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 [3]:
```
allel.vcf_to_hdf5('/users/mcevoysu/scratch/output/vcf_filtering/Pabies/ra
```

Get data

In [4]:
```
callset_var_fn = '/users/mcevoysu/scratch/output/scikit-allel/Pabies/raw_
callset_var = h5py.File(callset_var_fn, mode='r')
```

In [5]:
```
calldata_var = callset_var['calldata']
list(calldata_var)
```

Out[5]:
```
['AD', 'DP', 'GQ', 'GT', 'MIN_DP', 'PGT', 'PID', 'PL', 'PS', 'RGQ', 'S
B']
```

In [6]:
```
list(callset_var['variants'])
```

Out[6]:
```
['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 [7]:
```
variants = allel.VariantChunkedTable(callset_var['variants'])
variants
```

Out[7]: <VariantChunkedTable shape=(477988,) 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=81.6M cbytes=17.3M cratio=4.7 values=h5py._hl.group.Group>

	AC	AF	ALT	AN	BaseQRankSum	CHROM	DP	END	E
0	[2 -1 -1]	[0.002481 nan nan]	[b'C' b'' b'']	798	-1.068	b'MA_20601'	4893	-1	
1	[3 -1 -1]	[0.003722 nan nan]	[b'C' b'' b'']	798	-0.967	b'MA_20601'	4892	-1	
2	[2 -1 -1]	[0.003722 nan nan]	[b'G' b'' b'']	798	-0.621	b'MA_20601'	4895	-1	
...									
477985	[45 -1 -1]	[0.057 nan nan]	[b'T' b'' b'']	798	0.0	b'MA_3821795'	1743	-1	
477986	[16 -1 -1]	[0.02 nan nan]	[b'T' b'' b'']	798	0.0	b'MA_3821795'	1705	-1	
477987	[20 -1 -1]	[0.025 nan nan]	[b'T' b'' b'']	798	0.0	b'MA_3821795'	1704	-1	

In [8]:
```
variants_np = variants[:]
rawsnps = variants_np.query('(is_snp == True)')
rawsnps
```

Out[8]: <VariantTable shape=(327389,) 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	[2 -1 -1]	[0.002481 nan nan]	[b'C' b'' b'']	798	-1.068	b'MA_20601'	4893	-1	
1	[3 -1 -1]	[0.003722 nan nan]	[b'C' b'' b'']	798	-0.967	b'MA_20601'	4892	-1	
2	[2 -1 -1]	[0.003722 nan nan]	[b'G' b'' b'']	798	-0.621	b'MA_20601'	4895	-1	
...									
327386	[45 -1 -1]	[0.057 nan nan]	[b'T' b'' b'']	798	0.0	b'MA_3821795'	1743	-1	
327387	[16 -1 -1]	[0.02 nan nan]	[b'T' b'' b'']	798	0.0	b'MA_3821795'	1705	-1	
327388	[20 -1 -1]	[0.025 nan nan]	[b'T' b'' b'']	798	0.0	b'MA_3821795'	1704	-1	

In [9]:
```
notsnp = variants_np.query('(is_snp != True)')
notsnp
```

Out[9]: <VariantTable shape=(150599,) 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
0	[452 3 -1]	[0.566 0.003722 nan]	[b'A' b'*' b'']	798	0.043	b'MA_20601'	4583	-1
1	[443 3 9]	[0.555 0.003722 0.011]	[b'T' b'*' b'A']	798	0.224	b'MA_20601'	4565	-1
2	[2 -1 -1]	[0.002481 nan nan]	[b'*' b'' b'']	798	-0.921	b'MA_20601'	4196	-1
...								
150596	[51 307 9]	[0.063 0.386 0.011]	[b'A' b'C' b'*']	798	0.0	b'MA_3821795'	23493	-1
150597	[2 96 36]	[0.003722 0.12 0.045]	[b'*' b'T' b'G']	798	0.0	b'MA_3821795'	10924	-1
150598	[762 2 -1]	[0.954 0.003722 nan]	[b'G' b'*' b'']	798	0.0	b'MA_3821795'	11088	-1

Plot function

In [10]:
```
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 [11]:
```
numalt = rawsnps['numalt']
np.max(numalt)
```

Out[11]:
```
3
```

In [12]:
```
count_numalt = np.bincount(numalt)
count_numalt
```

Out[12]:
```
array([     0, 312376,  14547,    466])
```

In [13]:
```
n_multiallelic = np.sum(count_numalt[2:])
n_multiallelic
```

Out[13]:
```
15013
```

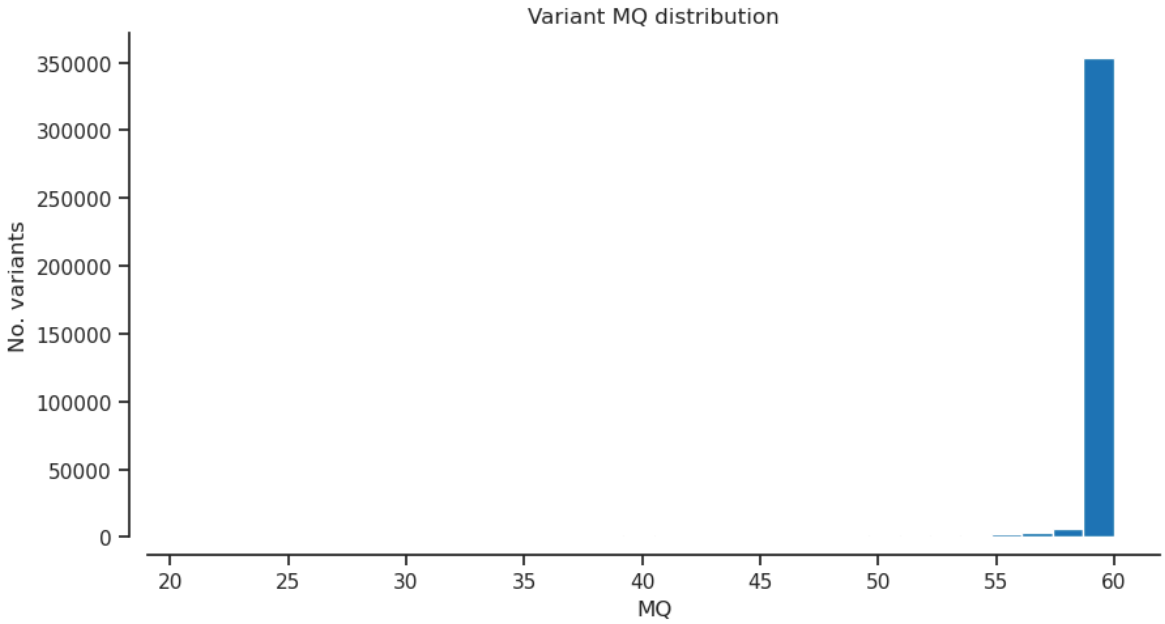
In [14]:
```
filter_expression = '(numalt == 1)'
biallelic_np = rawsnps.query(filter_expression)[:]
biallelic_np
```

Out[14]: <VariantTable shape=(312376,) 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')])>

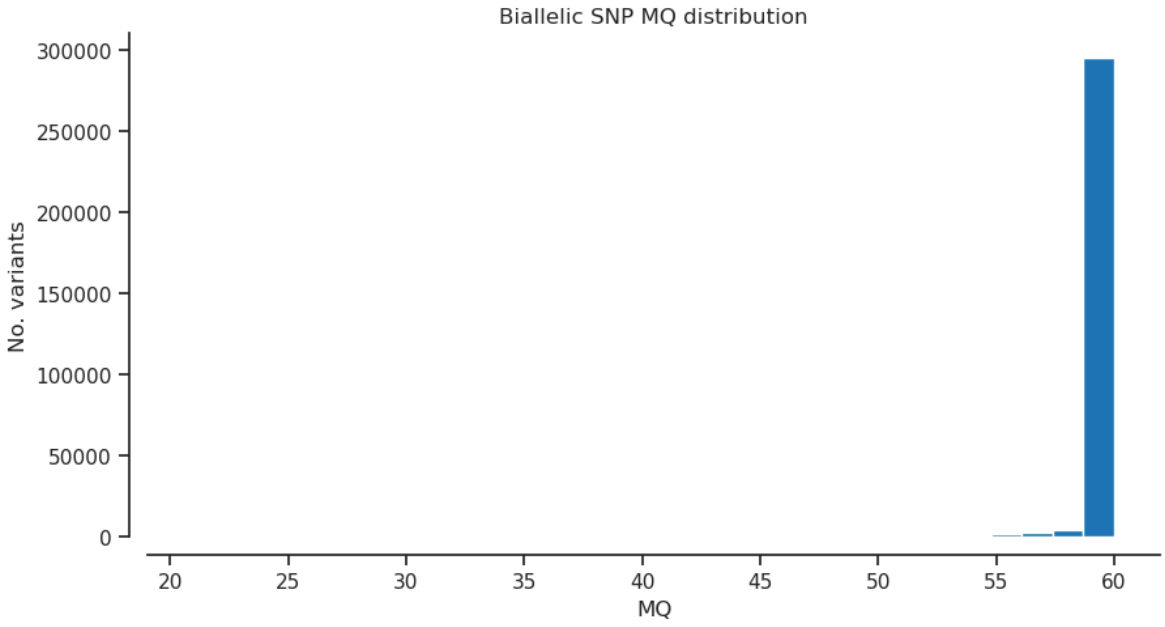
	AC	AF	ALT	AN	BaseQRankSum	CHROM	DP	END	E
0	[2 -1 -1]	[0.002481 nan nan]	[b'C' b'' b'']	798	-1.068	b'MA_20601'	4893	-1	
1	[3 -1 -1]	[0.003722 nan nan]	[b'C' b'' b'']	798	-0.967	b'MA_20601'	4892	-1	
2	[2 -1 -1]	[0.003722 nan nan]	[b'G' b'' b'']	798	-0.621	b'MA_20601'	4895	-1	
...									
312373	[45 -1 -1]	[0.057 nan nan]	[b'T' b'' b'']	798	0.0	b'MA_3821795'	1743	-1	
312374	[16 -1 -1]	[0.02 nan nan]	[b'T' b'' b'']	798	0.0	b'MA_3821795'	1705	-1	
312375	[20 -1 -1]	[0.025 nan nan]	[b'T' b'' b'']	798	0.0	b'MA_3821795'	1704	-1	

MQ - RMS mapping quality

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

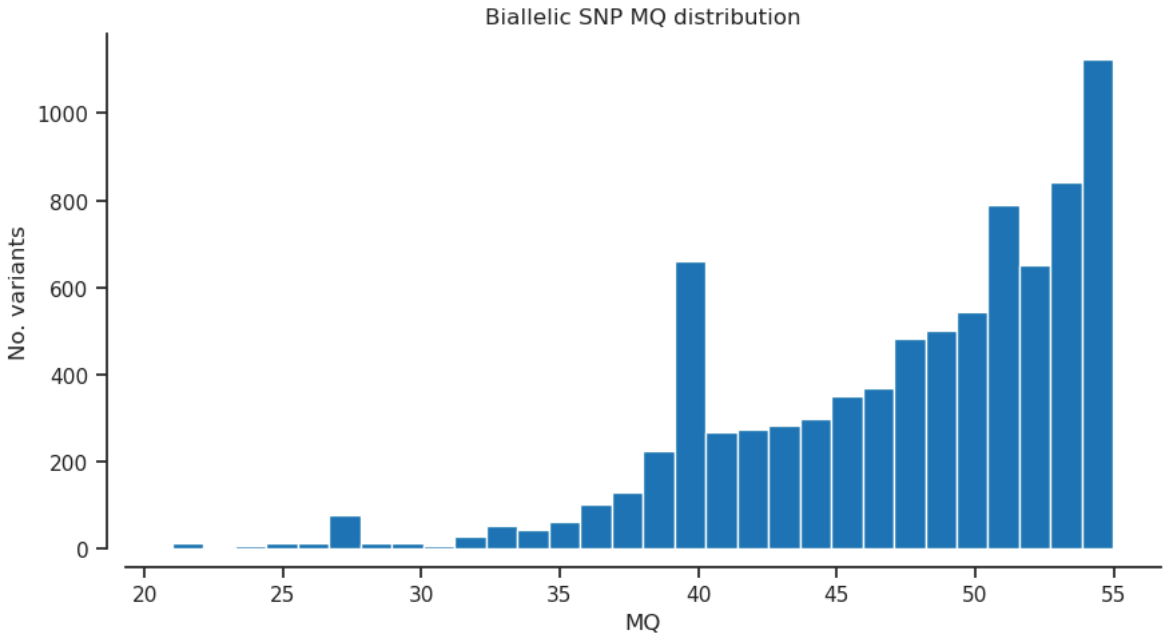


In [16]:
```
plot_hist('MQ','biallelic') # RMS mapping quality
```



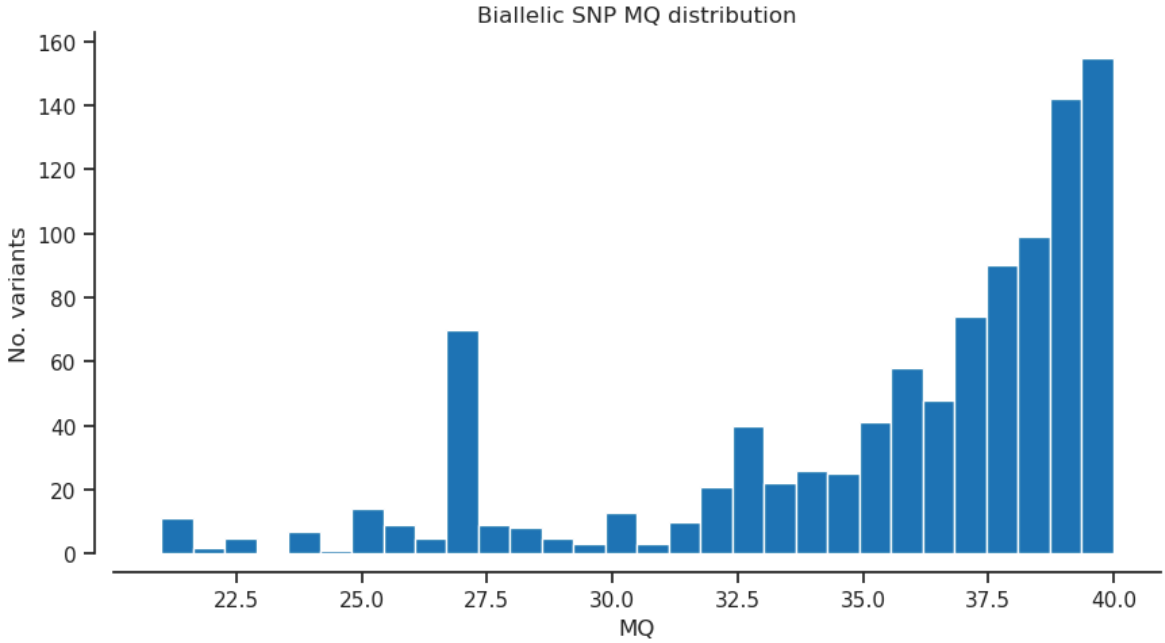
In [17]:
```
filter_expression = '(MQ < 55)'
bi_selection = biallelic_np.query(filter_expression)[:]
#np.count_nonzero(var_selection)
```

In [18]:
```
plot_hist('MQ')
```

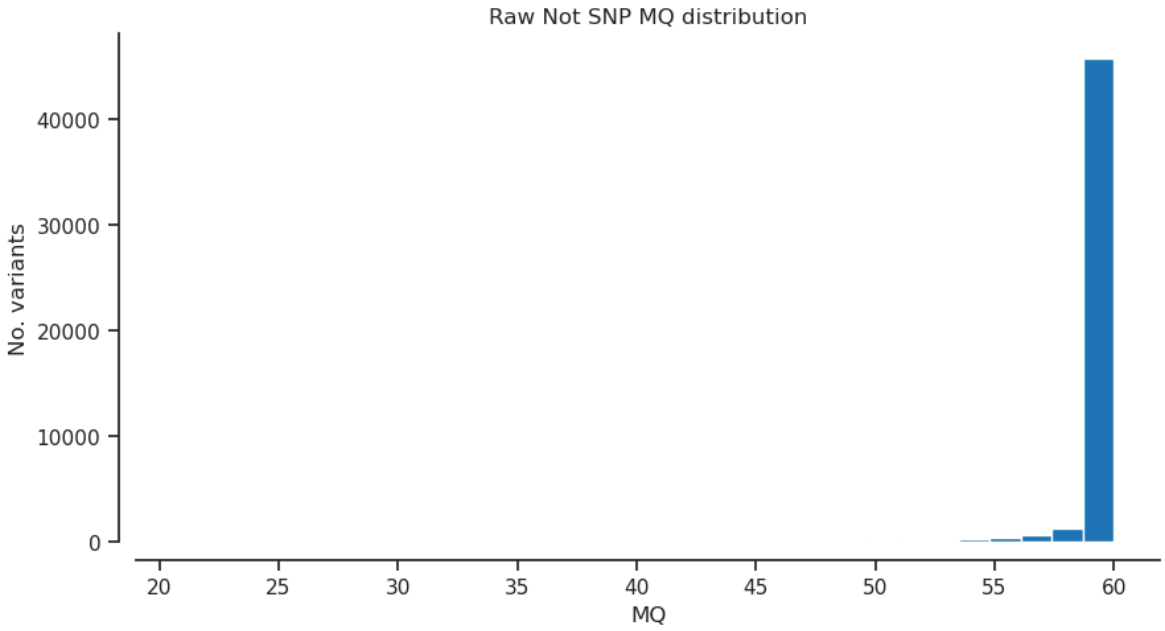


In [19]:
```
filter_expression = '(MQ < 40)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

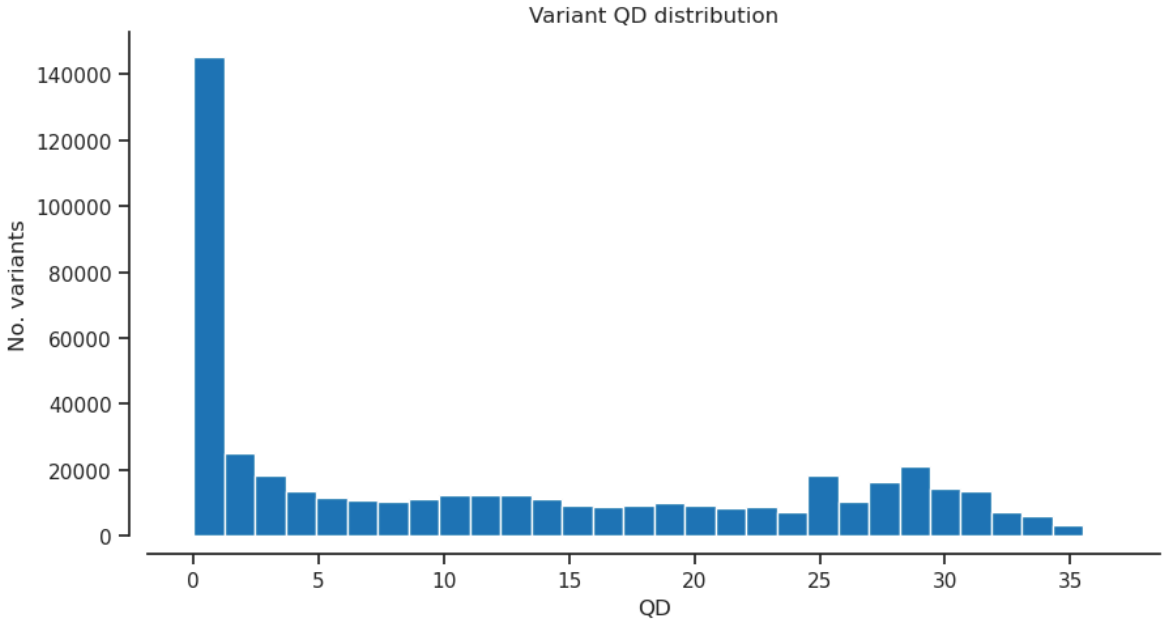


In [21]:
```
plot_hist('MQ','notsnp')
```

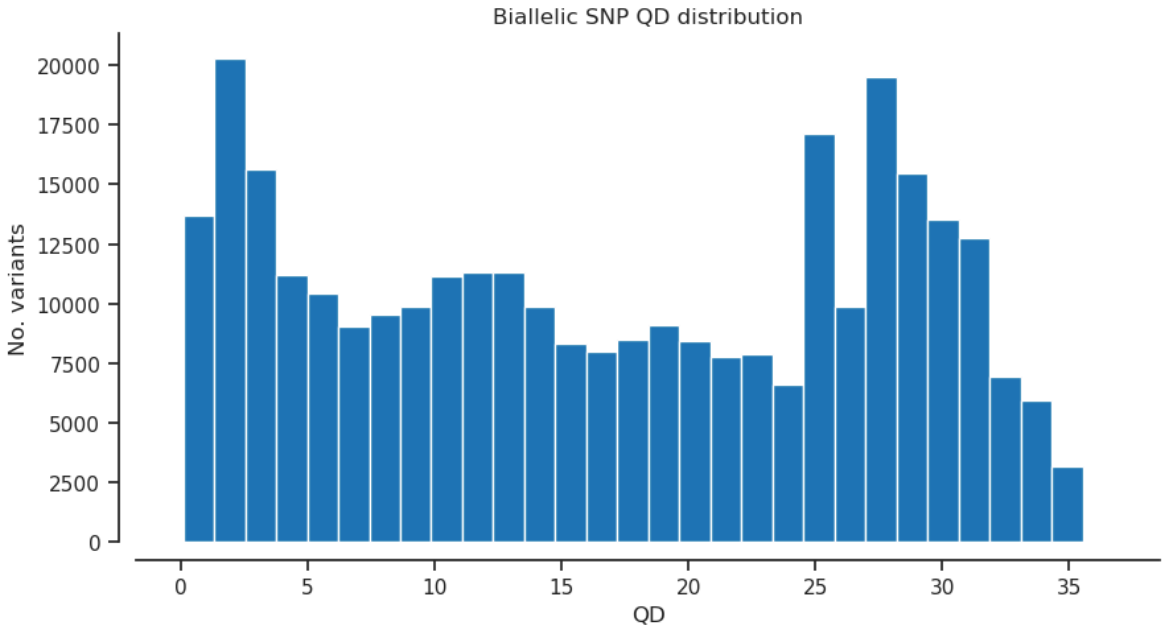


QD - Variant Confidence/Quality by Depth

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

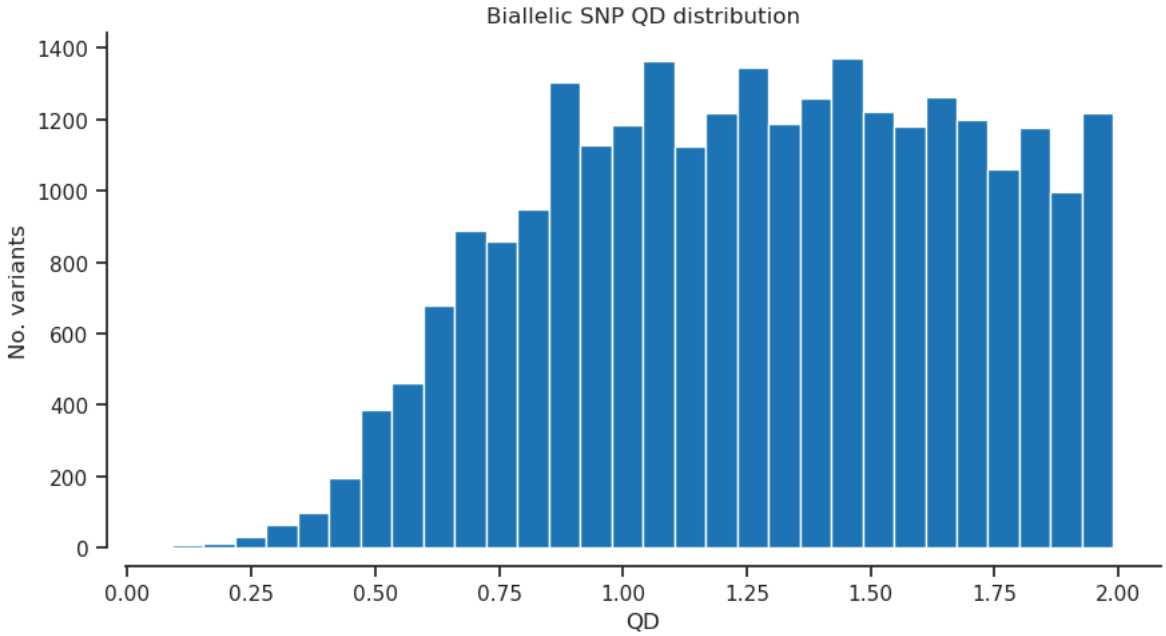


In [23]:
```
plot_hist('QD','biallelic') # Variant Confidence/Quality by Depth
```

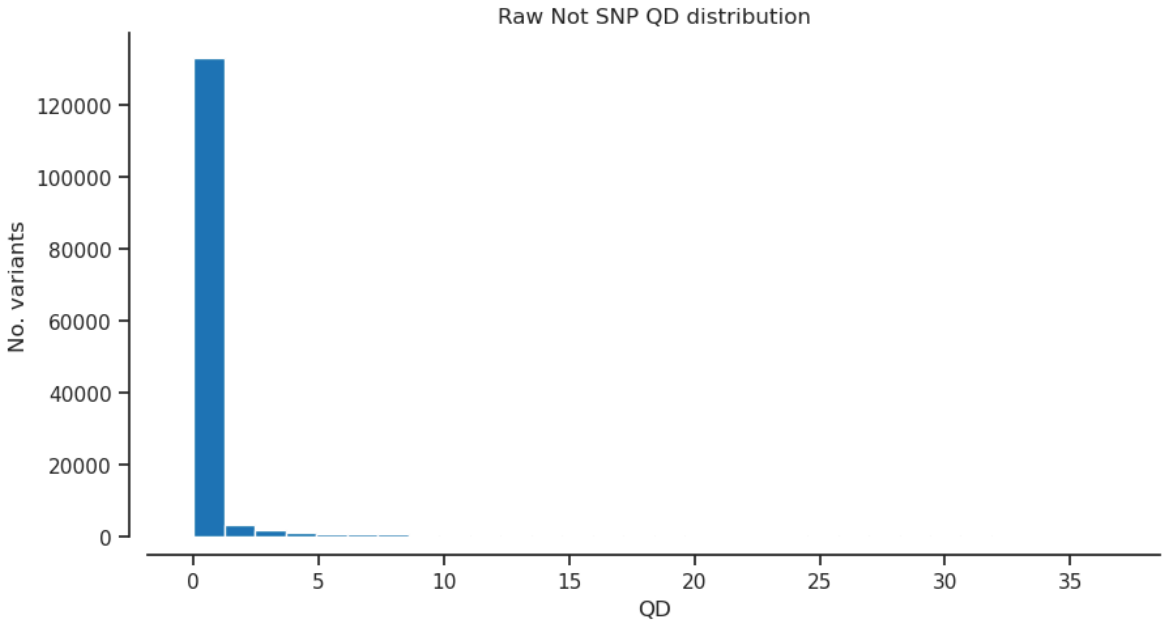


In [24]:
```
filter_expression = '(QD < 2)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

In [25]:
```
plot_hist('QD')
```

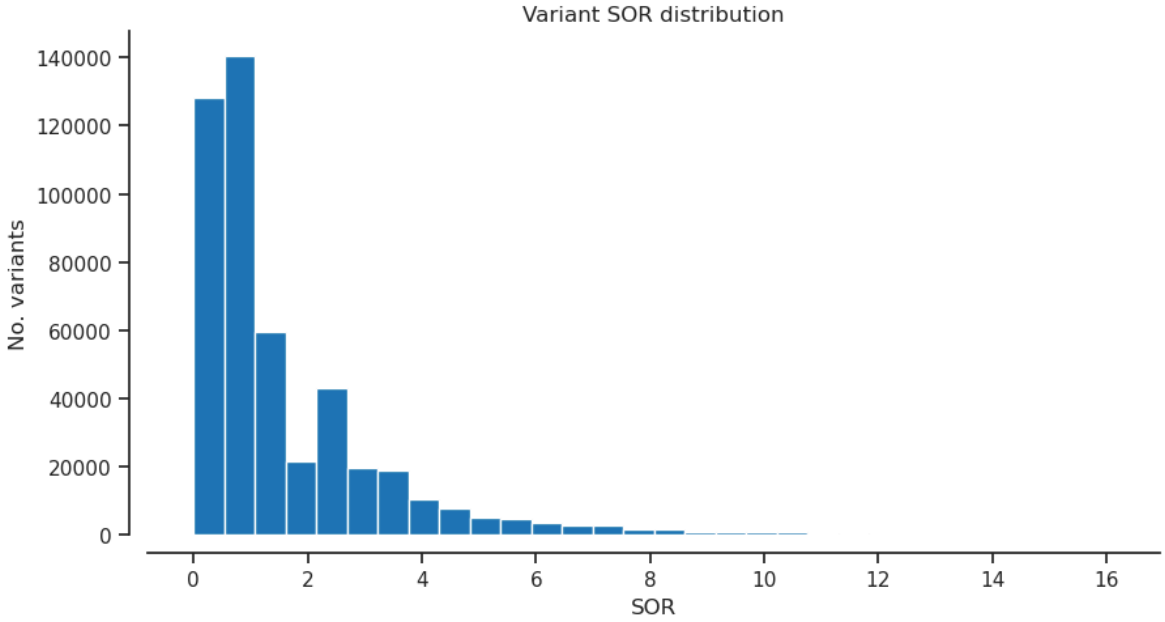


In [26]:
```
plot_hist('QD','notsnp') # Variant Confidence/Quality by Depth
```

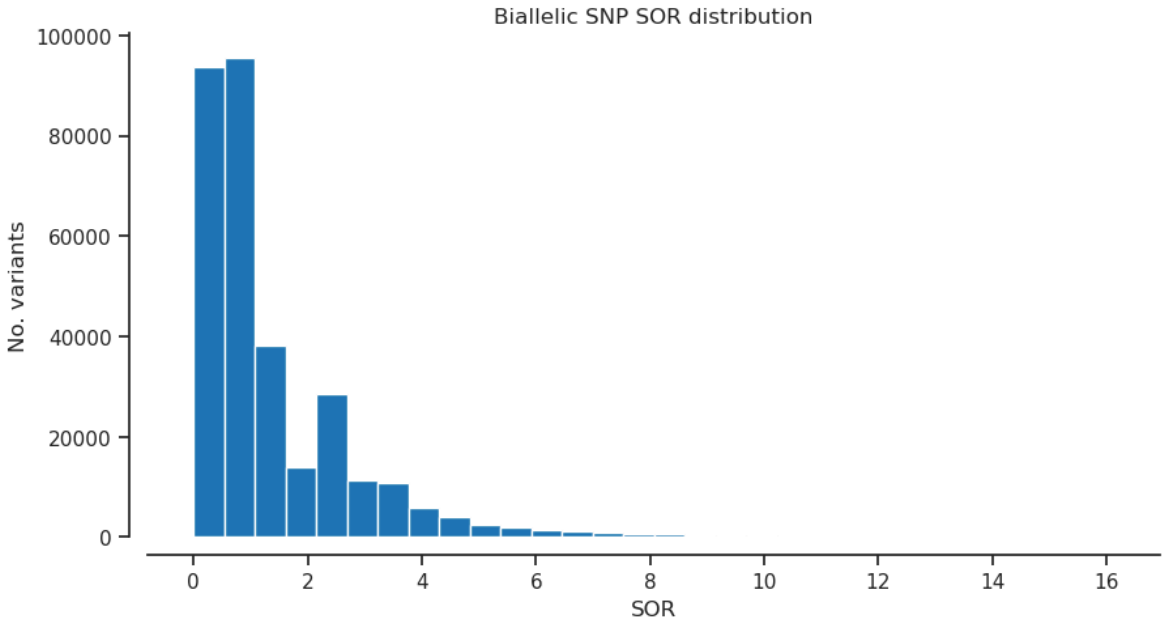


SOR - Symmetric Odds Ratio of 2x2 contingency table to detect strand bias

In [27]:
```
plot_hist('SOR', 'var') # Symmetric Odds Ratio of 2x2 contingency table t
```

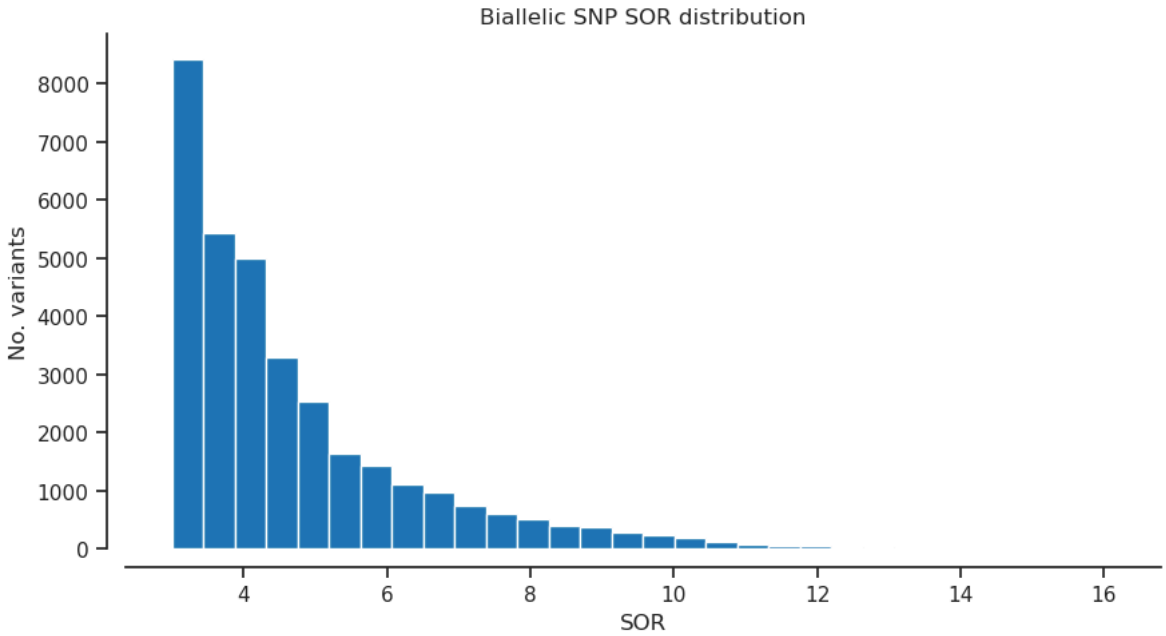


In [28]:
```
plot_hist('SOR','biallelic') # Symmetric Odds Ratio of 2x2 contingency ta
```

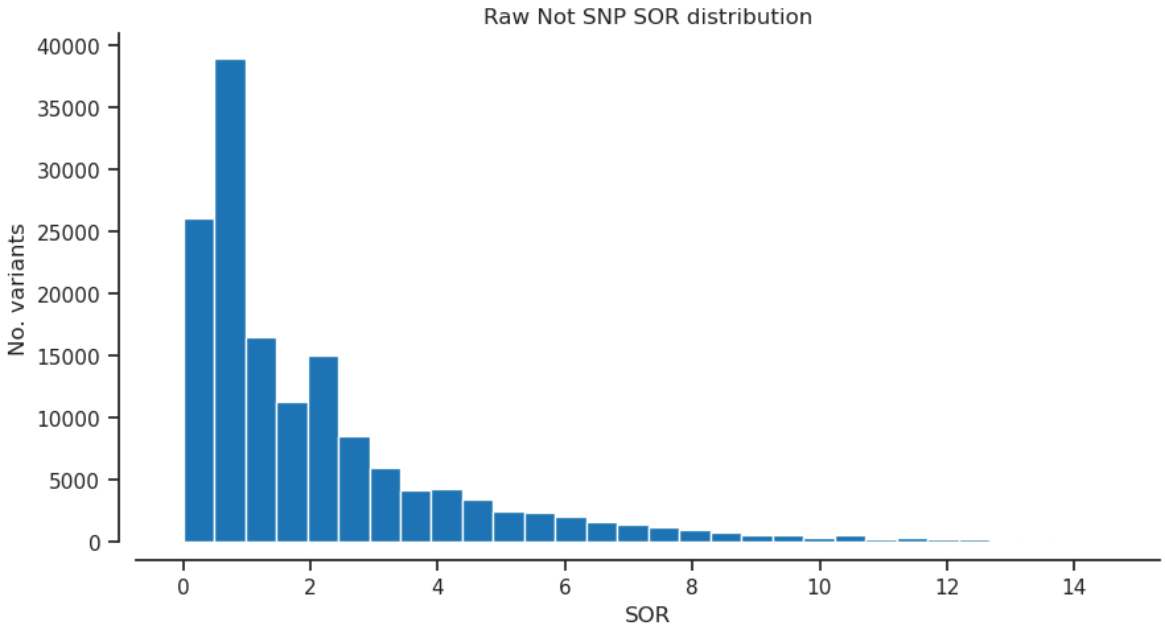


In [29]:
```
filter_expression = '(SOR > 3)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

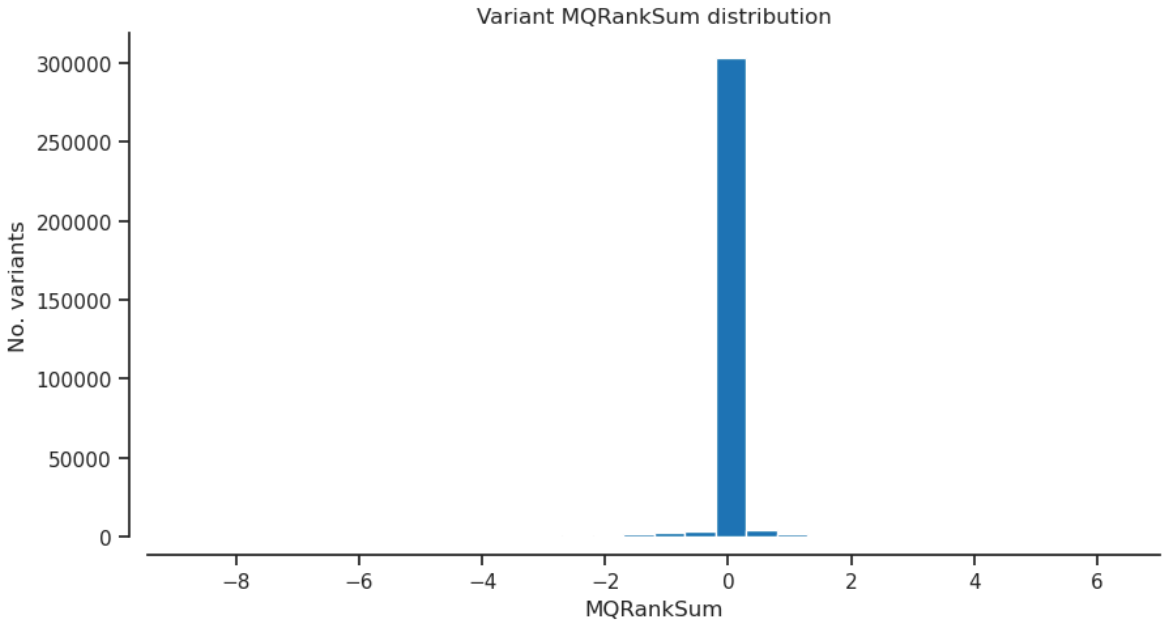


In [31]:
```
plot_hist('SOR','notsnp') # Symmetric Odds Ratio of 2x2 contingency table
```

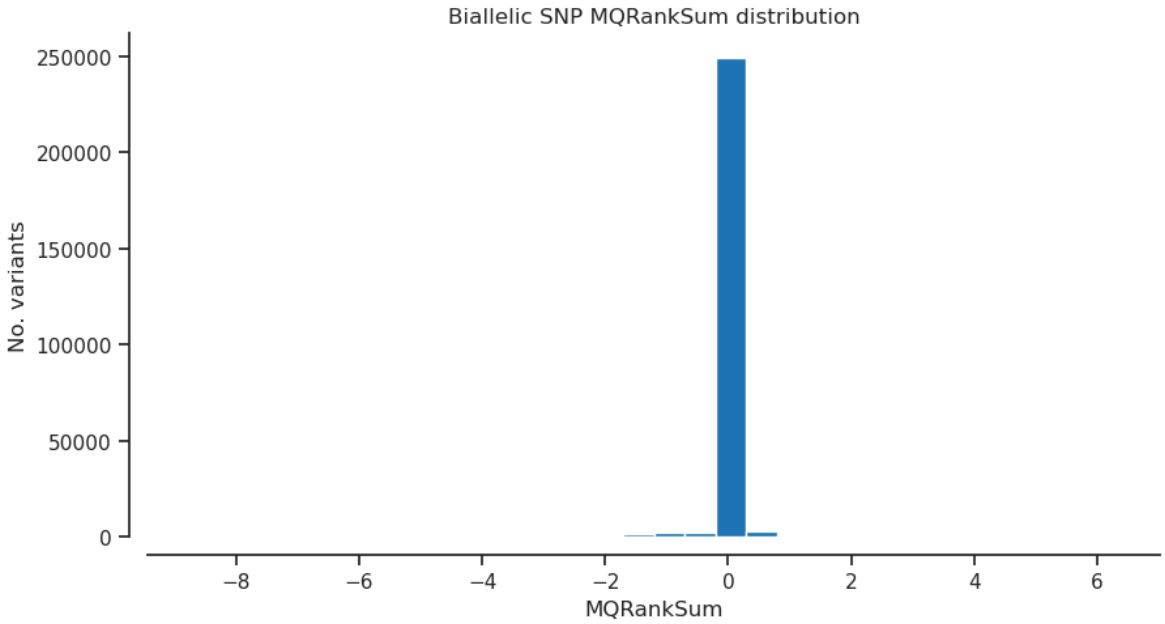


MQRankSum - Z-score From Wilcoxon rank sum test of Alt vs. Ref read mapping qualities

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

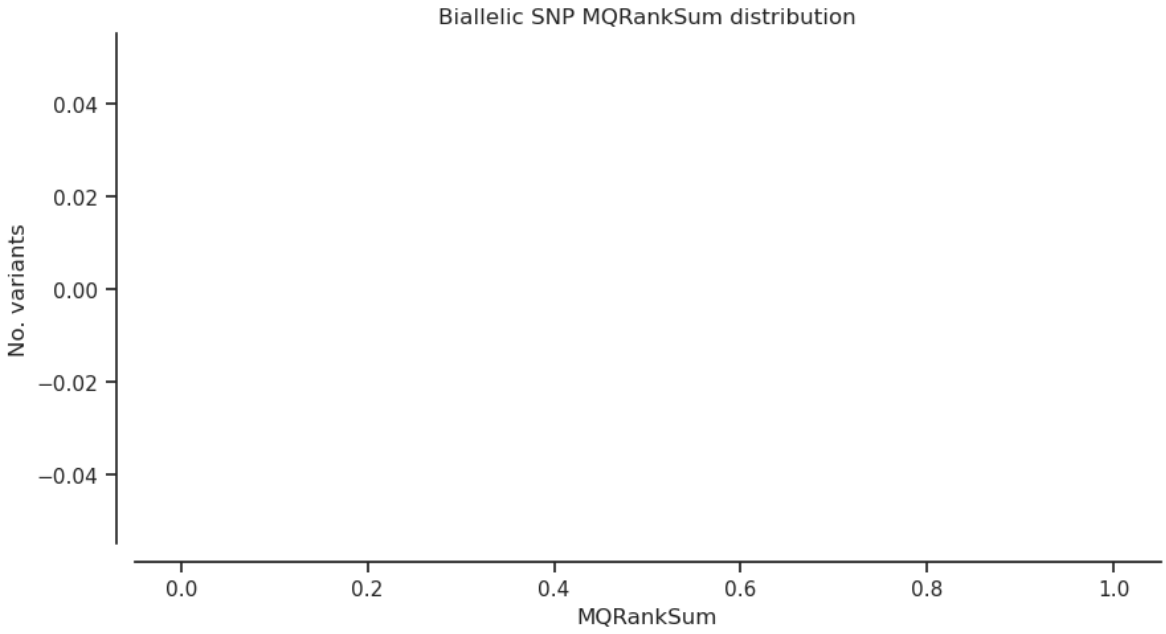


In [33]:
```
plot_hist('MQRankSum','biallelic') # Z-score From Wilcoxon rank sum test
```

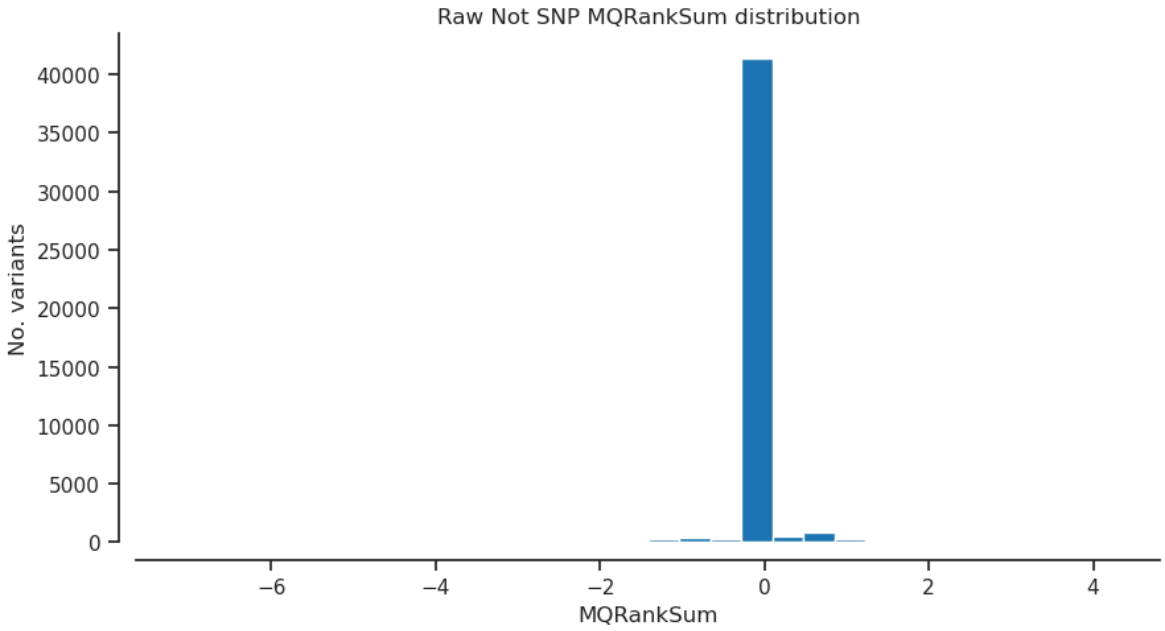


In [34]:
```
filter_expression = '(MQRankSum < -12.5)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

In [35]:
```
plot_hist('MQRankSum') # Z-score From Wilcoxon rank sum test of Alt vs. R
```

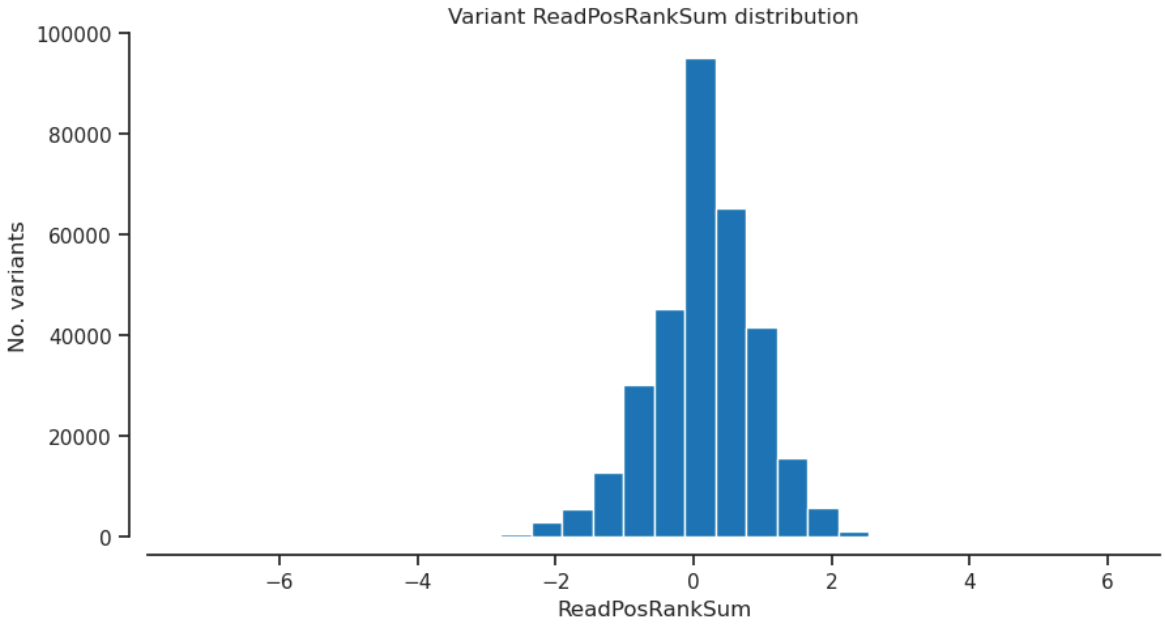


In [36]:
```
plot_hist('MQRankSum','notsnp') # Z-score From Wilcoxon rank sum test of
```

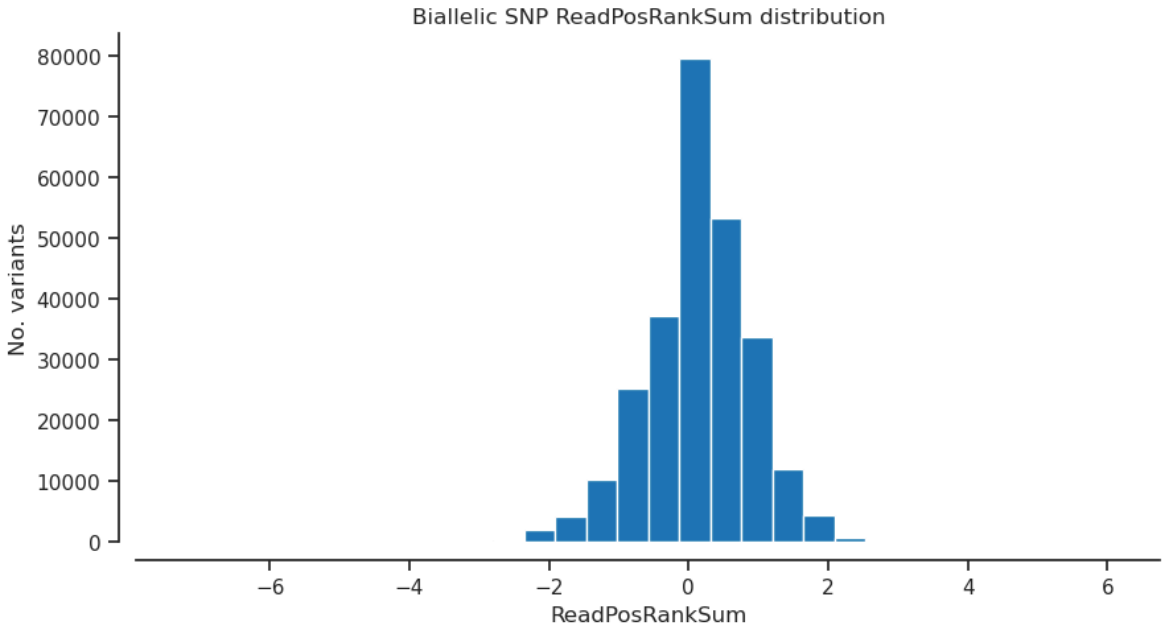


ReadPosRankSum - Z-score from Wilcoxon rank sum test of Alt vs. Ref read position bias

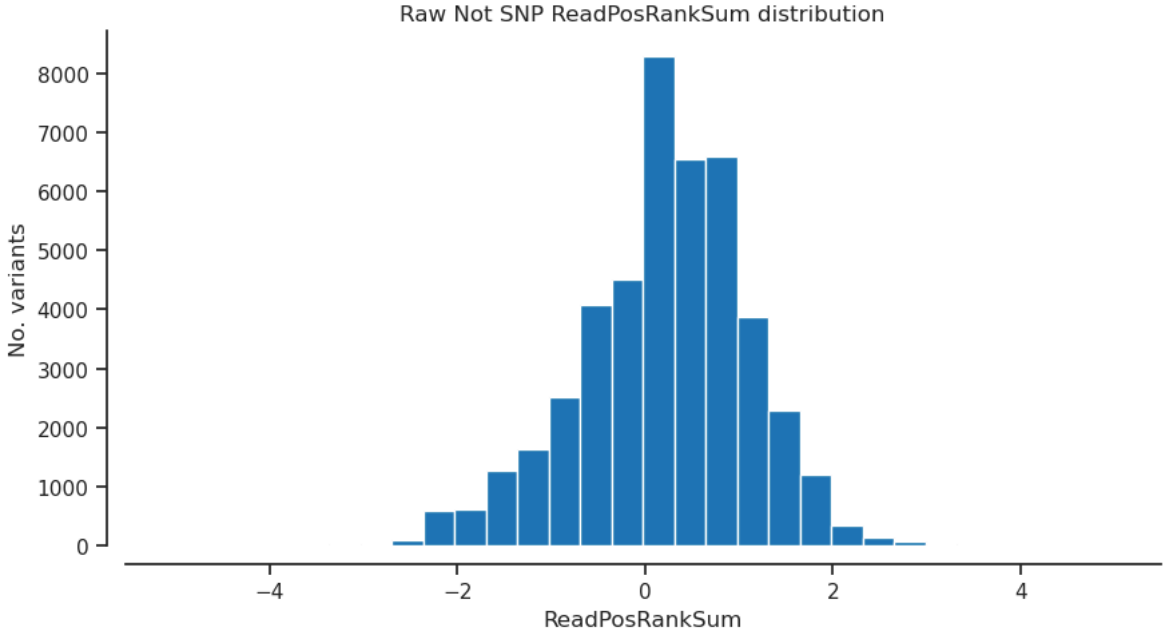
In [37]:
```
plot_hist('ReadPosRankSum','var') # Z-score from Wilcoxon rank sum test o
```



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

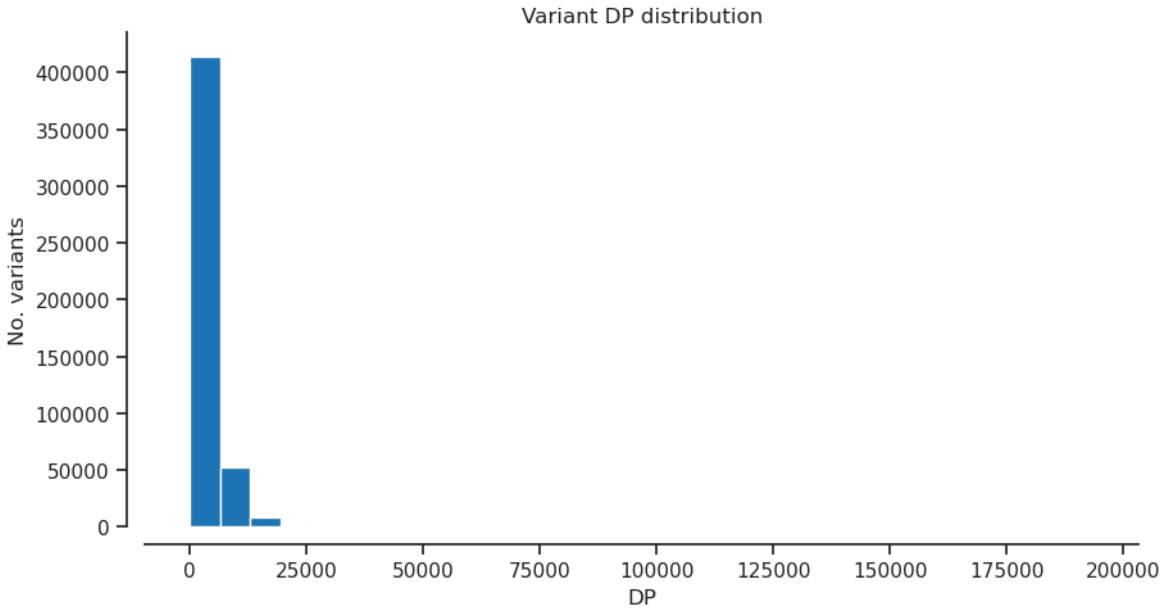


In [39]:
```
plot_hist('ReadPosRankSum','notsnp') # Z-score from Wilcoxon rank sum tes
```

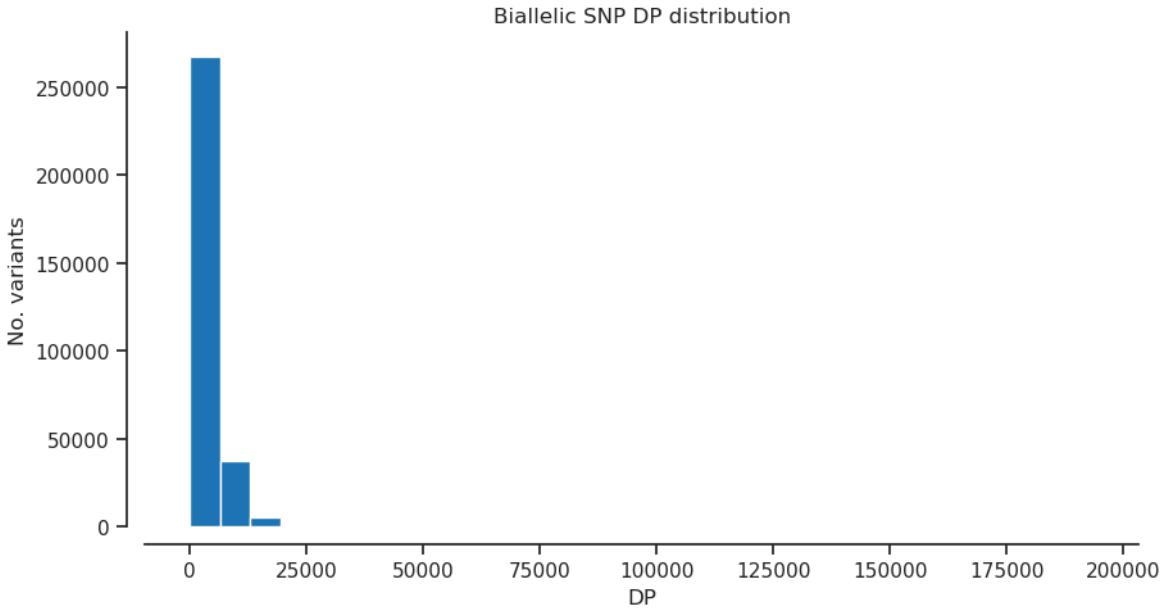


DP - Approximate read depth

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

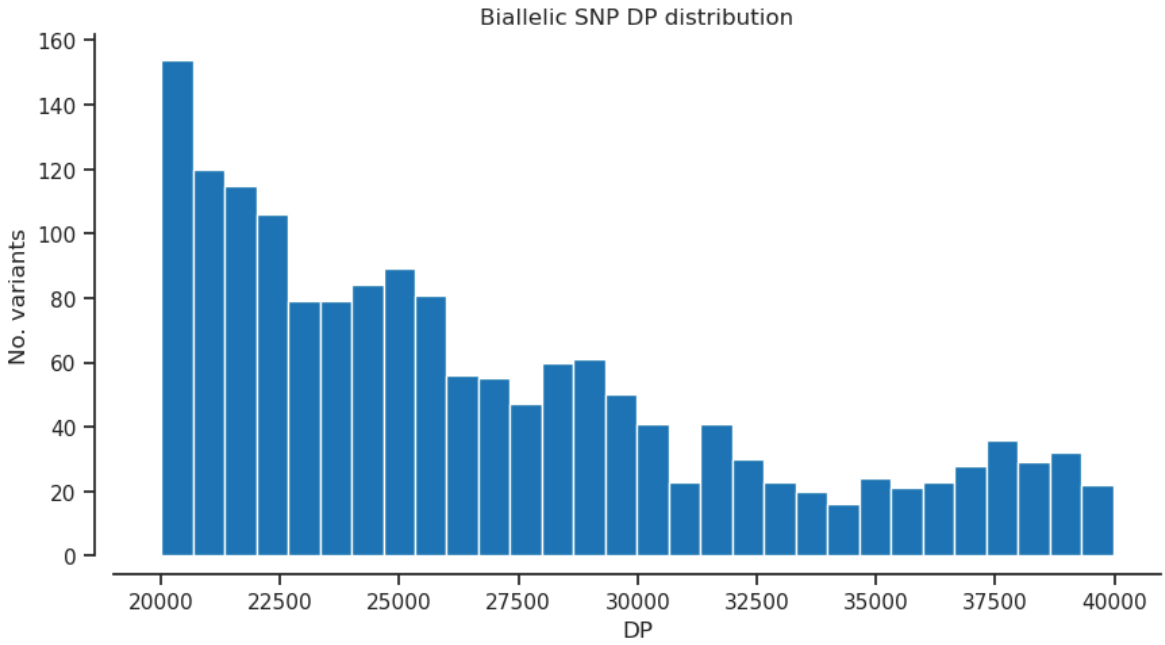


In [41]:
```
plot_hist('DP','biallelic')
```

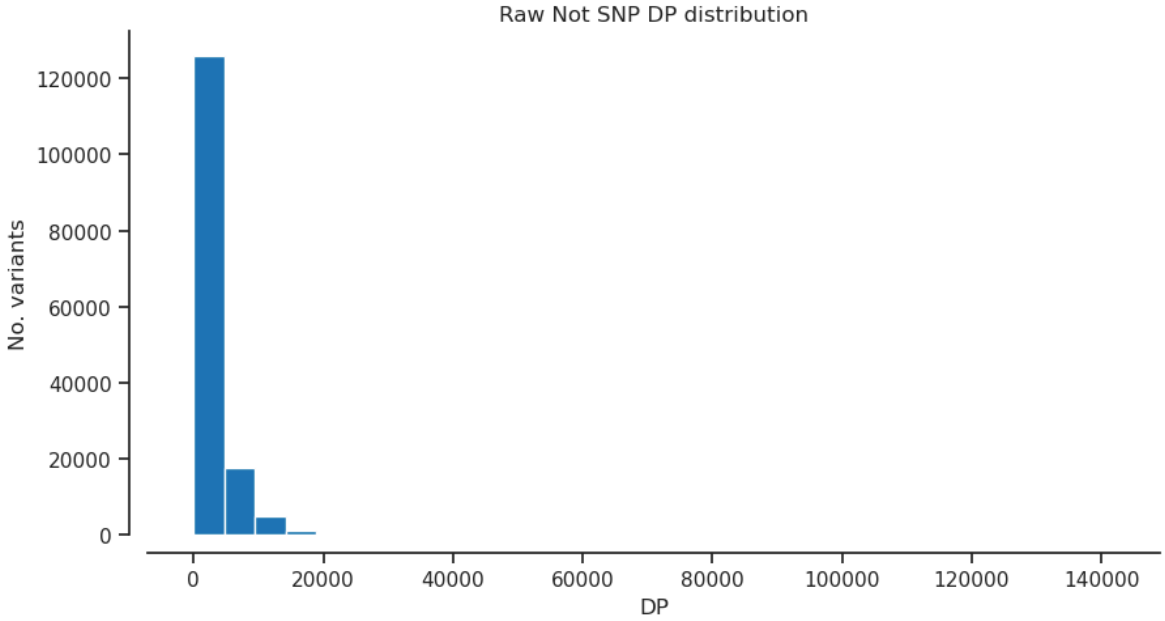


In [42]:
```
filter_expression = '(DP > 20000) & (DP < 40000)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

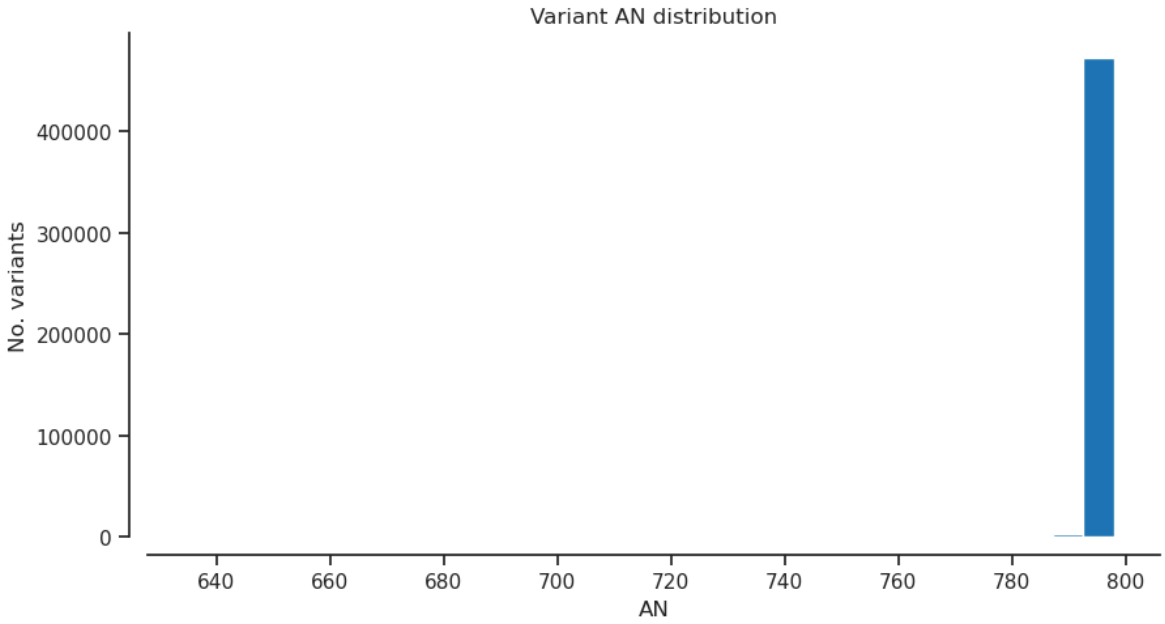


In [44]:
```
plot_hist('DP','notsnp')
```

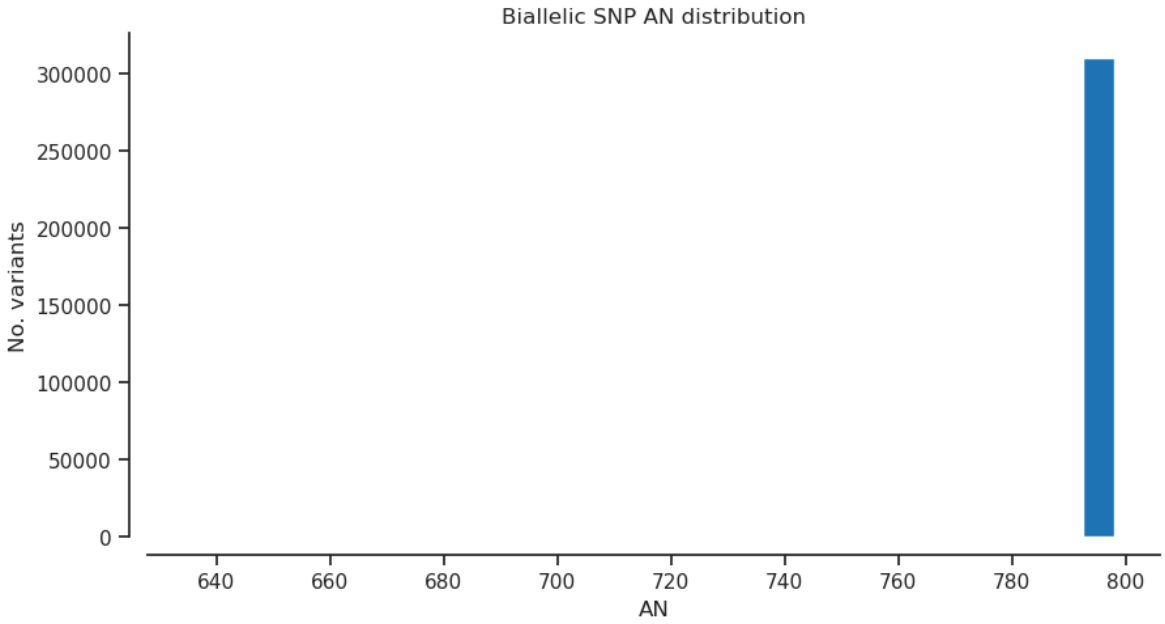


AN - Total number of alleles in called genotypes

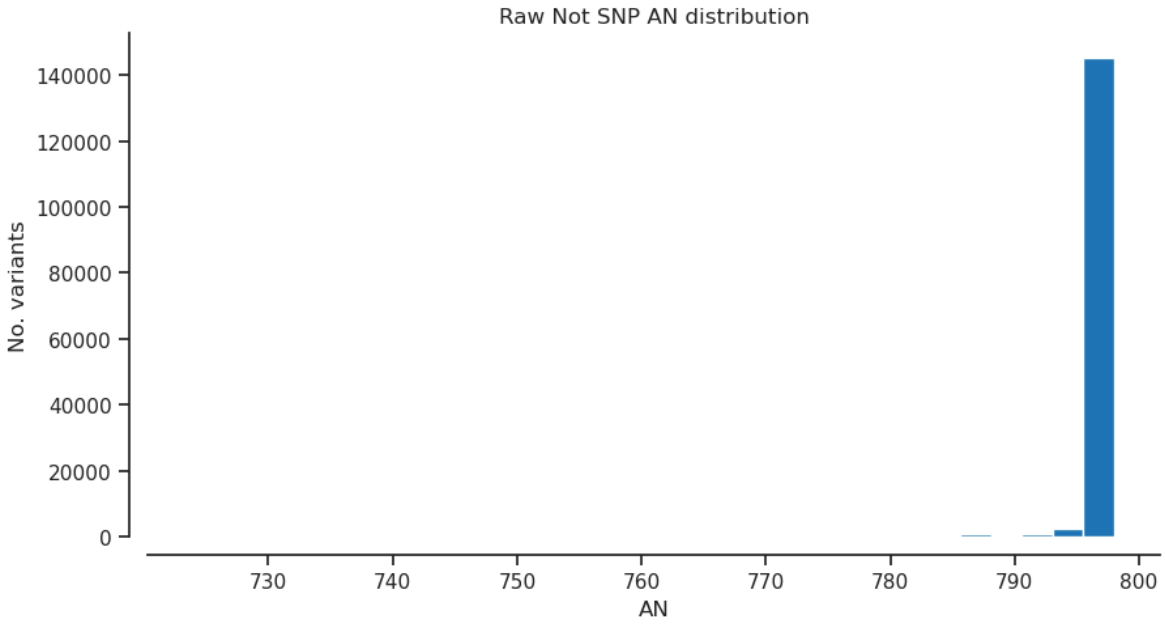
In [45]:
```
plot_hist('AN','var') # Total number of alleles in called genotypes
```



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



In [47]:
```
plot_hist('AN','notsnp') # Total number of alleles in called genotypes
```



Selected filter

In [48]:
```
# 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[48]:
```
250162
```

Genotype

In [49]:
```
calldata_var = callset_var['calldata']
list(calldata_var)
```

Out[49]:
```
['AD', 'DP', 'GQ', 'GT', 'MIN_DP', 'PGT', 'PID', 'PL', 'PS', 'RGQ', 'S
B']
```

In [50]:
```
genotypes_var = allel.GenotypeChunkedArray(calldata_var['GT'])
genotypes_var
```

Out[50]: <GenotypeChunkedArray shape=(477988, 399, 2) dtype=int8 chunks=(65536, 64, 2) nbytes=363.8M cbytes=15.0M cratio=24.3 compression=gzip compression_opts=1 values=h5py._hl.dataset.Dataset>

	0	1	2	3	4	...	394	395	396	397	398
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
...						...					
477985	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/1	0/0
477986	0/0	0/0	0/1	0/0	0/0	...	0/0	0/1	0/0	0/0	0/0
477987	0/0	0/0	0/1	0/0	0/0	...	0/0	0/1	0/0	0/0	0/0

In [51]:
```
# using the selected filters set above
gt_filtered_snps = genotypes_var.subset(variant_selection)
gt_filtered_snps
```

Out[51]: <GenotypeChunkedArray shape=(250162, 399, 2) dtype=int8 chunks=(1955, 399, 2) nbytes=190.4M cbytes=14.9M cratio=12.7 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3	4	...	394	395	396	397	398
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
...						...					
250159	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
250160	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
250161	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0

In [52]:
```
# grab the allele counts for the populations
ac = gt_filtered_snps.count_alleles()
ac
```

Out[52]: <AlleleCountsChunkedArray shape=(250162, 4) dtype=int32 chunks=(31271, 4) nbytes=3.8M cbytes=706.1K cratio=5.5 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3
0	796	2	0	0
1	795	3	0	0
2	796	2	0	0
...		...		
250159	796	2	0	0
250160	794	4	0	0
250161	794	4	0	0

In [53]:
```
ac[:]
```

Out[53]: <AlleleCountsArray shape=(250162, 4) dtype=int32>

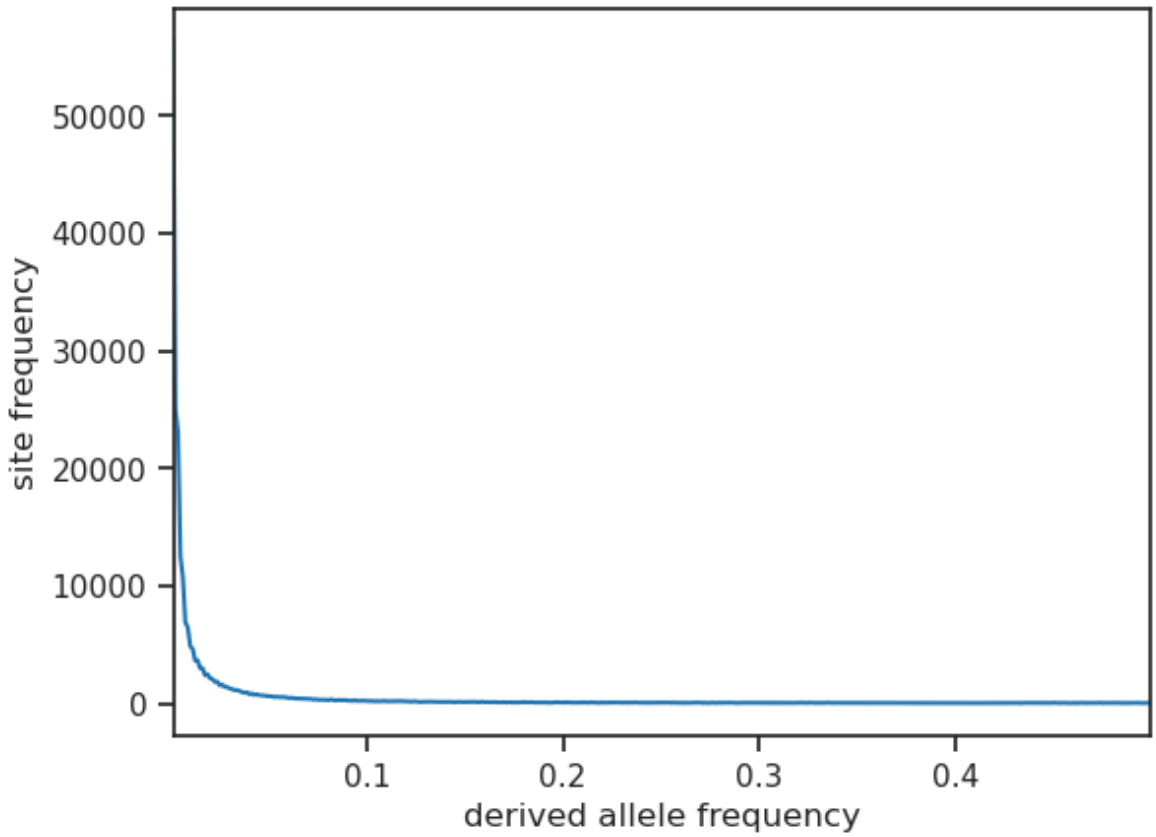
	0	1	2	3
0	796	2	0	0
1	795	3	0	0
2	796	2	0	0
...		...		
250159	796	2	0	0
250160	794	4	0	0
250161	794	4	0	0

In [54]:
```
# 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[54]:
```
array([[796,   2],
       [795,   3],
       [796,   2],
       ...,
       [796,   2],
       [794,   4],
       [794,   4]], dtype=int32)
```

In [55]:
```
# 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[55]:
```
<Axes: xlabel='derived allele frequency', ylabel='site frequency'>
```



In [56]:
```
biallelic = (ac.max_allele() == 1)
###This is the filter expression for biallelic sites
biallelic
```

Out[56]:
```
<ChunkedArrayWrapper shape=(250162,) dtype=bool chunks=(250162,)
   nbytes=244.3K cbytes=47.1K cratio=5.2
   compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shu
ffle': 1, 'blocksize': 0}
   values=zarr.core.Array>
```

In [57]:
```
# select only the biallelic variants
gt_biallelic = gt_filtered_snps.compress(biallelic)
gt_biallelic
```

Out[57]: <GenotypeChunkedArray shape=(236210, 399, 2) dtype=int8 chunks=(1846, 399, 2) nbytes=179.8M cbytes=13.4M cratio=13.4 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3	4	...	394	395	396	397	398
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
...	...										
236207	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
236208	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
236209	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0

In [58]:
```
n_variants = len(gt_biallelic)
n_variants
```

Out[58]: 236210

In [59]:
```
pc_missing = gt_biallelic.count_missing(axis=0)[:] * 100 / n_variants
pc_het = gt_biallelic.count_het(axis=0)[:] * 100 / n_variants
```

Samples

In [60]:
```
samples_var = callset_var['samples']
samples_var = list(samples_var)
samples_var
```

Out[60]:
```
[b'AUT00032-101',
 b'AUT00032-102',
 b'AUT00032-103',
 b'AUT00032-104',
 b'AUT00032-105',
 b'AUT00032-106',
 b'AUT00032-107',
 b'AUT00032-108',
 b'AUT00032-109',
 b'AUT00032-110',
 b'AUT00032-111',
 b'AUT00032-112',
 b'AUT00032-113',
 b'AUT00032-114',
 b'AUT00032-115',
 b'AUT00032-116',
 b'AUT00032-117',
 b'AUT00032-118',
 b'AUT00032-119',
 b'AUT00032-120',
 b'AUT00032-121',
 b'AUT00032-122',
 b'AUT00032-123',
 b'AUT00032-124',
 b'AUT00032-125',
 b'BIH00016-001',
 b'BIH00016-002',
 b'BIH00016-003',
 b'BIH00016-004',
 b'BIH00016-005',
 b'BIH00016-006',
 b'BIH00016-007',
 b'BIH00016-008',
 b'BIH00016-009',
 b'BIH00016-010',
 b'BIH00016-011',
 b'BIH00016-012',
 b'BIH00016-013',
 b'BIH00016-014',
 b'BIH00016-015',
 b'BIH00016-016',
 b'BIH00016-017',
 b'BIH00016-018',
 b'BIH00016-019',
 b'BIH00016-020',
 b'BIH00016-021',
 b'BIH00016-022',
 b'BIH00016-023',
 b'BIH00016-024',
 b'BIH00016-025',
 b'DEU00103-001',
 b'DEU00103-002',
 b'DEU00103-003',
 b'DEU00103-004',
 b'DEU00103-005',
 b'DEU00103-006',
 b'DEU00103-007',
 b'DEU00103-008',
 b'DEU00103-009',
 b'DEU00103-010',
```

```
b'DEU00103-011',
b'DEU00103-012',
b'DEU00103-013',
b'DEU00103-014',
b'DEU00103-015',
b'DEU00103-016',
b'DEU00103-017',
b'DEU00103-018',
b'DEU00103-019',
b'DEU00103-020',
b'DEU00103-021',
b'DEU00103-022',
b'DEU00103-023',
b'DEU00103-024',
b'DEU00103-025',
b'DEU00152-001',
b'DEU00152-002',
b'DEU00152-003',
b'DEU00152-004',
b'DEU00152-005',
b'DEU00152-006',
b'DEU00152-007',
b'DEU00152-008',
b'DEU00152-009',
b'DEU00152-010',
b'DEU00152-011',
b'DEU00152-012',
b'DEU00152-013',
b'DEU00152-014',
b'DEU00152-015',
b'DEU00152-016',
b'DEU00152-017',
b'DEU00152-018',
b'DEU00152-019',
b'DEU00152-020',
b'DEU00152-021',
b'DEU00152-022',
b'DEU00152-023',
b'DEU00152-024',
b'DEU00152-025',
b'FIN00010-001',
b'FIN00010-002',
b'FIN00010-003',
b'FIN00010-004',
b'FIN00010-005',
b'FIN00010-006',
b'FIN00010-007',
b'FIN00010-008',
b'FIN00010-009',
b'FIN00010-010',
b'FIN00010-011',
b'FIN00010-012',
b'FIN00010-013',
b'FIN00010-014',
b'FIN00010-015',
b'FIN00010-016',
b'FIN00010-017',
b'FIN00010-018',
b'FIN00010-019',
b'FIN00010-020',
```

```
b'FIN00010-021',
b'FIN00010-022',
b'FIN00010-023',
b'FIN00010-024',
b'FIN00010-025',
b'FIN00045-001',
b'FIN00045-002',
b'FIN00045-003',
b'FIN00045-004',
b'FIN00045-005',
b'FIN00045-006',
b'FIN00045-007',
b'FIN00045-008',
b'FIN00045-009',
b'FIN00045-010',
b'FIN00045-011',
b'FIN00045-012',
b'FIN00045-013',
b'FIN00045-014',
b'FIN00045-015',
b'FIN00045-016',
b'FIN00045-017',
b'FIN00045-018',
b'FIN00045-019',
b'FIN00045-020',
b'FIN00045-021',
b'FIN00045-022',
b'FIN00045-023',
b'FIN00045-024',
b'FIN00045-025',
b'FRA00083-001',
b'FRA00083-002',
b'FRA00083-003',
b'FRA00083-004',
b'FRA00083-005',
b'FRA00083-006',
b'FRA00083-007',
b'FRA00083-008',
b'FRA00083-009',
b'FRA00083-010',
b'FRA00083-011',
b'FRA00083-012',
b'FRA00083-013',
b'FRA00083-014',
b'FRA00083-015',
b'FRA00083-016',
b'FRA00083-017',
b'FRA00083-018',
b'FRA00083-019',
b'FRA00083-020',
b'FRA00083-021',
b'FRA00083-022',
b'FRA00083-023',
b'FRA00083-024',
b'FRA00083-025',
b'FRA00092-001',
b'FRA00092-002',
b'FRA00092-003',
b'FRA00092-004',
b'FRA00092-005',
```

```
b'FRA00092-006',
b'FRA00092-007',
b'FRA00092-008',
b'FRA00092-009',
b'FRA00092-010',
b'FRA00092-011',
b'FRA00092-012',
b'FRA00092-013',
b'FRA00092-014',
b'FRA00092-015',
b'FRA00092-016',
b'FRA00092-017',
b'FRA00092-018',
b'FRA00092-019',
b'FRA00092-020',
b'FRA00092-021',
b'FRA00092-022',
b'FRA00092-023',
b'FRA00092-024',
b'ITA00014-001',
b'ITA00014-002',
b'ITA00014-003',
b'ITA00014-004',
b'ITA00014-005',
b'ITA00014-006',
b'ITA00014-007',
b'ITA00014-008',
b'ITA00014-009',
b'ITA00014-010',
b'ITA00014-011',
b'ITA00014-012',
b'ITA00014-013',
b'ITA00014-014',
b'ITA00014-015',
b'ITA00014-016',
b'ITA00014-017',
b'ITA00014-018',
b'ITA00014-019',
b'ITA00014-020',
b'ITA00014-021',
b'ITA00014-022',
b'ITA00014-023',
b'ITA00014-024',
b'ITA00014-025',
b'ITA00050-001',
b'ITA00050-002',
b'ITA00050-003',
b'ITA00050-004',
b'ITA00050-005',
b'ITA00050-006',
b'ITA00050-007',
b'ITA00050-008',
b'ITA00050-009',
b'ITA00050-010',
b'ITA00050-011',
b'ITA00050-012',
b'ITA00050-013',
b'ITA00050-014',
b'ITA00050-015',
b'ITA00050-016',
```

```
b'ITA00050-017',
b'ITA00050-018',
b'ITA00050-019',
b'ITA00050-020',
b'ITA00050-021',
b'ITA00050-022',
b'ITA00050-023',
b'ITA00050-024',
b'ITA00050-025',
b'ITA00260-101',
b'ITA00260-102',
b'ITA00260-103',
b'ITA00260-104',
b'ITA00260-105',
b'ITA00260-106',
b'ITA00260-107',
b'ITA00260-108',
b'ITA00260-109',
b'ITA00260-110',
b'ITA00260-111',
b'ITA00260-112',
b'ITA00260-113',
b'ITA00260-114',
b'ITA00260-115',
b'ITA00260-116',
b'ITA00260-117',
b'ITA00260-118',
b'ITA00260-119',
b'ITA00260-120',
b'ITA00260-121',
b'ITA00260-122',
b'ITA00260-123',
b'ITA00260-124',
b'ITA00260-125',
b'LTU00120-001',
b'LTU00120-002',
b'LTU00120-003',
b'LTU00120-004',
b'LTU00120-005',
b'LTU00120-006',
b'LTU00120-007',
b'LTU00120-008',
b'LTU00120-009',
b'LTU00120-010',
b'LTU00120-011',
b'LTU00120-012',
b'LTU00120-013',
b'LTU00120-014',
b'LTU00120-015',
b'LTU00120-016',
b'LTU00120-017',
b'LTU00120-018',
b'LTU00120-019',
b'LTU00120-020',
b'LTU00120-021',
b'LTU00120-022',
b'LTU00120-023',
b'LTU00120-024',
b'LTU00120-025',
b'ROU00058-001',
```

```
b'ROU00058-002',
b'ROU00058-003',
b'ROU00058-004',
b'ROU00058-005',
b'ROU00058-006',
b'ROU00058-007',
b'ROU00058-008',
b'ROU00058-009',
b'ROU00058-010',
b'ROU00058-011',
b'ROU00058-012',
b'ROU00058-013',
b'ROU00058-014',
b'ROU00058-015',
b'ROU00058-016',
b'ROU00058-017',
b'ROU00058-018',
b'ROU00058-019',
b'ROU00058-020',
b'ROU00058-021',
b'ROU00058-022',
b'ROU00058-023',
b'ROU00058-024',
b'ROU00058-025',
b'ROU00415-101',
b'ROU00415-102',
b'ROU00415-103',
b'ROU00415-104',
b'ROU00415-105',
b'ROU00415-106',
b'ROU00415-107',
b'ROU00415-108',
b'ROU00415-109',
b'ROU00415-110',
b'ROU00415-111',
b'ROU00415-112',
b'ROU00415-113',
b'ROU00415-114',
b'ROU00415-115',
b'ROU00415-116',
b'ROU00415-117',
b'ROU00415-118',
b'ROU00415-119',
b'ROU00415-120',
b'ROU00415-121',
b'ROU00415-122',
b'ROU00415-123',
b'ROU00415-124',
b'ROU00415-125',
b'SVN00002-001',
b'SVN00002-002',
b'SVN00002-003',
b'SVN00002-004',
b'SVN00002-005',
b'SVN00002-006',
b'SVN00002-007',
b'SVN00002-008',
b'SVN00002-009',
b'SVN00002-010',
b'SVN00002-011',
```

```
 b'SVN00002-012',
 b'SVN00002-013',
 b'SVN00002-014',
 b'SVN00002-015',
 b'SVN00002-016',
 b'SVN00002-017',
 b'SVN00002-018',
 b'SVN00002-019',
 b'SVN00002-020',
 b'SVN00002-021',
 b'SVN00002-022',
 b'SVN00002-023',
 b'SVN00002-024',
 b'SVN00002-025',
 b'SVN00024-001',
 b'SVN00024-002',
 b'SVN00024-003',
 b'SVN00024-004',
 b'SVN00024-005',
 b'SVN00024-006',
 b'SVN00024-007',
 b'SVN00024-008',
 b'SVN00024-009',
 b'SVN00024-010',
 b'SVN00024-011',
 b'SVN00024-012',
 b'SVN00024-013',
 b'SVN00024-014',
 b'SVN00024-015',
 b'SVN00024-016',
 b'SVN00024-017',
 b'SVN00024-018',
 b'SVN00024-019',
 b'SVN00024-020',
 b'SVN00024-021',
 b'SVN00024-022',
 b'SVN00024-023',
 b'SVN00024-024',
 b'SVN00024-025']
```

In [65]:
```
samples_fn = '~/scratch/data/Pabies/Picea_abies_sample_list_scikit-allel.
samples = pandas.read_csv(samples_fn, sep='\t')
samples
```

Out[65]:

	ID	Population
0	AUT00032-101	AUT00032
1	AUT00032-102	AUT00032
2	AUT00032-103	AUT00032
3	AUT00032-104	AUT00032
4	AUT00032-105	AUT00032
...	...	...
394	SVN00024-021	SVN00024
395	SVN00024-022	SVN00024
396	SVN00024-023	SVN00024
397	SVN00024-024	SVN00024
398	SVN00024-025	SVN00024

399 rows × 2 columns

In [66]:
```
samples.Population.value_counts()
```

Out[66]:
```
Population
AUT00032    25
BIH00016    25
DEU00103    25
DEU00152    25
FIN00010    25
FIN00045    25
FRA00083    25
ITA00014    25
ROU00058    25
ITA00050    25
ITA00260    25
LTU00120    25
SVN00002    25
ROU00415    25
SVN00024    25
FRA00092    24
Name: count, dtype: int64
```

In [67]:
```
populations = samples.Population.unique()
populations
###This identifiers come from the metadata file
```

Out[67]:
```
array(['AUT00032', 'BIH00016', 'DEU00103', 'DEU00152', 'FIN00010',
       'FIN00045', 'FRA00083', 'FRA00092', 'ITA00014', 'ITA00050',
       'ITA00260', 'LTU00120', 'ROU00058', 'ROU00415', 'SVN00002',
       'SVN00024'], dtype=object)
```

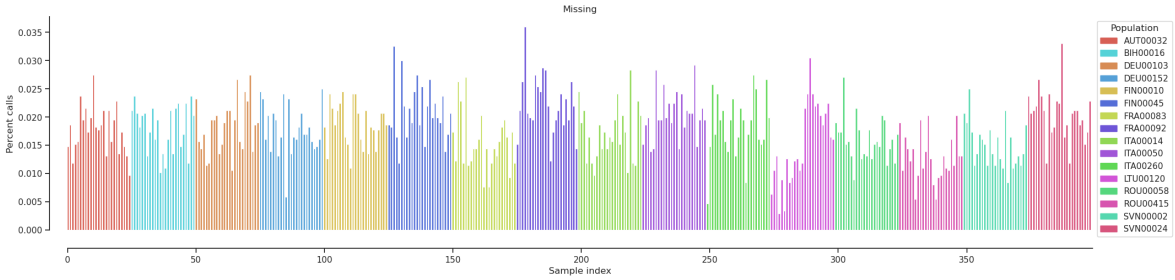
Gt frequency function

In [68]:
```
def plot_genotype_frequency(pc, title):
    fig, ax = plt.subplots(figsize=(24, 5))
```

```
    sns.despine(ax=ax, offset=24)
    left = np.arange(len(pc))
    palette = sns.color_palette("hls", 16)
    pop2color = {'AUT00032': palette[0],
                 'BIH00016': palette[8],
                 'DEU00103': palette[1],
                 'DEU00152': palette[9],
                 'FIN00010': palette[2],
                 'FIN00045': palette[10],
                 'FRA00083': palette[3],
                 'FRA00092': palette[11],
                 'ITA00014': palette[4],
                 'ITA00050': palette[12],
                 'ITA00260': palette[5],
                 'LTU00120': palette[13],
                 'ROU00058': palette[6],
                 'ROU00415': palette[14],
                 'SVN00002': palette[7],
                'SVN00024': palette[15]}
    colors = [pop2color[p] for p in samples.Population]
    ax.bar(left, pc, color=colors)
    ax.set_xlim(0, len(pc))
    ax.set_xlabel('Sample index')
    ax.set_ylabel('Percent calls')
    ax.set_title(title)
    handles = [mpl.patches.Patch(color=palette[0]),
               mpl.patches.Patch(color=palette[8]),
               mpl.patches.Patch(color=palette[1]),
               mpl.patches.Patch(color=palette[9]),
               mpl.patches.Patch(color=palette[2]),
               mpl.patches.Patch(color=palette[10]),
               mpl.patches.Patch(color=palette[3]),
               mpl.patches.Patch(color=palette[11]),
               mpl.patches.Patch(color=palette[4]),
               mpl.patches.Patch(color=palette[12]),
               mpl.patches.Patch(color=palette[5]),
               mpl.patches.Patch(color=palette[13]),
               mpl.patches.Patch(color=palette[6]),
               mpl.patches.Patch(color=palette[14]),
               mpl.patches.Patch(color=palette[7]),
              mpl.patches.Patch(color=palette[15])]
    ax.legend(handles=handles, labels=['AUT00032', 'BIH00016', 'DEU00103'
       'FIN00045', 'FRA00083', 'FRA00092', 'ITA00014', 'ITA00050',
       'ITA00260', 'LTU00120', 'ROU00058', 'ROU00415', 'SVN00002',
       'SVN00024'], title='Population',
              bbox_to_anchor=(1, 1), loc='upper left')
```

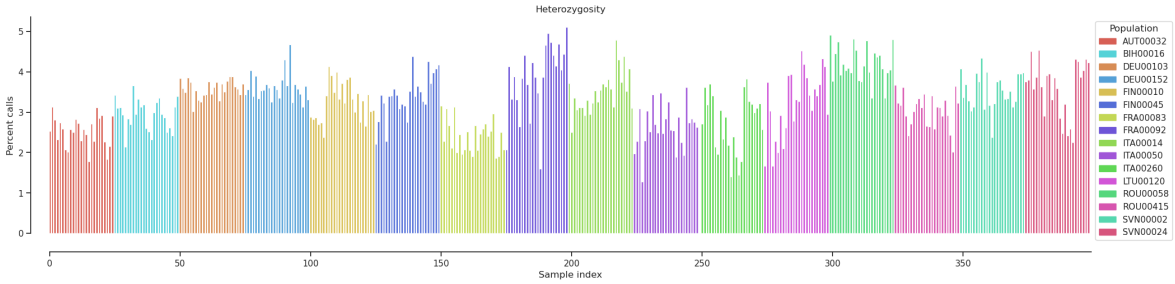
Plot missing

In [69]:
```
plot_genotype_frequency(pc_missing, 'Missing')
```



Plot heterozygosity

In [70]:
```
plot_genotype_frequency(pc_het, 'Heterozygosity')
```



PCA

In [72]:
```
palette = sns.color_palette("hls",16)
pop_colours = {
                'AUT00032': palette[0],
                 'BIH00016': palette[8],
                 'DEU00103': palette[1],
                 'DEU00152': palette[9],
                 'FIN00010': palette[2],
                 'FIN00045': palette[10],
                 'FRA00083': palette[3],
                 'FRA00092': palette[11],
                 'ITA00014': palette[4],
                 'ITA00050': palette[12],
                 'ITA00260': palette[5],
                 'LTU00120': palette[13],
                 'ROU00058': palette[6],
                 'ROU00415': palette[14],
                 'SVN00002': palette[7],
                'SVN00024': palette[15]
}
```

In [73]:
```
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 [74]:
```
ac2 = gt_biallelic.count_alleles()
ac2
```

Out[74]: <AlleleCountsChunkedArray shape=(236210, 2) dtype=int32 chunks=(29527, 2) nbytes=1.8M cbytes=530.2K cratio=3.5 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1
0	796	2
1	795	3
2	796	2
...	...	
236207	796	2
236208	794	4
236209	794	4

In [75]:
```
flt = (ac2[:, :2].min(axis=1) > 1)
gf = gt_biallelic.compress(flt, axis=0)
gn = gf.to_n_alt()
gn
```

Out[75]:
```
<ChunkedArrayWrapper shape=(179863, 399) dtype=int8 chunks=(2811, 399)
   nbytes=68.4M cbytes=9.4M cratio=7.3
   compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shu
ffle': 1, 'blocksize': 0}
   values=zarr.core.Array>
```

In [76]:
```
coords1, model1 = allel.pca(gn, n_components=10, scaler='patterson')
```

In [77]:
```
fig_pca(coords1, model1, 'Figure 1. Conventional PCA.')
```

Figure 1. Conventional PCA.