-002-AA02-H07',
```

```
b'154-ITAA64-004-AA02-B08',
b'155-ITAA64-005-AA02-C08',
b'156-ITAA64-006-AA02-D08',
b'157-ITAA64-007-AA02-E08',
b'158-ITAA64-008-AA02-F08',
b'159-ITAA64-009-AA02-G08',
b'16-ATAA51-016-AA01-H02',
b'160-ITAA64-010-AA02-H08',
b'161-ITAA64-011-AA02-A09',
b'162-ITAA64-012-AA02-B09',
b'163-ITAA64-013-AA02-C09',
b'164-ITAA64-014-AA02-D09',
b'165-ITAA64-015-AA02-E09',
b'166-ITAA64-016-AA02-F09',
b'167-ITAA64-017-AA02-G09',
b'168-ITAA64-018-AA02-H09',
b'169-ITAA64-019-AA02-A10',
b'17-ATAA51-017-AA01-A03',
b'170-ITAA64-020-AA02-B10',
b'171-ITAA64-021-AA02-C10',
b'172-ITAA64-022-AA02-D10',
b'173-ITAA64-023-AA02-E10',
b'174-ITAA64-024-AA02-F10',
b'175-ITAA64-025-AA02-G10',
b'176-ROAA58-001-AA02-H10',
b'177-ROAA58-002-AA02-A11',
b'178-ROAA58-003-AA02-B11',
b'179-ROAA58-004-AA02-C11',
b'18-ATAA51-018-AA01-B03',
b'180-ROAA58-005-AA02-D11',
b'181-ROAA58-006-AA02-E11',
b'182-ROAA58-007-AA02-F11',
b'183-ROAA58-008-AA02-G11',
b'184-ROAA58-009-AA02-H11',
b'185-ROAA58-010-AA02-A12',
b'186-ROAA58-011-AA02-B12',
b'187-ROAA58-012-AA02-C12',
b'188-ROAA58-013-AA02-D12',
b'189-ROAA58-014-AA02-E12',
b'19-ATAA51-019-AA01-C03',
b'190-ROAA58-015-AA02-F12',
b'191-ROAA58-016-AA02-G12',
b'192-ROAA58-017-AA02-H12',
b'193-ROAA58-018-AA03-A01',
b'194-ROAA58-019-AA03-B01',
b'195-ROAA58-020-AA03-C01',
b'196-ROAA58-021-AA03-D01',
b'197-ROAA58-022-AA03-E01',
b'198-ROAA58-023-AA03-F01',
b'199-ROAA58-024-AA03-G01',
b'2-ATAA51-002-AA01-B01',
b'20-ATAA51-020-AA01-D03',
b'200-ROAA58-025-AA03-H01',
b'201-ROAA59-001-AA03-A02',
b'202-ROAA59-002-AA03-B02',
b'203-ROAA59-003-AA03-C02',
b'204-ROAA59-004-AA03-D02',
b'205-ROAA59-005-AA03-E02',
b'206-ROAA59-006-AA03-F02',
b'207-ROAA59-007-AA03-G02',
```

```
b'208-ROAA59-008-AA03-H02',
b'209-ROAA59-009-AA03-A03',
b'21-ATAA51-021-AA01-E03',
b'210-ROAA59-010-AA03-B03',
b'211-ROAA59-011-AA03-C03',
b'212-ROAA59-012-AA03-D03',
b'213-ROAA59-013-AA03-E03',
b'214-ROAA59-014-AA03-F03',
b'215-ROAA59-015-AA03-G03',
b'216-ROAA59-016-AA03-H03',
b'218-ROAA59-018-AA03-B04',
b'219-ROAA59-019-AA03-C04',
b'22-ATAA51-022-AA01-F03',
b'220-ROAA59-020-AA03-D04',
b'221-ROAA59-021-AA03-E04',
b'222-ROAA59-022-AA03-F04',
b'223-ROAA59-023-AA03-G04',
b'224-ROAA59-024-AA03-H04',
b'225-ROAA59-025-AA03-A05',
b'226-ROAA60-001-AA03-B05',
b'227-ROAA60-002-AA03-C05',
b'228-ROAA60-003-AA03-D05',
b'229-ROAA60-004-AA03-E05',
b'23-ATAA51-023-AA01-G03',
b'230-ROAA60-005-AA03-F05',
b'231-ROAA60-006-AA03-G05',
b'232-ROAA60-007-AA03-H05',
b'233-ROAA60-008-AA03-A06',
b'234-ROAA60-009-AA03-B06',
b'235-ROAA60-010-AA03-C06',
b'236-ROAA60-011-AA03-D06',
b'237-ROAA60-012-AA03-E06',
b'238-ROAA60-013-AA03-F06',
b'239-ROAA60-014-AA03-G06',
b'24-ATAA51-024-AA01-H03',
b'240-ROAA60-015-AA03-H06',
b'241-ROAA60-016-AA03-A07',
b'242-ROAA60-017-AA03-B07',
b'243-ROAA60-018-AA03-C07',
b'244-ROAA60-019-AA03-D07',
b'245-ROAA60-020-AA03-E07',
b'246-ROAA60-021-AA03-F07',
b'247-ROAA60-022-AA03-G07',
b'248-ROAA60-023-AA03-H07',
b'249-ROAA60-024-AA03-A08',
b'25-ATAA51-025-AA01-A04',
b'250-ROAA60-025-AA03-B08',
b'251-ROAA61-001-AA03-C08',
b'252-ROAA61-002-AA03-D08',
b'253-ROAA61-003-AA03-E08',
b'254-ROAA61-004-AA03-F08',
b'255-ROAA61-005-AA03-G08',
b'256-ROAA61-006-AA03-H08',
b'257-ROAA61-007-AA03-A09',
b'258-ROAA61-008-AA03-B09',
b'259-ROAA61-009-AA03-C09',
b'26-ATAA52-001-AA01-B04',
b'260-ROAA61-010-AA03-D09',
b'261-ROAA61-011-AA03-E09',
b'262-ROAA61-012-AA03-F09',
```

```
b'263-ROAA61-013-AA03-G09',
b'264-ROAA61-014-AA03-H09',
b'265-ROAA61-015-AA03-A10',
b'266-ROAA61-016-AA03-B10',
b'267-ROAA61-017-AA03-C10',
b'268-ROAA61-018-AA03-D10',
b'269-ROAA61-019-AA03-E10',
b'27-ATAA52-002-AA01-C04',
b'270-ROAA61-020-AA03-F10',
b'271-ROAA61-021-AA03-G10',
b'272-ROAA61-022-AA03-H10',
b'273-ROAA61-023-AA03-A11',
b'274-ROAA61-024-AA03-B11',
b'275-ROAA61-025-AA03-C11',
b'276-DEAA55-001-AA03-D11',
b'277-DEAA55-002-AA03-E11',
b'278-DEAA55-003-AA03-F11',
b'279-DEAA55-004-AA03-G11',
b'28-ATAA52-003-AA01-D04',
b'280-DEAA55-005-AA03-H11',
b'281-DEAA55-006-AA03-A12',
b'282-DEAA55-007-AA03-B12',
b'283-DEAA55-008-AA03-C12',
b'284-DEAA55-009-AA03-D12',
b'285-DEAA55-010-AA03-E12',
b'286-DEAA55-011-AA03-F12',
b'287-DEAA55-012-AA03-G12',
b'288-DEAA55-013-AA03-H12',
b'289-DEAA55-014-AA04-A01',
b'29-ATAA52-004-AA01-E04',
b'290-DEAA55-015-AA04-B01',
b'291-DEAA55-016-AA04-C01',
b'292-DEAA55-017-AA04-D01',
b'293-DEAA55-018-AA04-E01',
b'294-DEAA55-019-AA04-F01',
b'295-DEAA55-020-AA04-G01',
b'296-DEAA55-021-AA04-H01',
b'297-DEAA55-022-AA04-A02',
b'298-DEAA55-023-AA04-B02',
b'299-DEAA55-024-AA04-C02',
b'3-ATAA51-003-AA01-C01',
b'30-ATAA52-005-AA01-F04',
b'300-DEAA55-025-AA04-D02',
b'301-FRAA54-001-AA04-E02',
b'302-FRAA54-002-AA04-F02',
b'303-FRAA54-003-AA04-G02',
b'304-FRAA54-004-AA04-H02',
b'305-FRAA54-005-AA04-A03',
b'306-FRAA54-006-AA04-B03',
b'307-FRAA54-007-AA04-C03',
b'308-FRAA54-008-AA04-D03',
b'309-FRAA54-009-AA04-E03',
b'31-ATAA52-006-AA01-G04',
b'310-FRAA54-010-AA04-F03',
b'311-FRAA54-011-AA04-G03',
b'312-FRAA54-012-AA04-H03',
b'313-FRAA54-013-AA04-A04',
b'314-FRAA54-014-AA04-B04',
b'315-FRAA54-015-AA04-C04',
b'316-FRAA54-016-AA04-D04',
```

```
b'317-FRAA54-017-AA04-E04',
b'318-FRAA54-018-AA04-F04',
b'319-FRAA54-019-AA04-G04',
b'32-ATAA52-007-AA01-H04',
b'320-FRAA54-020-AA04-H04',
b'321-FRAA54-021-AA04-A05',
b'322-FRAA54-022-AA04-B05',
b'323-FRAA54-023-AA04-C05',
b'324-FRAA54-024-AA04-D05',
b'325-FRAA54-025-AA04-E05',
b'326-ITAA65-001-AA04-F05',
b'327-ITAA65-002-AA04-G05',
b'328-ITAA65-003-AA04-H05',
b'329-ITAA65-004-AA04-A06',
b'33-ATAA52-008-AA01-A05',
b'330-ITAA65-005-AA04-B06',
b'331-ITAA65-006-AA04-C06',
b'332-ITAA65-007-AA04-D06',
b'333-ITAA65-008-AA04-E06',
b'334-ITAA65-009-AA04-F06',
b'335-ITAA65-010-AA04-G06',
b'336-ITAA65-011-AA04-H06',
b'337-ITAA65-012-AA04-A07',
b'338-ITAA65-013-AA04-B07',
b'339-ITAA65-014-AA04-C07',
b'34-ATAA52-009-AA01-B05',
b'340-ITAA65-015-AA04-D07',
b'341-ITAA65-016-AA04-E07',
b'342-ITAA65-017-AA04-F07',
b'343-ITAA65-018-AA04-G07',
b'344-ITAA65-019-AA04-H07',
b'345-ITAA65-020-AA04-A08',
b'346-ITAA65-021-AA04-B08',
b'347-ITAA65-022-AA04-C08',
b'348-ITAA65-023-AA04-D08',
b'349-ITAA65-024-AA04-E08',
b'35-ATAA52-010-AA01-C05',
b'350-ITAA65-025-AA04-F08',
b'36-ATAA52-011-AA01-D05',
b'37-ATAA52-012-AA01-E05',
b'38-ATAA52-013-AA01-F05',
b'381-ITAA64-031-AA04-E12',
b'382-ROAA59-032-AA04-F12',
b'385-FRAA03-001-AA05-A01',
b'386-FRAA03-002-AA05-B01',
b'387-FRAA03-004-AA05-C01',
b'388-FRAA03-005-AA05-D01',
b'389-FRAA03-006-AA05-E01',
b'39-ATAA52-014-AA01-G05',
b'390-FRAA03-009-AA05-F01',
b'391-FRAA03-010-AA05-G01',
b'392-FRAA03-011-AA05-H01',
b'393-FRAA03-012-AA05-A02',
b'394-FRAA03-013-AA05-B02',
b'395-FRAA03-014-AA05-C02',
b'396-FRAA03-016-AA05-D02',
b'397-FRAA03-017-AA05-E02',
b'398-FRAA03-019-AA05-F02',
b'399-FRAA03-020-AA05-G02',
b'4-ATAA51-004-AA01-D01',
```

```
b'40-ATAA52-015-AA01-H05',
b'400-FRAA03-021-AA05-H02',
b'401-FRAA03-022-AA05-A03',
b'402-FRAA03-023-AA05-B03',
b'403-FRAA03-025-AA05-C03',
b'404-FRAA05-001-AA05-D03',
b'405-FRAA05-002-AA05-E03',
b'406-FRAA05-003-AA05-F03',
b'407-FRAA05-004-AA05-G03',
b'408-FRAA05-006-AA05-H03',
b'409-FRAA05-007-AA05-A04',
b'41-ATAA52-016-AA01-A06',
b'410-FRAA05-008-AA05-B04',
b'411-FRAA05-010-AA05-C04',
b'412-FRAA05-011-AA05-D04',
b'413-FRAA05-012-AA05-E04',
b'414-FRAA05-013-AA05-F04',
b'415-FRAA05-015-AA05-G04',
b'416-FRAA05-017-AA05-H04',
b'417-FRAA05-018-AA05-A05',
b'418-FRAA05-019-AA05-B05',
b'419-FRAA05-020-AA05-C05',
b'42-ATAA52-017-AA01-B06',
b'420-FRAA05-021-AA05-D05',
b'421-FRAA05-023-AA05-E05',
b'422-FRAA05-024-AA05-F05',
b'423-ESAA01-001-AA05-G05',
b'424-ESAA01-002-AA05-H05',
b'425-ESAA01-003-AA05-A06',
b'426-ESAA01-004-AA05-B06',
b'427-ESAA01-005-AA05-C06',
b'428-ESAA01-007-AA05-D06',
b'429-ESAA01-010-AA05-E06',
b'43-ATAA52-018-AA01-C06',
b'430-ESAA01-011-AA05-F06',
b'431-ESAA01-013-AA05-G06',
b'432-ESAA01-015-AA05-H06',
b'433-ESAA01-017-AA05-A07',
b'434-ESAA01-018-AA05-B07',
b'435-ESAA01-019-AA05-C07',
b'436-ESAA01-020-AA05-D07',
b'437-ESAA01-021-AA05-E07',
b'438-ESAA01-022-AA05-F07',
b'439-ESAA01-023-AA05-G07',
b'44-ATAA52-019-AA01-D06',
b'440-ESAA01-024-AA05-H07',
b'441-ESAA01-025-AA05-A08',
b'442-ITAA66-001-AA05-B08',
b'443-ITAA66-002-AA05-C08',
b'444-ITAA66-003-AA05-D08',
b'445-ITAA66-004-AA05-E08',
b'446-ITAA66-005-AA05-F08',
b'447-ITAA66-006-AA05-G08',
b'448-ITAA66-007-AA05-H08',
b'449-ITAA66-008-AA05-A09',
b'45-ATAA52-020-AA01-E06',
b'450-ITAA66-009-AA05-B09',
b'451-ITAA66-010-AA05-C09',
b'452-ITAA66-011-AA05-D09',
b'453-ITAA66-012-AA05-E09',
```

```
b'454-ITAA66-013-AA05-F09',
b'455-ITAA66-014-AA05-G09',
b'456-ITAA66-015-AA05-H09',
b'457-ITAA66-016-AA05-A10',
b'458-ITAA66-017-AA05-B10',
b'459-ITAA66-018-AA05-C10',
b'46-ATAA52-021-AA01-F06',
b'460-ITAA66-019-AA05-D10',
b'461-ITAA66-020-AA05-E10',
b'462-ITAA66-021-AA05-F10',
b'463-ITAA66-022-AA05-G10',
b'464-ITAA66-023-AA05-H10',
b'465-ITAA66-024-AA05-A11',
b'466-ITAA66-025-AA05-B11',
b'47-ATAA52-022-AA01-G06',
b'48-ATAA52-023-AA01-H06',
b'49-ATAA52-024-AA01-A07',
b'5-ATAA51-005-AA01-E01',
b'50-ATAA52-025-AA01-B07',
b'51-ITAA56-001-AA01-C07',
b'52-ITAA56-002-AA01-D07',
b'53-ITAA56-003-AA01-E07',
b'54-ITAA56-004-AA01-F07',
b'55-ITAA56-005-AA01-G07',
b'56-ITAA56-006-AA01-H07',
b'57-ITAA56-007-AA01-A08',
b'58-ITAA56-008-AA01-B08',
b'59-ITAA56-009-AA01-C08',
b'6-ATAA51-006-AA01-F01',
b'60-ITAA56-010-AA01-D08',
b'61-ITAA56-011-AA01-E08',
b'62-ITAA56-012-AA01-F08',
b'63-ITAA56-013-AA01-G08',
b'64-ITAA56-014-AA01-H08',
b'65-ITAA56-015-AA01-A09',
b'66-ITAA56-016-AA01-B09',
b'67-ITAA56-017-AA01-C09',
b'68-ITAA56-018-AA01-D09',
b'69-ITAA56-019-AA01-E09',
b'7-ATAA51-007-AA01-G01',
b'70-ITAA56-020-AA01-F09',
b'71-ITAA56-021-AA01-G09',
b'72-ITAA56-022-AA01-H09',
b'73-ITAA56-023-AA01-A10',
b'74-ITAA56-024-AA01-B10',
b'75-ITAA56-025-AA01-C10',
b'76-ITAA57-001-AA01-D10',
b'77-ITAA57-002-AA01-E10',
b'78-ITAA57-003-AA01-F10',
b'79-ITAA57-004-AA01-G10',
b'8-ATAA51-008-AA01-H01',
b'80-ITAA57-005-AA01-H10',
b'81-ITAA57-006-AA01-A11',
b'82-ITAA57-007-AA01-B11',
b'83-ITAA57-008-AA01-C11',
b'84-ITAA57-009-AA01-D11',
b'85-ITAA57-010-AA01-E11',
b'86-ITAA57-011-AA01-F11',
b'87-ITAA57-012-AA01-G11',
b'88-ITAA57-013-AA01-H11',
```

```
 b'89-ITAA57-014-AA01-A12',
 b'9-ATAA51-009-AA01-A02',
 b'90-ITAA57-015-AA01-B12',
 b'91-ITAA57-016-AA01-C12',
 b'92-ITAA57-017-AA01-D12',
 b'93-ITAA57-018-AA01-E12',
 b'94-ITAA57-019-AA01-F12',
 b'95-ITAA57-020-AA01-G12',
 b'96-ITAA57-021-AA01-H12',
 b'97-ITAA57-022-AA02-A01',
 b'98-ITAA57-023-AA02-B01',
 b'99-ITAA57-024-AA02-C01']
```

In [58]:
```
samples_fn = '~/scratch/data/Aalba/aalba_sample_list-scikit-allel.txt'
samples = pandas.read_csv(samples_fn, sep='\t')
samples
```

Out[58]:

	ID	Population
0	1-ATAA51-001-AA01-A01	AUT00179
1	10-ATAA51-010-AA01-B02	AUT00179
2	100-ITAA57-025-AA02-D01	ITA00271
3	101-SIAA62-001-AA02-E01	SVN00025
4	102-SIAA62-002-AA02-F01	SVN00025
...	...	...
427	95-ITAA57-020-AA01-G12	ITA00271
428	96-ITAA57-021-AA01-H12	ITA00271
429	97-ITAA57-022-AA02-A01	ITA00271
430	98-ITAA57-023-AA02-B01	ITA00271
431	99-ITAA57-024-AA02-C01	ITA00271

432 rows × 2 columns

In [59]:
```
samples.Population.value_counts()
```

Out[59]:
```
Population
AUT00179    25
ITA00271    25
SVN00025    25
SVN00023    25
ITA00260    25
ROU00358    25
ROU00104    25
ROU00389    25
ROU00477    25
AUT00215    25
DEU00114    25
FRA00006    25
ITA00069    25
ITA00029    25
ITA00217    25
FRA00019    19
ESP00339    19
FRA00004    19
Name: count, dtype: int64
```

In [60]:
```
populations = samples.Population.unique()
populations
###This identifiers come from the metadata file
```

Out[60]:
```
array(['AUT00179', 'ITA00271', 'SVN00025', 'SVN00023', 'ITA00260',
       'ROU00358', 'ROU00104', 'ROU00389', 'ROU00477', 'AUT00215',
       'DEU00114', 'FRA00006', 'ITA00069', 'FRA00019', 'FRA00004',
       'ESP00339', 'ITA00217', 'ITA00029'], dtype=object)
```

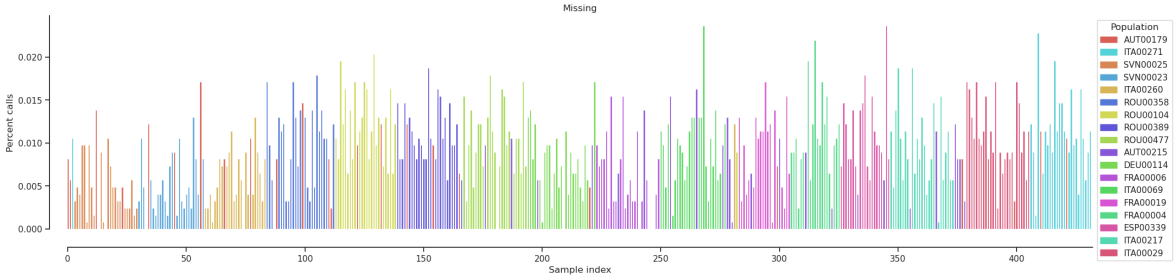
Gt frequency function

In [61]:
```
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", 18)
    pop2color = {'AUT00179': palette[0],
                 'ITA00271': palette[9],
                 'SVN00025': palette[1],
                 'SVN00023': palette[10],
                 'ITA00260': palette[2],
                 'ROU00358': palette[11],
                 'ROU00104': palette[3],
                 'ROU00389': palette[12],
                 'ROU00477': palette[4],
                 'AUT00215': palette[13],
                 'DEU00114': palette[5],
                 'FRA00006': palette[14],
                 'ITA00069': palette[6],
                 'FRA00019': palette[15],
                 'FRA00004': palette[7],
                 'ESP00339': palette[16],
                 'ITA00217': palette[8],
                 'ITA00029': palette[17]}
    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[9]),
               mpl.patches.Patch(color=palette[1]),
               mpl.patches.Patch(color=palette[10]),
               mpl.patches.Patch(color=palette[2]),
               mpl.patches.Patch(color=palette[11]),
               mpl.patches.Patch(color=palette[3]),
               mpl.patches.Patch(color=palette[12]),
               mpl.patches.Patch(color=palette[4]),
               mpl.patches.Patch(color=palette[13]),
               mpl.patches.Patch(color=palette[5]),
               mpl.patches.Patch(color=palette[14]),
               mpl.patches.Patch(color=palette[6]),
               mpl.patches.Patch(color=palette[15]),
               mpl.patches.Patch(color=palette[7]),
               mpl.patches.Patch(color=palette[16]),
               mpl.patches.Patch(color=palette[8]),
               mpl.patches.Patch(color=palette[17])]
    ax.legend(handles=handles, labels=['AUT00179', 'ITA00271', 'SVN00025'
       'ROU00358', 'ROU00104', 'ROU00389', 'ROU00477', 'AUT00215',
       'DEU00114', 'FRA00006', 'ITA00069', 'FRA00019', 'FRA00004',
       'ESP00339', 'ITA00217', 'ITA00029'], title='Population',
              bbox_to_anchor=(1, 1), loc='upper left')
```

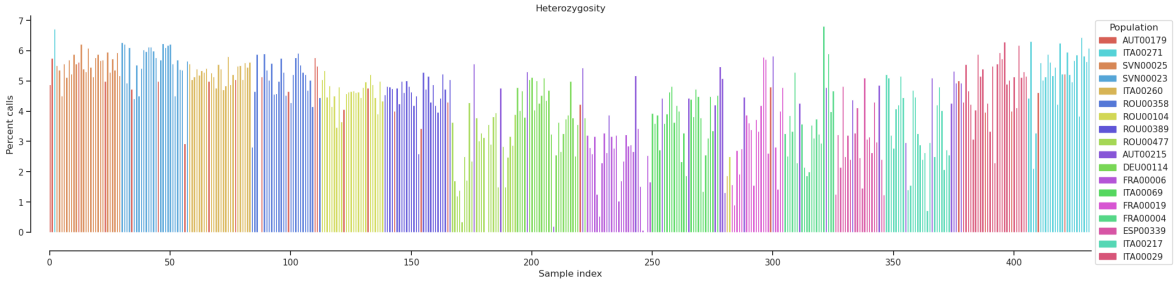
Plot missing

In [62]:
```
plot_genotype_frequency(pc_missing, 'Missing')
```



Plot heterozygosity

In [63]:
```
plot_genotype_frequency(pc_het, 'Heterozygosity')
```



PCA

In [64]:
```
palette = sns.color_palette("hls",18)
pop_colours = { 'AUT00179': palette[0],
                'ITA00271': palette[9],
                'SVN00025': palette[1],
                'SVN00023': palette[10],
                'ITA00260': palette[2],
                'ROU00358': palette[11],
                'ROU00104': palette[3],
                'ROU00389': palette[12],
                'ROU00477': palette[4],
                'AUT00215': palette[13],
                'DEU00114': palette[5],
                'FRA00006': palette[14],
                'ITA00069': palette[6],
                'FRA00019': palette[15],
                'FRA00004': palette[7],
                'ESP00339': palette[16],
                'ITA00217': palette[8],
                'ITA00029': palette[17]
}
```

In [65]:
```
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 [66]:
```
ac2 = gt_biallelic.count_alleles()
ac2
```

Out[66]: <AlleleCountsChunkedArray shape=(122679, 2) dtype=int32 chunks=(30670, 2) nbytes=958.4K cbytes=244.9K cratio=3.9 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1
0	862	2
1	850	14
2	858	6
...	...	
122676	863	1
122677	863	1
122678	862	2

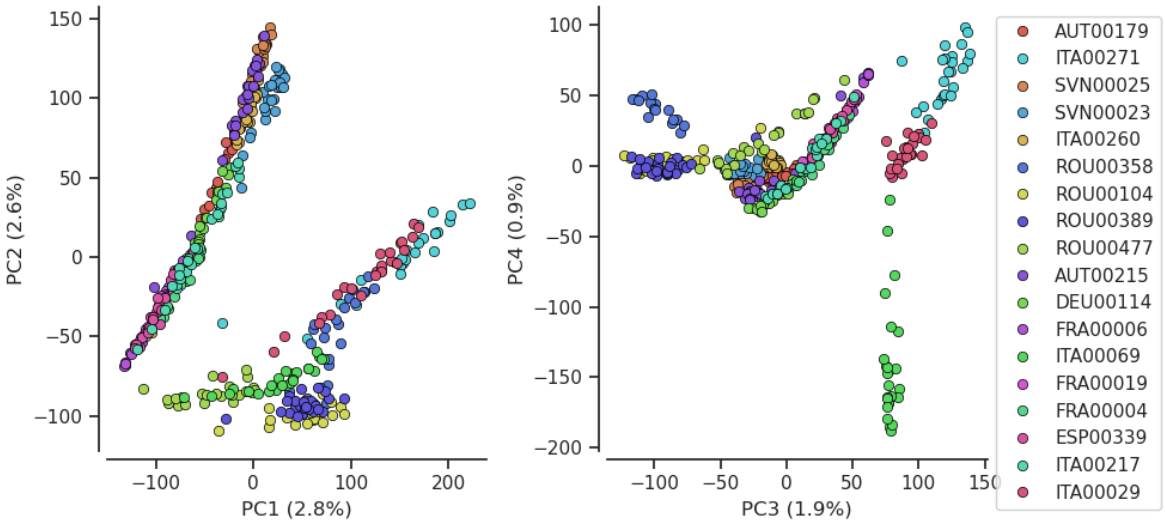
In [67]:
```
flt = (ac2[:, :2].min(axis=1) > 1)
gf = gt_biallelic.compress(flt, axis=0)
gn = gf.to_n_alt()
gn
```

Out[67]:
```
<ChunkedArrayWrapper shape=(80396, 432) dtype=int8 chunks=(2513, 432)
   nbytes=33.1M cbytes=5.3M cratio=6.3
   compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shu
ffle': 1, 'blocksize': 0}
   values=zarr.core.Array>
```

In [68]:
```
coords1, model1 = allel.pca(gn, n_components=10, scaler='patterson')
```

In [69]:
```
fig_pca(coords1, model1, 'Figure 1. Conventional PCA.')
```

Figure 1. Conventional PCA.



In [ ]: