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/Fsylvatica/vcf_filterin
```

Get data

In [3]:
```
callset_var_fn = '/users/mcevoysu/scratch/output/Fsylvatica/scikit-allel/
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=(273617,) 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=46.7M cbytes=10.3M cratio=4.6 values=h5py._hl.group.Group>

	AC	AF	ALT	AN	BaseQRankSum	CHROM	DP
0	[52 -1 -1]	[0.069 nan nan]	[b'C' b'' b'']	750	-0.926	b'Bhaga_1'	2153
1	[6 -1 -1]	[0.007916 nan nan]	[b'G' b'' b'']	750	0.798	b'Bhaga_1'	1952
2	[7 -1 -1]	[0.009235 nan nan]	[b'A' b'' b'']	750	-0.362	b'Bhaga_1'	1490
...							
273614	[21 2 -1]	[0.028 0.002639 nan]	[b'C' b'T' b'']	750	-0.385	b'Bhaga_Unplaced_827'	800
273615	[2 -1 -1]	[0.002639 nan nan]	[b'A' b'' b'']	750	nan	b'Bhaga_Unplaced_827'	791
273616	[1 -1 -1]	[0.001319 nan nan]	[b'A' b'' b'']	750	0.0	b'Bhaga_Unplaced_827'	421

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

Out[7]: <VariantTable shape=(166080,) dtype=(numpy.record, [('AC', '<i4', (3,)), ('AF', '<f4', (3,)), ('ALT', 'O', (3,)), ('AN', '<i4'), ('BaseQRankSum', '<f4'), ('CHROM', 'O'), ('DP', '<i4'), ('END', '<i4'), ('ExcessHet', '<f4'), ('FILTER_LowQual', '?'), ('FILTER_PASS', '?'), ('FS', '<f4'), ('ID', 'O'), ('InbreedingCoeff', '<f4'), ('MLEAC', '<i4', (3,)), ('MLEAF', '<f4', (3,)), ('MQ', '<f4'), ('MQRankSum', '<f4'), ('POS', '<i4'), ('QD', '<f4'), ('QUAL', '<f4'), ('RAW_MQandDP', '<i4', (2,)), ('REF', 'O'), ('ReadPosRankSum', '<f4'), ('SOR', '<f4'), ('altlen', '<i4', (3,)), ('is_snp', '?'), ('numalt', '<i4')])>

	AC	AF	ALT	AN	BaseQRankSum	CHROM	DP
0	[52 -1 -1]	[0.069 nan nan]	[b'C' b'' b'']	750	-0.926	b'Bhaga_1'	2153
1	[6 -1 -1]	[0.007916 nan nan]	[b'G' b'' b'']	750	0.798	b'Bhaga_1'	1952
2	[7 -1 -1]	[0.009235 nan nan]	[b'A' b'' b'']	750	-0.362	b'Bhaga_1'	1490
...							
166077	[21 2 -1]	[0.028 0.002639 nan]	[b'C' b'T' b'']	750	-0.385	b'Bhaga_Unplaced_827'	800
166078	[2 -1 -1]	[0.002639 nan nan]	[b'A' b'' b'']	750	nan	b'Bhaga_Unplaced_827'	791
166079	[1 -1 -1]	[0.001319 nan nan]	[b'A' b'' b'']	750	0.0	b'Bhaga_Unplaced_827'	421

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

Out[8]: <VariantTable shape=(107537,) dtype=(numpy.record, [('AC', '<i4', (3,)), ('AF', '<f4', (3,)), ('ALT', 'O', (3,)), ('AN', '<i4'), ('BaseQRankSum', '<f4'), ('CHROM', 'O'), ('DP', '<i4'), ('END', '<i4'), ('ExcessHet', '<f4'), ('FILTER_LowQual', '?'), ('FILTER_PASS', '?'), ('FS', '<f4'), ('ID', 'O'), ('InbreedingCoeff', '<f4'), ('MLEAC', '<i4', (3,)), ('MLEAF', '<f4', (3,)), ('MQ', '<f4'), ('MQRankSum', '<f4'), ('POS', '<i4'), ('QD', '<f4'), ('QUAL', '<f4'), ('RAW_MQandDP', '<i4', (2,)), ('REF', 'O'), ('ReadPosRankSum', '<f4'), ('SOR', '<f4'), ('altlen', '<i4', (3,)), ('is_snp', '?'), ('numalt', '<i4')])>

	AC	AF	ALT	AN	BaseQRankSum	CHROM	DP
0	[22 -1 -1]	[0.029 nan nan]	[b'*' b'' b'']	750	nan	b'Bhaga_1'	284
1	[16 -1 -1]	[0.021 nan nan]	[b'*' b'' b'']	748	nan	b'Bhaga_1'	68
2	[4 -1 -1]	[0.005277 nan nan]	[b'*' b'' b'']	750	-0.918	b'Bhaga_1'	6154
...							
107534	[3 -1 -1]	[0.003958 nan nan]	[b'*' b'' b'']	750	nan	b'Bhaga_Unplaced_827'	776
107535	[3 -1 -1]	[0.003958 nan nan]	[b'*' b'' b'']	750	nan	b'Bhaga_Unplaced_827'	778
107536	[3 3 2]	[0.003958 0.003958 0.002639]	[b'*' b'A' b'C']	750	0.967	b'Bhaga_Unplaced_827'	787

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, 161207,   4771,    102])
```

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

Out[12]:
```
4873
```

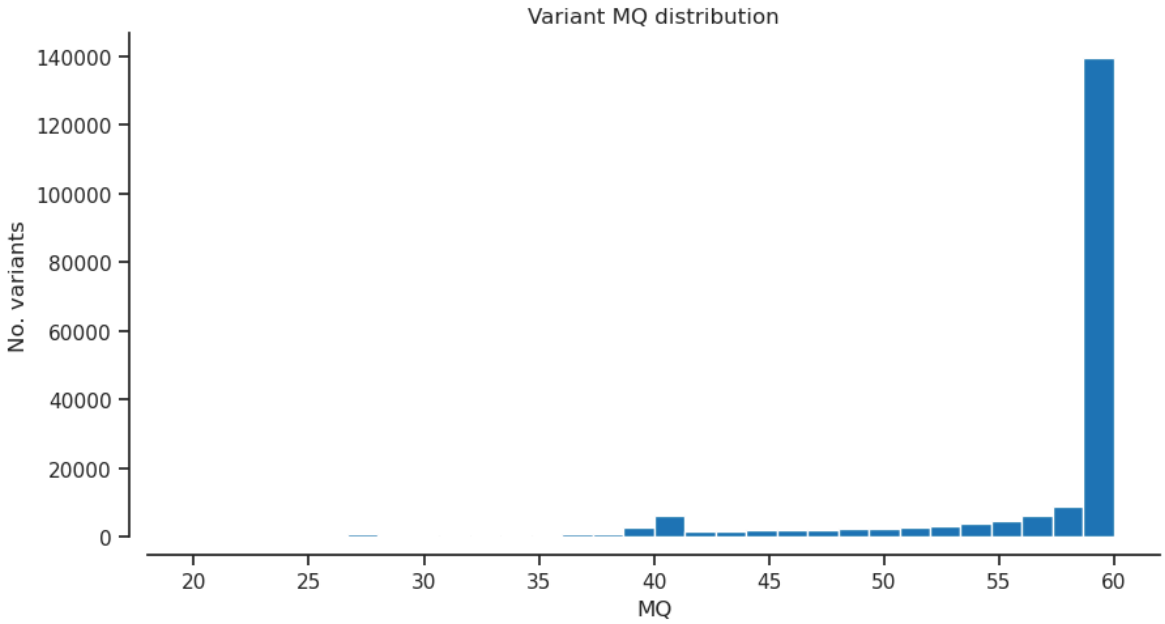
In [13]:
```
filter_expression = '(numalt == 1)'
biallelic_np = rawsnps.query(filter_expression)[:]
biallelic_np
```

Out[13]: <VariantTable shape=(161207,) dtype=(numpy.record, [('AC', '<i4', (3,)), ('AF', '<f4', (3,)), ('ALT', 'O', (3,)), ('AN', '<i4'), ('BaseQRankSum', '<f4'), ('CHROM', 'O'), ('DP', '<i4'), ('END', '<i4'), ('ExcessHet', '<f4'), ('FILTER_LowQual', '?'), ('FILTER_PASS', '?'), ('FS', '<f4'), ('ID', 'O'), ('InbreedingCoeff', '<f4'), ('MLEAC', '<i4', (3,)), ('MLEAF', '<f4', (3,)), ('MQ', '<f4'), ('MQRankSum', '<f4'), ('POS', '<i4'), ('QD', '<f4'), ('QUAL', '<f4'), ('RAW_MQandDP', '<i4', (2,)), ('REF', 'O'), ('ReadPosRankSum', '<f4'), ('SOR', '<f4'), ('altlen', '<i4', (3,)), ('is_snp', '?'), ('numalt', '<i4')])>

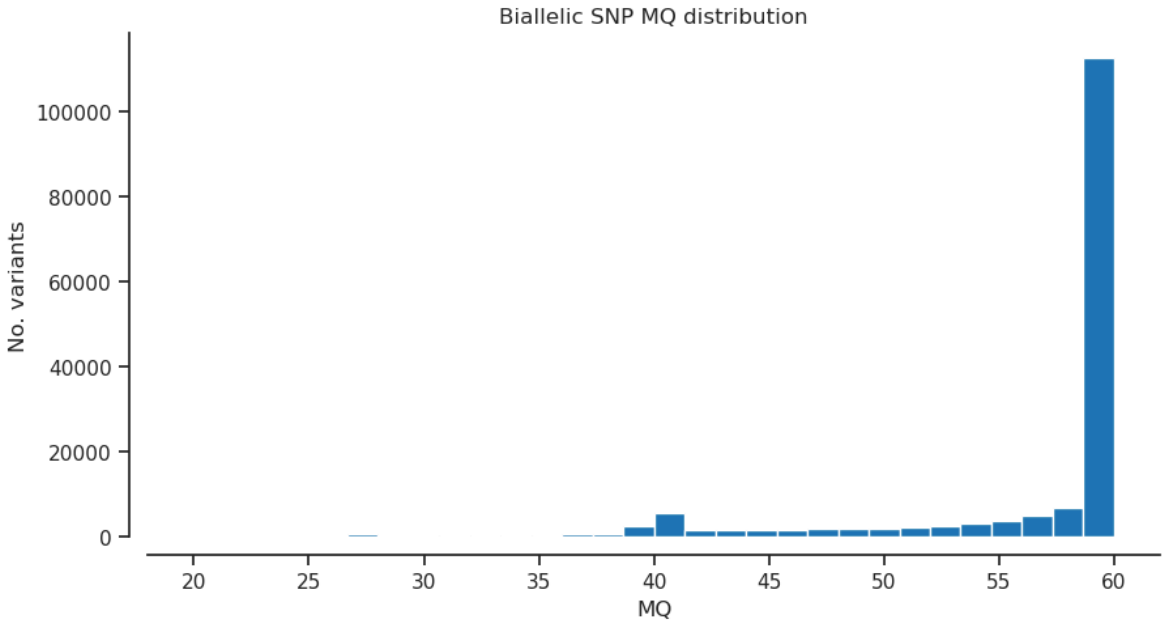
	AC	AF	ALT	AN	BaseQRankSum	CHROM	DP
0	[52 -1 -1]	[0.069 nan nan]	[b'C' b'' b'']	750	-0.926	b'Bhaga_1'	2153
1	[6 -1 -1]	[0.007916 nan nan]	[b'G' b'' b'']	750	0.798	b'Bhaga_1'	1952
2	[7 -1 -1]	[0.009235 nan nan]	[b'A' b'' b'']	750	-0.362	b'Bhaga_1'	1490
...							
161204	[3 -1 -1]	[0.003958 nan nan]	[b'T' b'' b'']	750	-0.524	b'Bhaga_Unplaced_827'	805
161205	[2 -1 -1]	[0.002639 nan nan]	[b'A' b'' b'']	750	nan	b'Bhaga_Unplaced_827'	791
161206	[1 -1 -1]	[0.001319 nan nan]	[b'A' b'' b'']	750	0.0	b'Bhaga_Unplaced_827'	421

MQ - RMS mapping quality

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

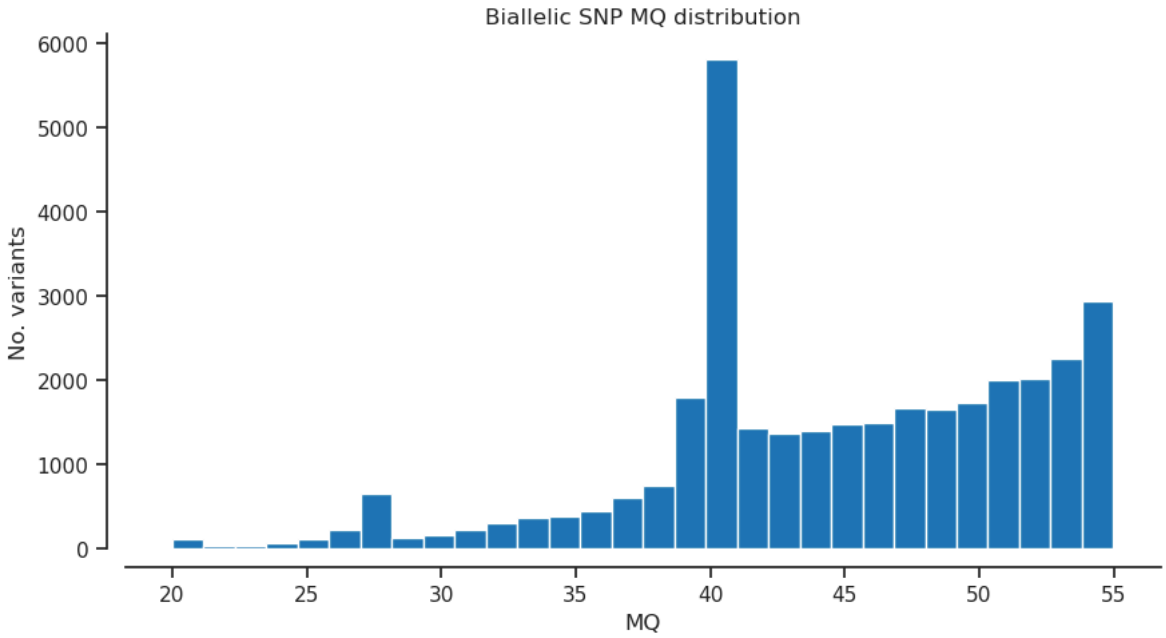


In [15]:
```
plot_hist('MQ','biallelic') # RMS mapping quality
```



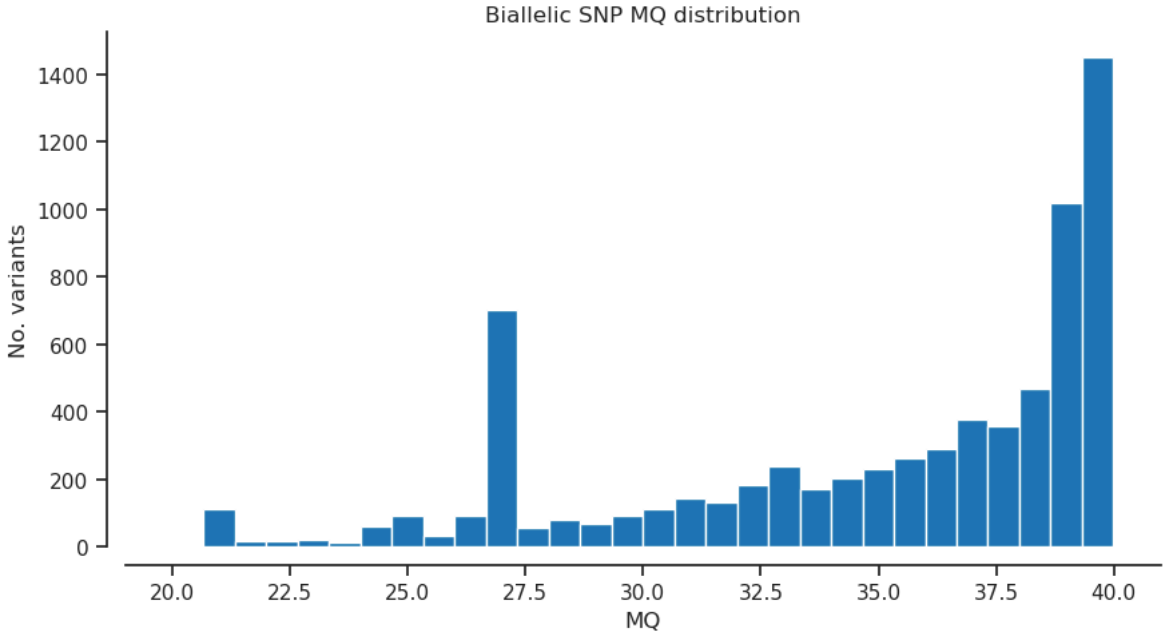
In [16]:
```
filter_expression = '(MQ < 55)'
bi_selection = biallelic_np.query(filter_expression)[:]
#np.count_nonzero(var_selection)
```

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

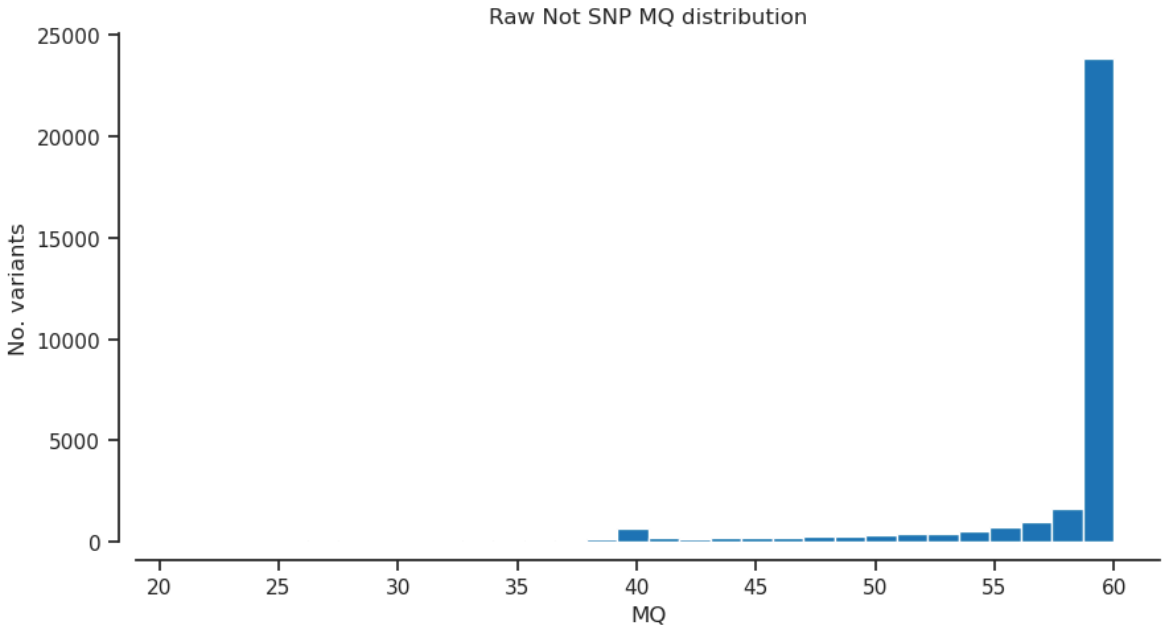


In [18]:
```
filter_expression = '(MQ < 40)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

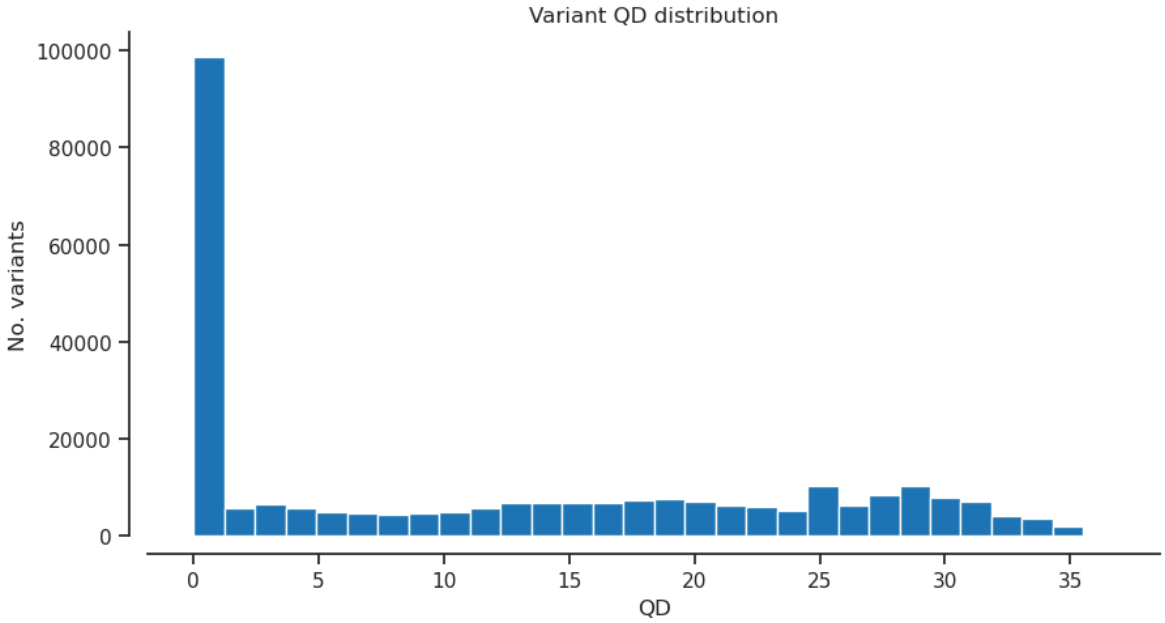


In [20]:
```
plot_hist('MQ','notsnp')
```

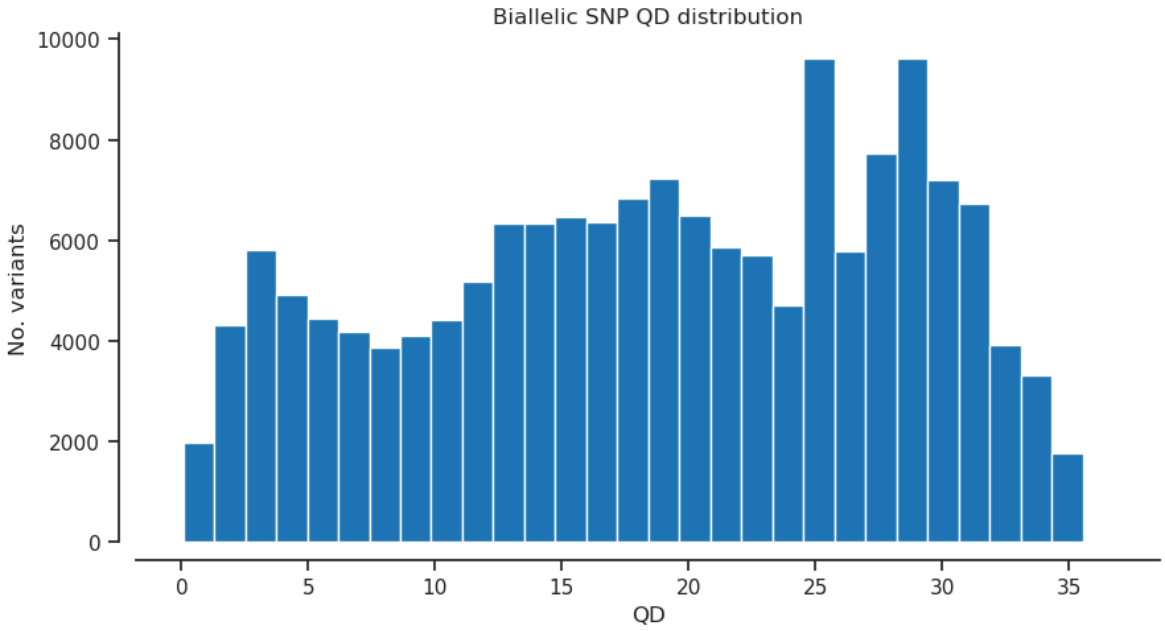


QD - Variant Confidence/Quality by Depth

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

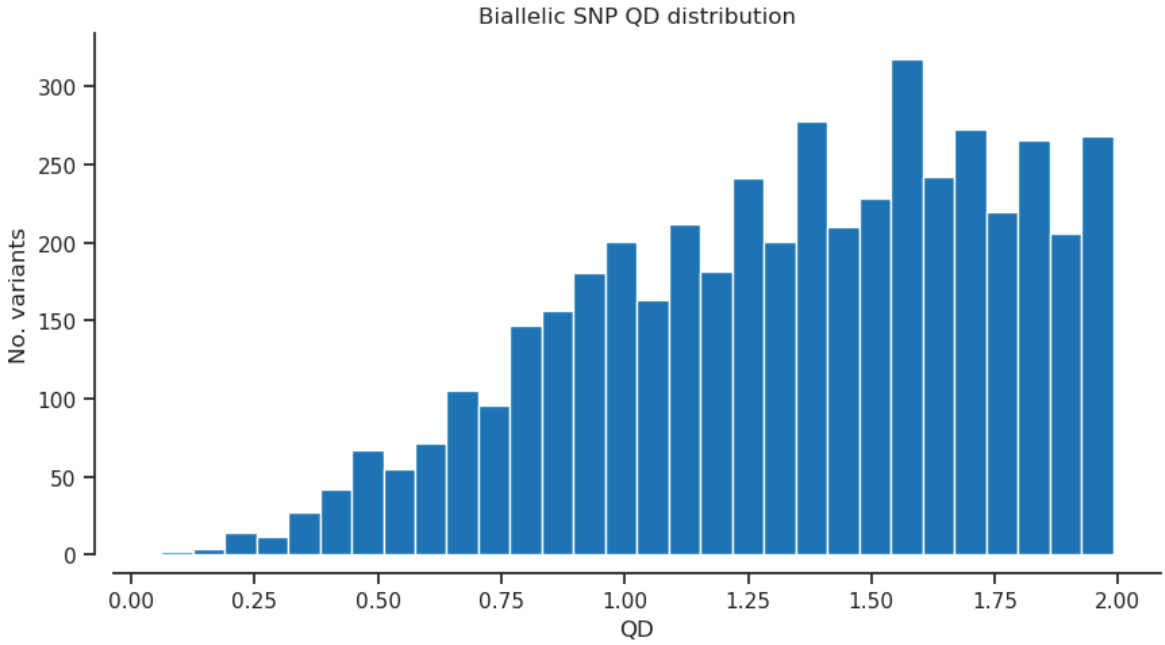


In [22]:
```
plot_hist('QD','biallelic') # Variant Confidence/Quality by Depth
```

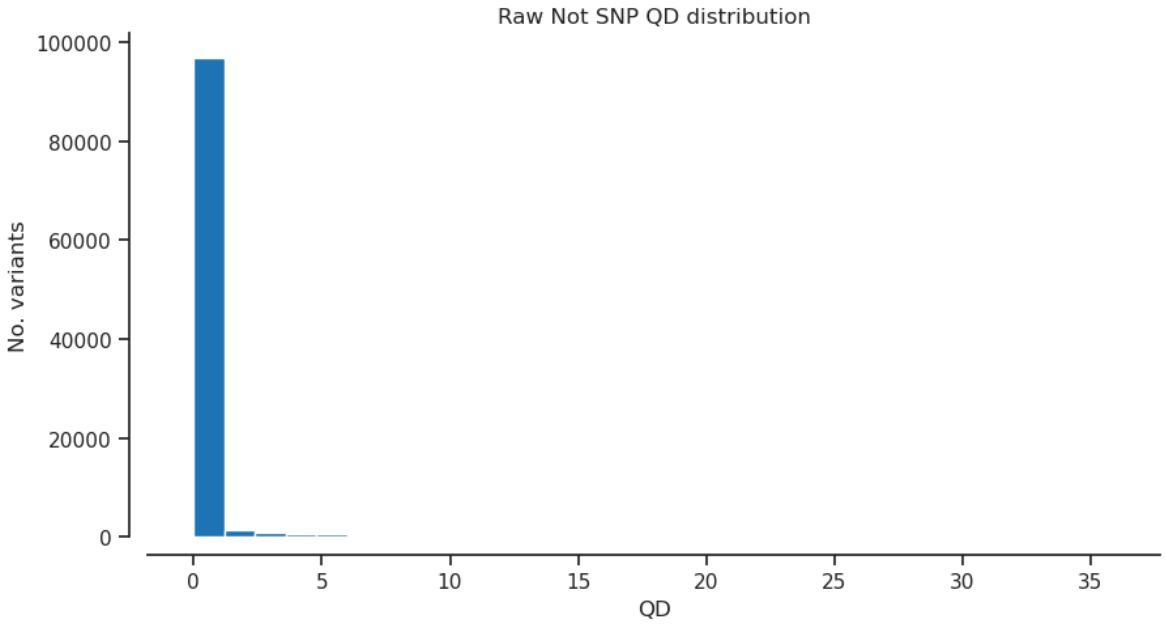


In [23]:
```
filter_expression = '(QD < 2)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

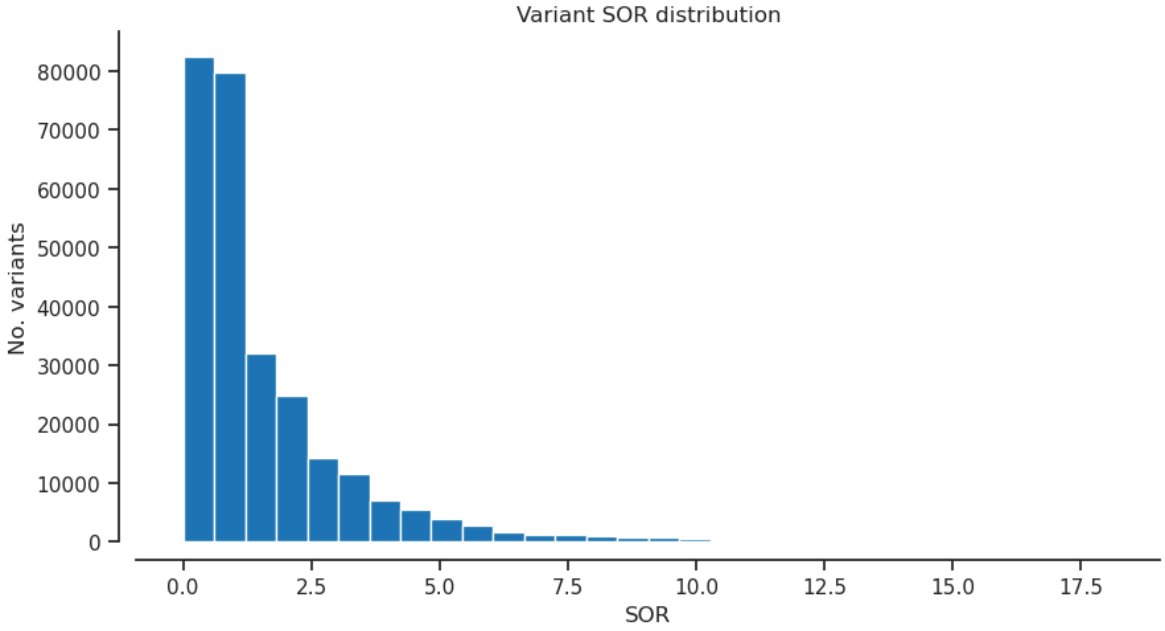


In [25]:
```
plot_hist('QD','notsnp') # Variant Confidence/Quality by Depth
```

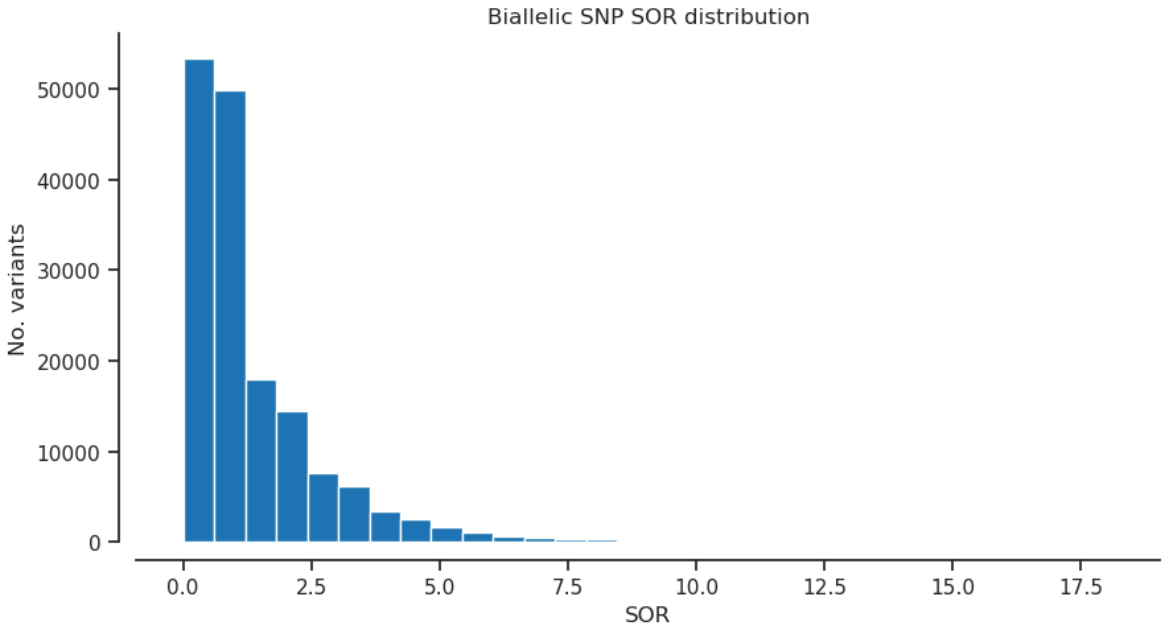


SOR - Symmetric Odds Ratio of 2x2 contingency table to detect strand bias

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

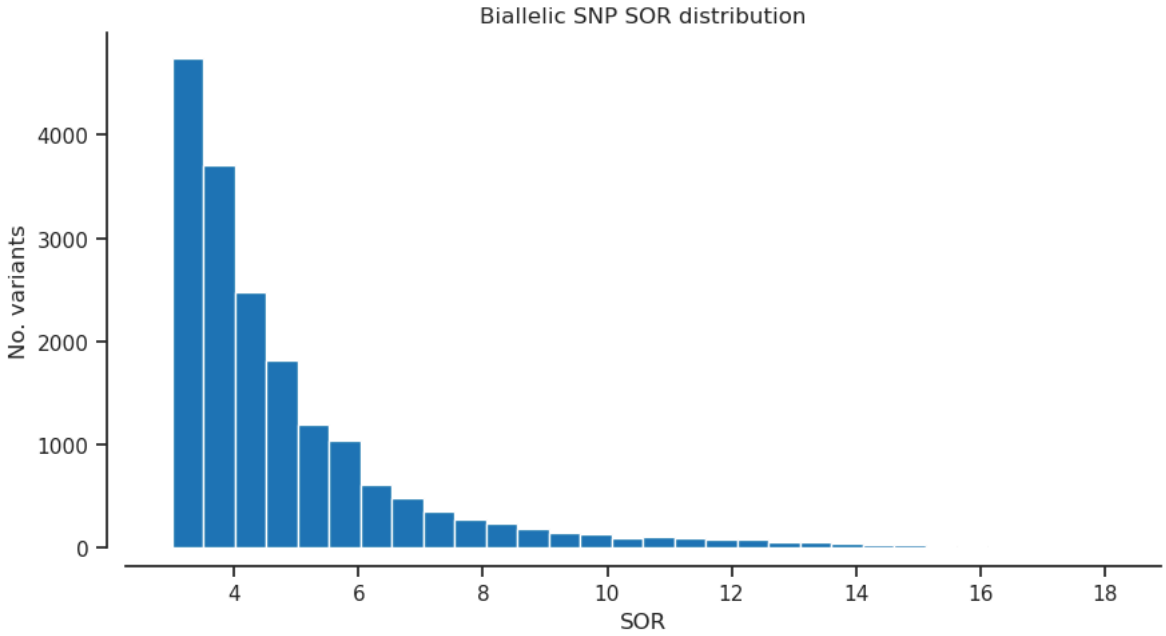


In [27]:
```
plot_hist('SOR','biallelic') # Symmetric Odds Ratio of 2x2 contingency ta
```

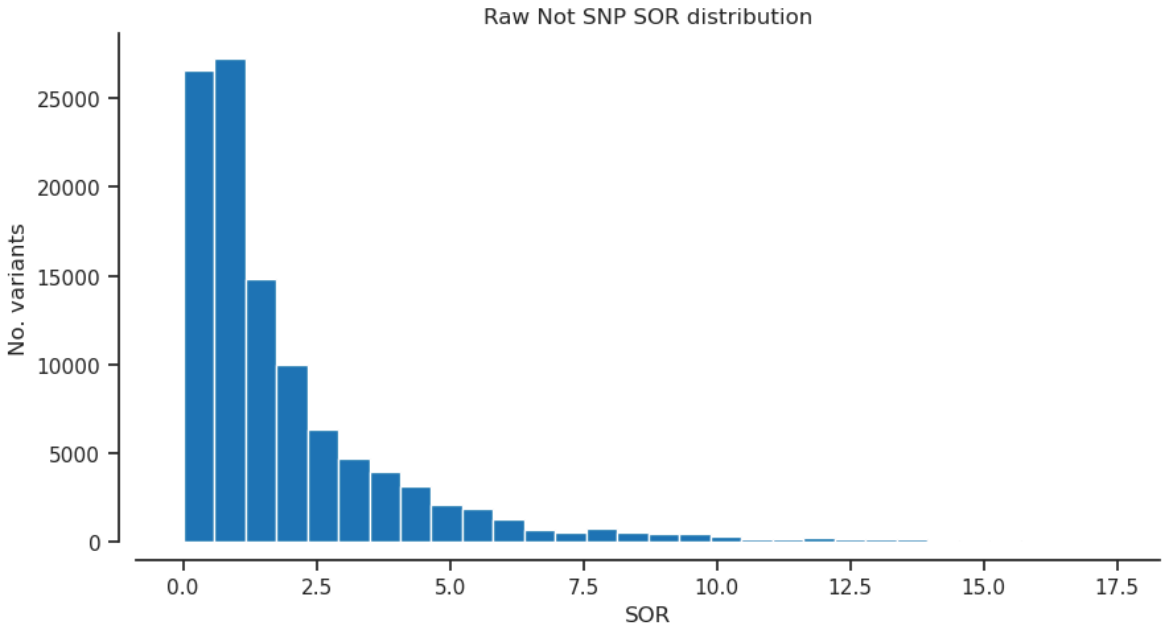


In [28]:
```
filter_expression = '(SOR > 3)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

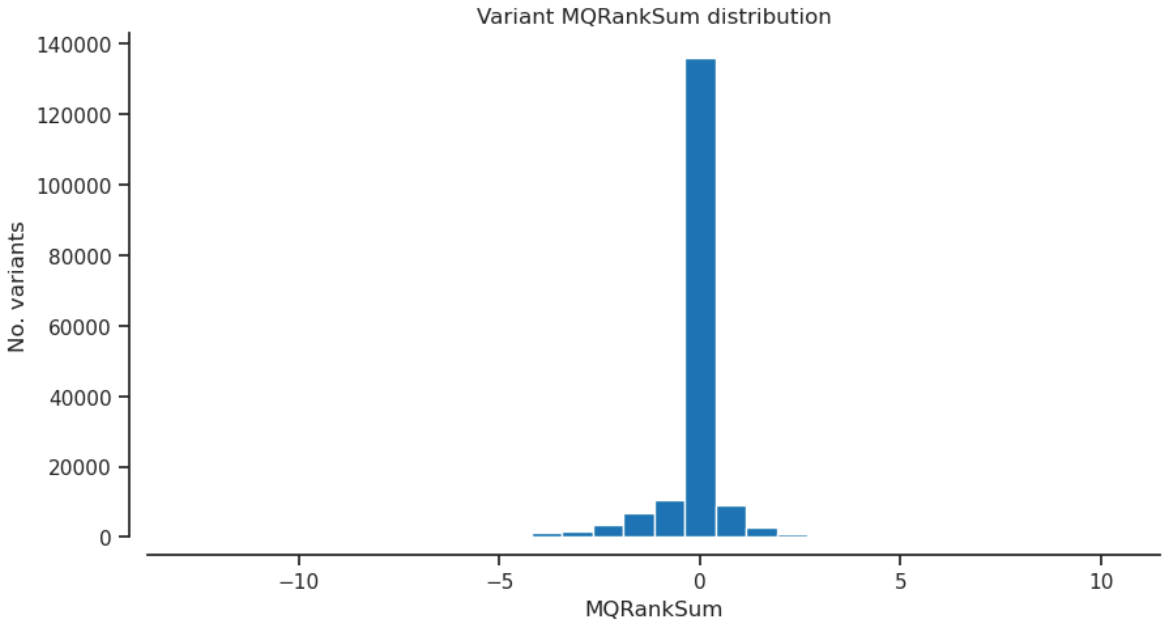


In [30]:
```
plot_hist('SOR','notsnp') # Symmetric Odds Ratio of 2x2 contingency table
```

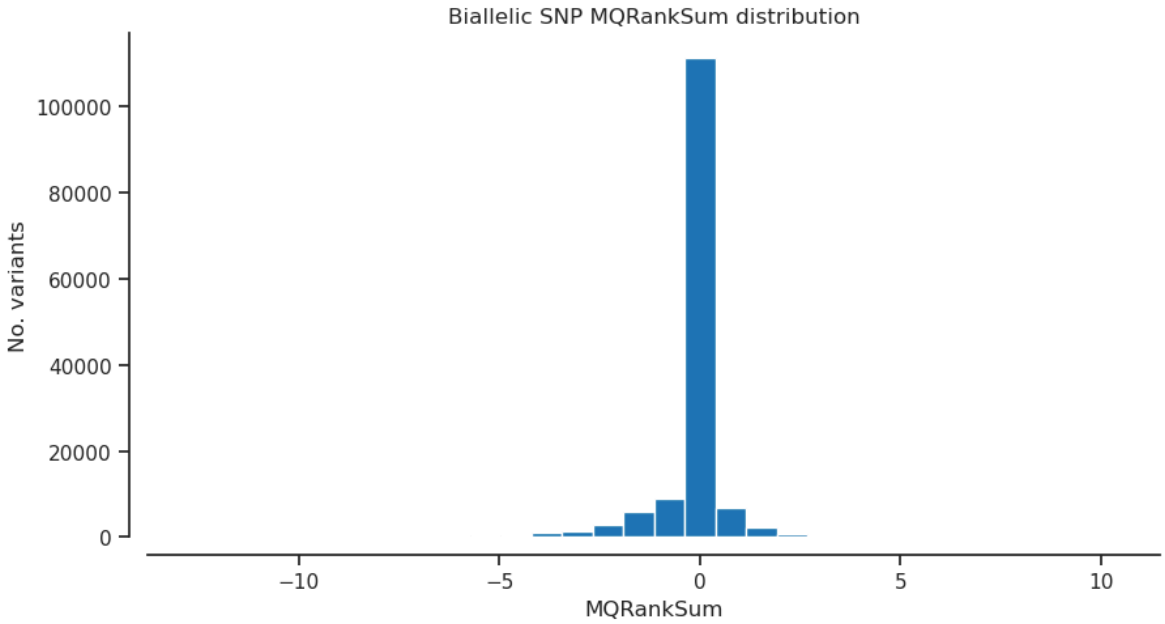


MQRankSum - Z-score From Wilcoxon rank sum test of Alt vs. Ref read mapping qualities

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

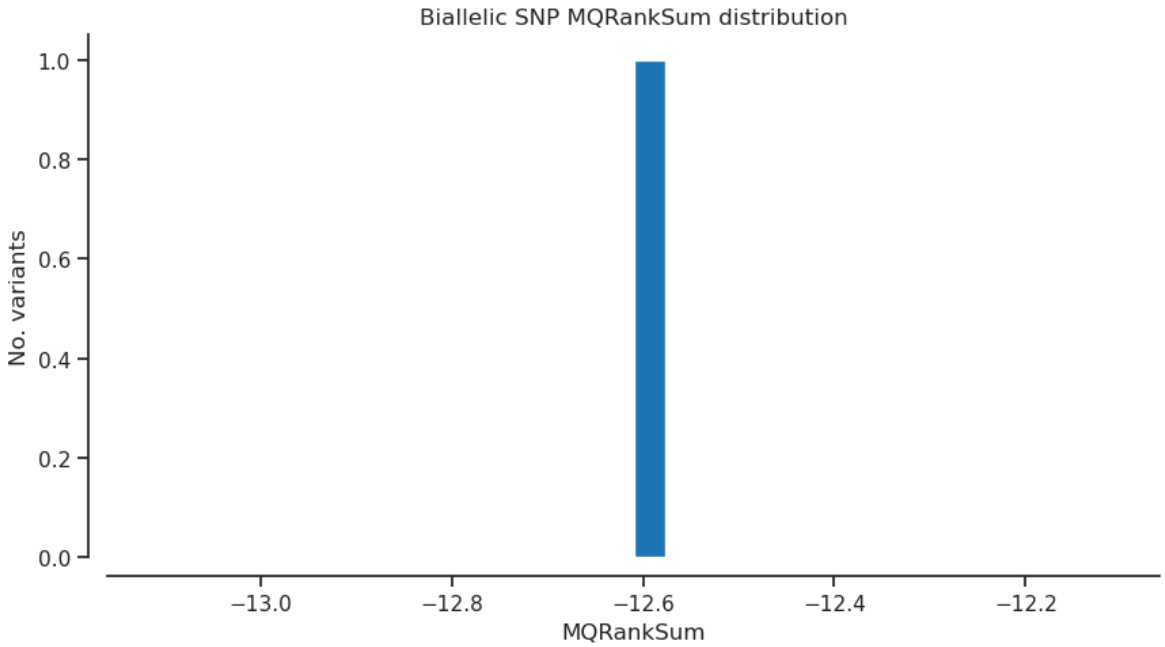


In [32]:
```
plot_hist('MQRankSum','biallelic') # Z-score From Wilcoxon rank sum test
```

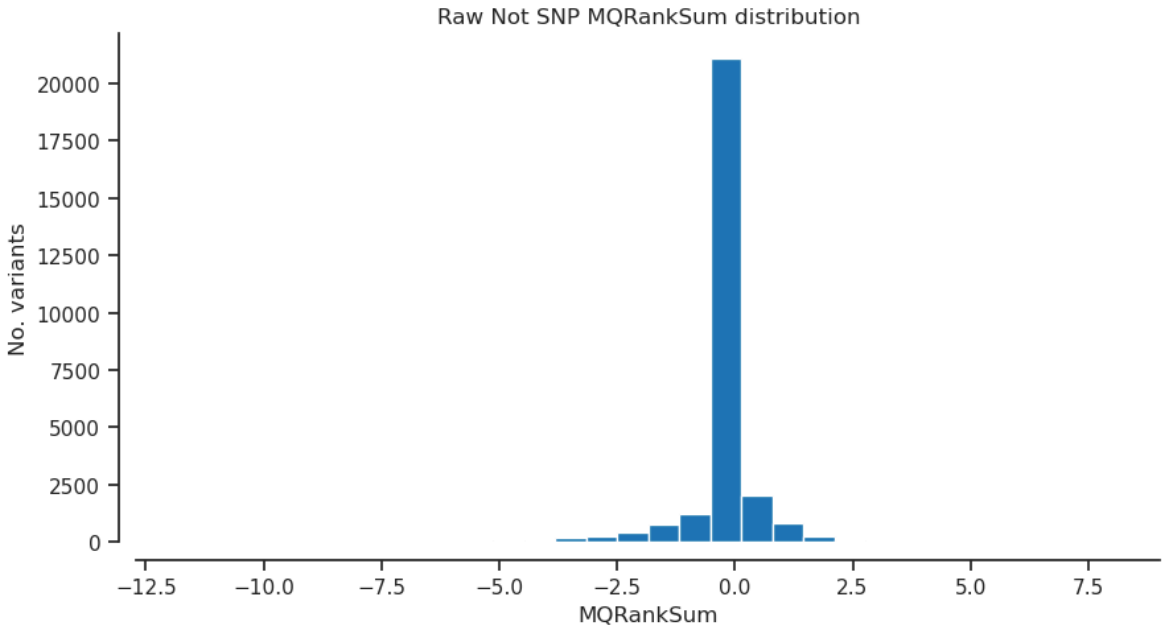


In [33]:
```
filter_expression = '(MQRankSum < -12.5)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

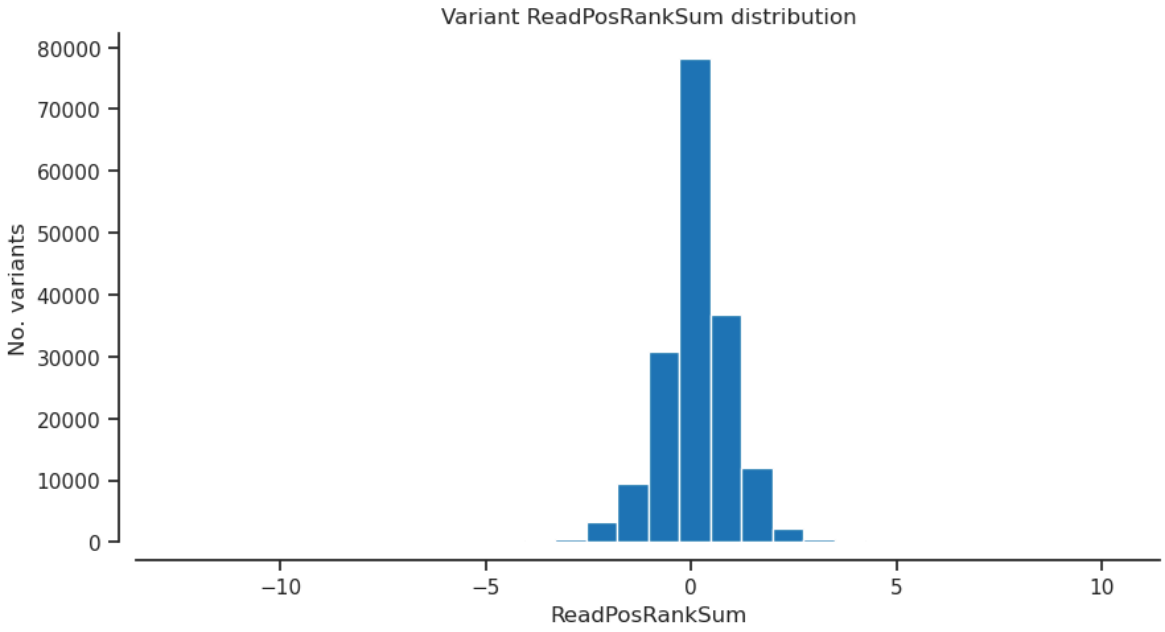


In [35]:
```
plot_hist('MQRankSum','notsnp') # Z-score From Wilcoxon rank sum test of
```

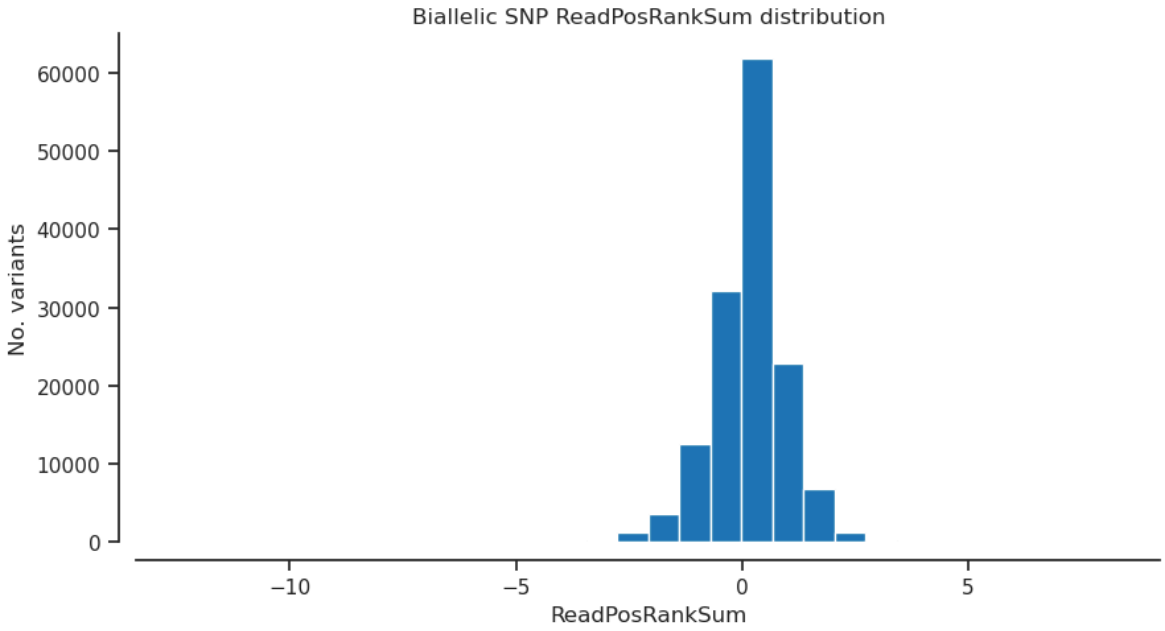


ReadPosRankSum - Z-score from Wilcoxon rank sum test of Alt vs. Ref read position bias

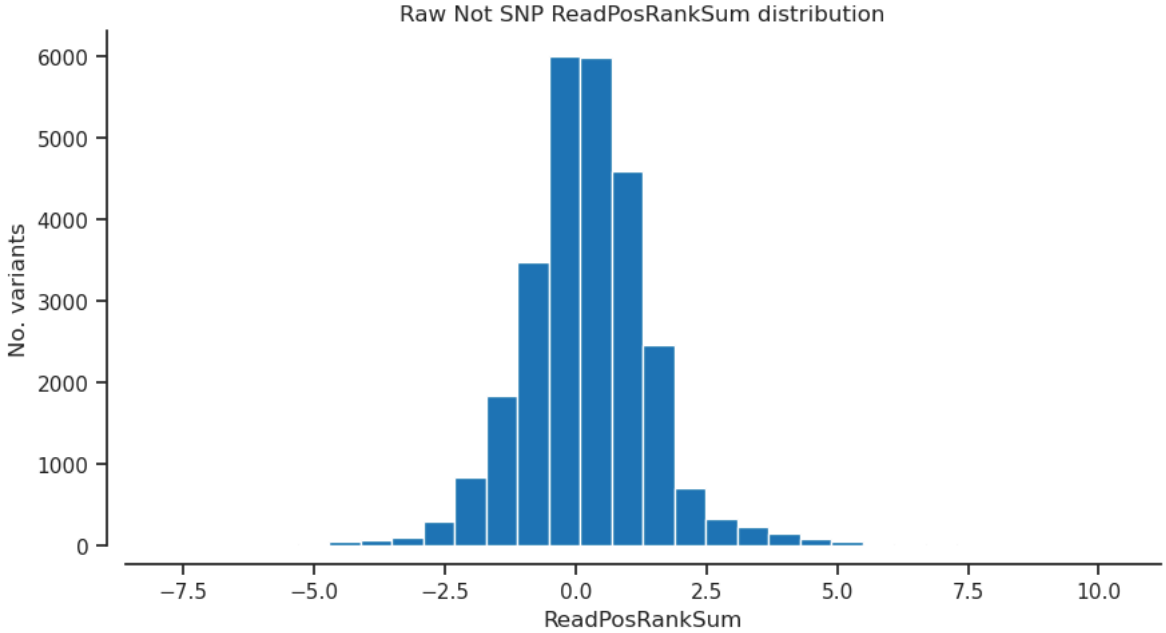
In [36]:
```
plot_hist('ReadPosRankSum','var') # Z-score from Wilcoxon rank sum test o
```



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

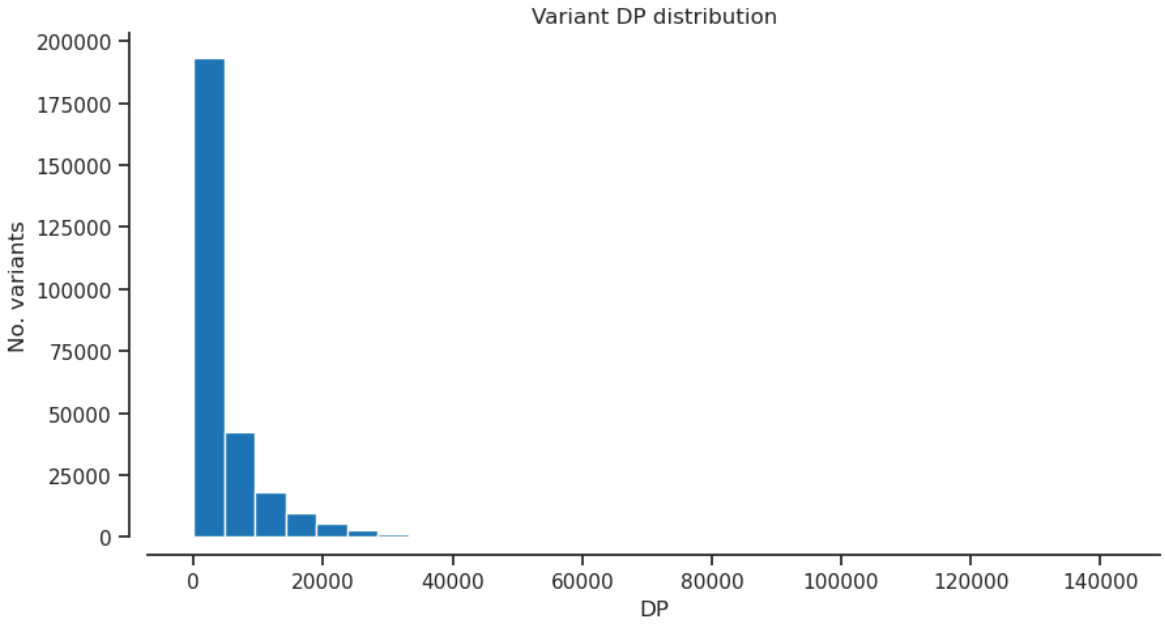


In [38]:
```
plot_hist('ReadPosRankSum','notsnp') # Z-score from Wilcoxon rank sum tes
```

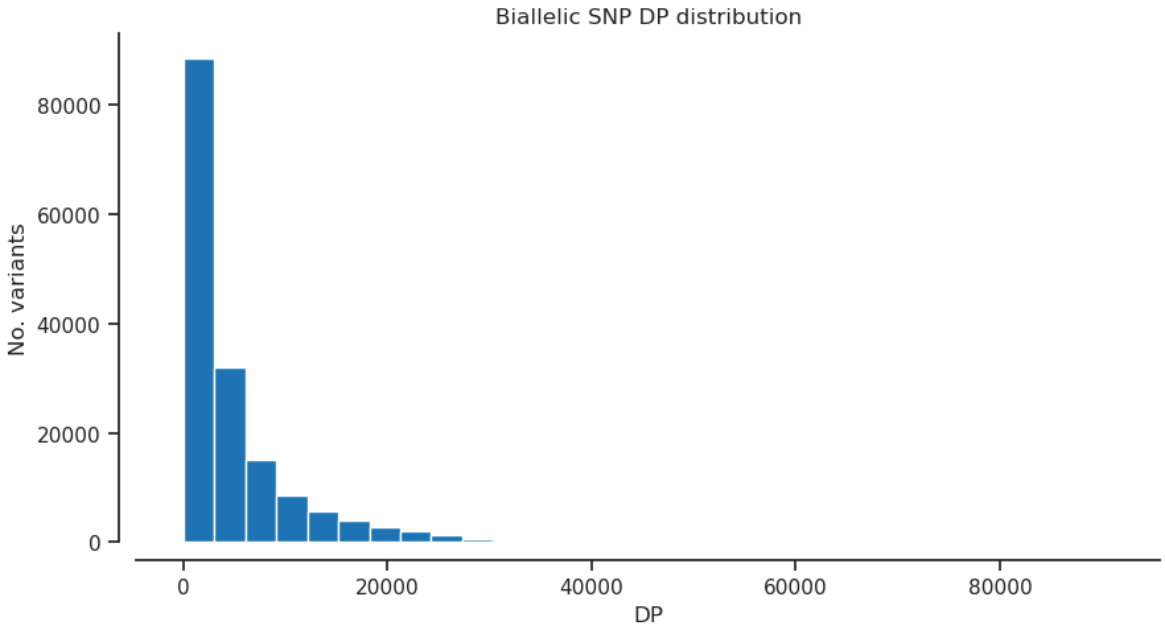


DP - Approximate read depth

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

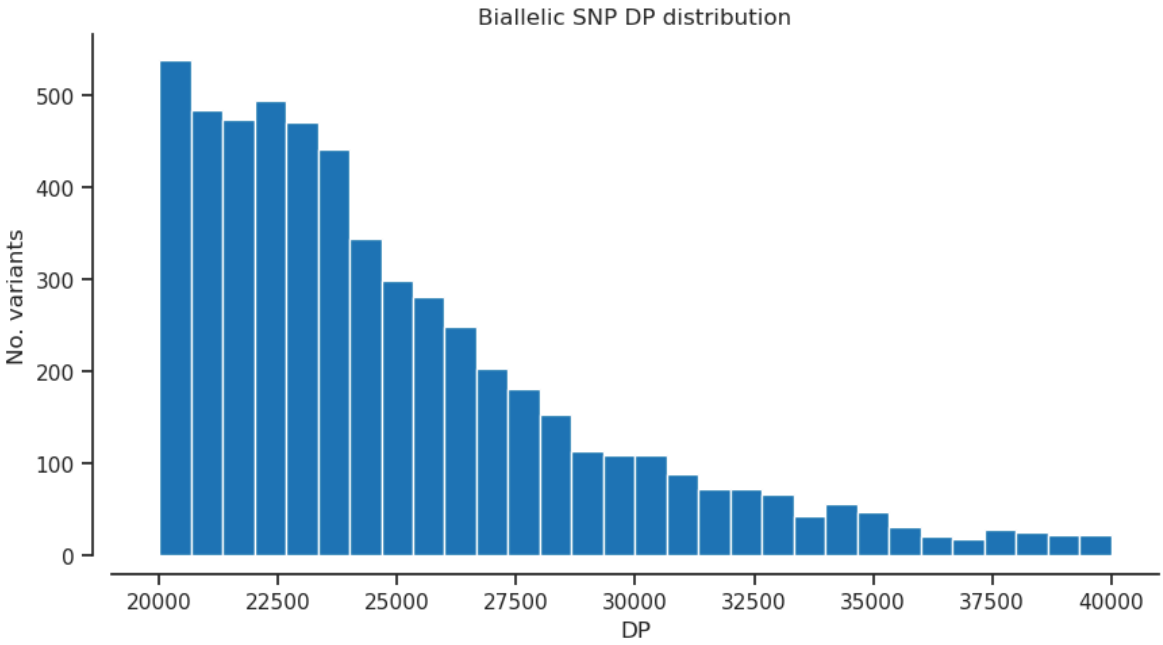


In [40]:
```
plot_hist('DP','biallelic')
```

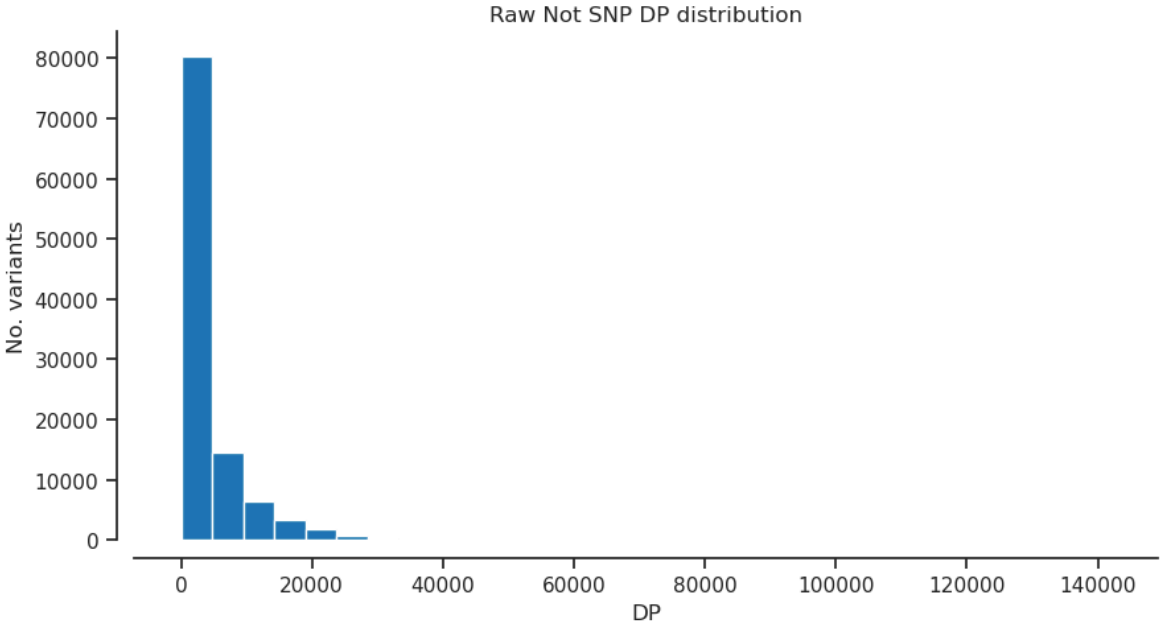


In [41]:
```
filter_expression = '(DP > 20000) & (DP < 40000)'
bi_selection = biallelic_np.query(filter_expression)[:]
```

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

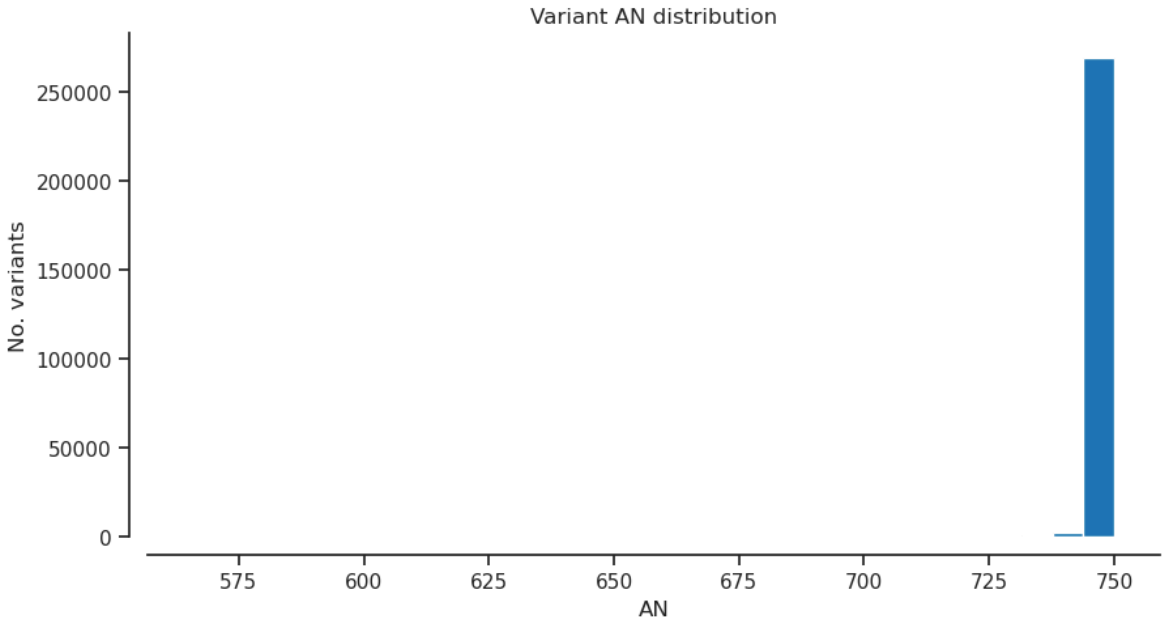


In [43]:
```
plot_hist('DP','notsnp')
```

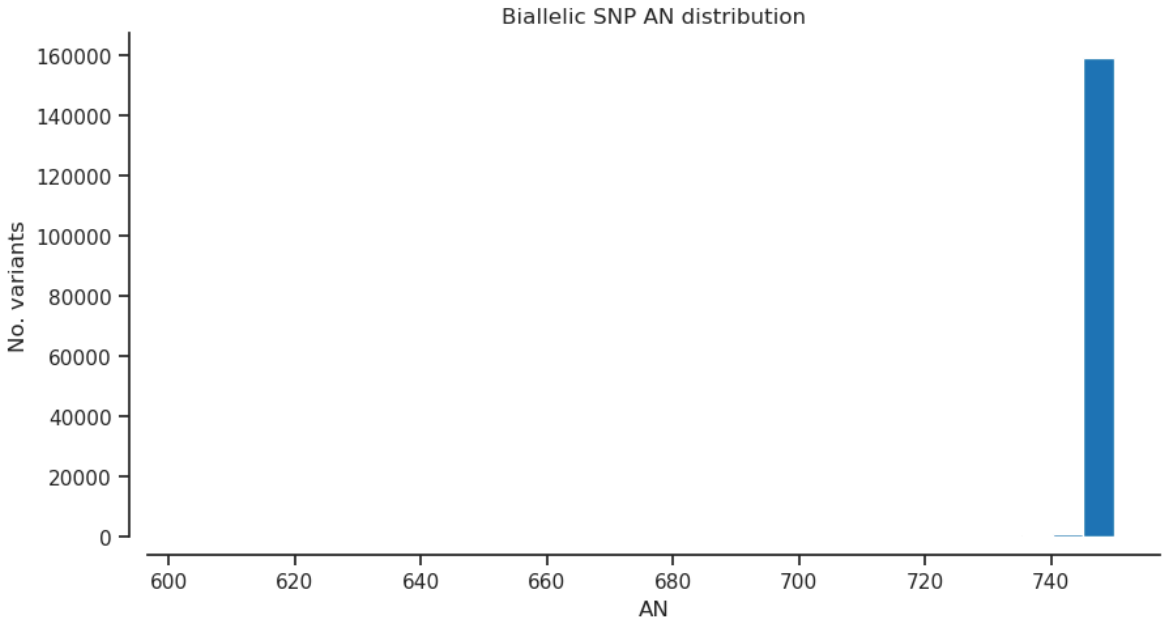


AN - Total number of alleles in called genotypes

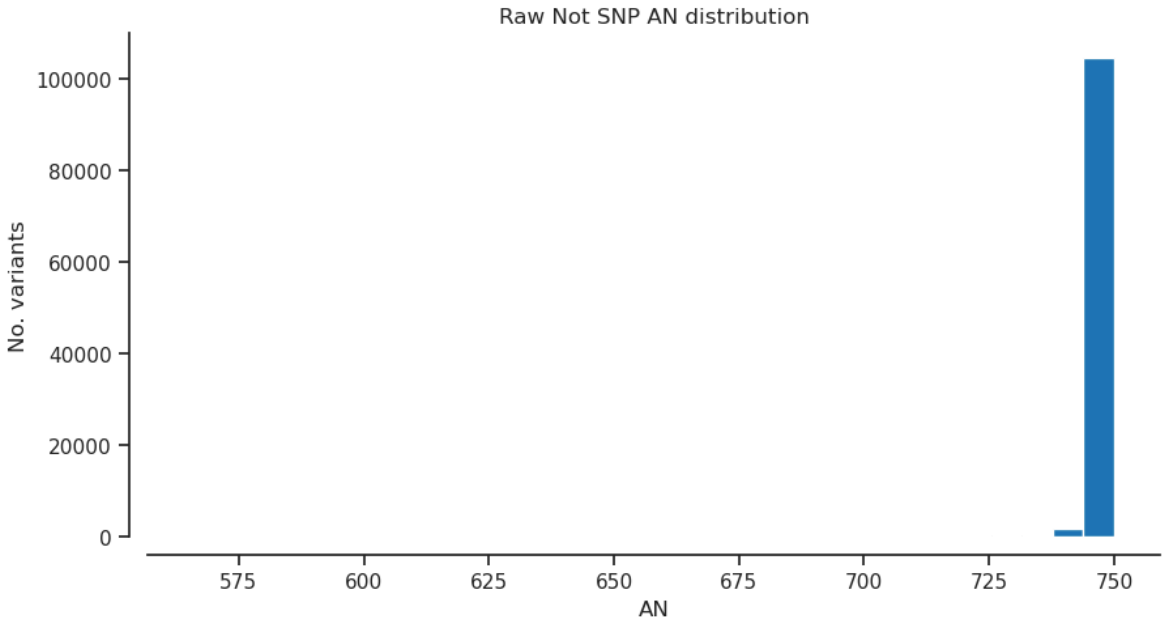
In [44]:
```
plot_hist('AN','var') # Total number of alleles in called genotypes
```



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]: 137114

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=(273617, 375, 2) dtype=int8 chunks=(65536, 64, 2) nbytes=195.7M cbytes=9.9M cratio=19.8 compression=gzip compression_opts=1 values=h5py._hl.dataset.Dataset>

	0	1	2	3	4	...	370	371	372	373	374
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
...						...					
273614	0/0	1/1	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
273615	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
273616	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=(137114, 375, 2) dtype=int8 chunks=(1072, 375, 2) nbytes=98.1M cbytes=10.2M cratio=9.6 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3	4	...	370	371	372	373	374
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
...						...					
137111	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
137112	0/0	1/1	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
137113	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=(137114, 4) dtype=int32 chunks=(17140, 4) nbytes=2.1M cbytes=349.6K cratio=6.1 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3
0	698	52	0	0
1	744	6	0	0
2	743	7	0	0
...		...		
137111	747	3	0	0
137112	727	21	2	0
137113	749	1	0	0

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

Out[52]: <AlleleCountsArray shape=(137114, 4) dtype=int32>

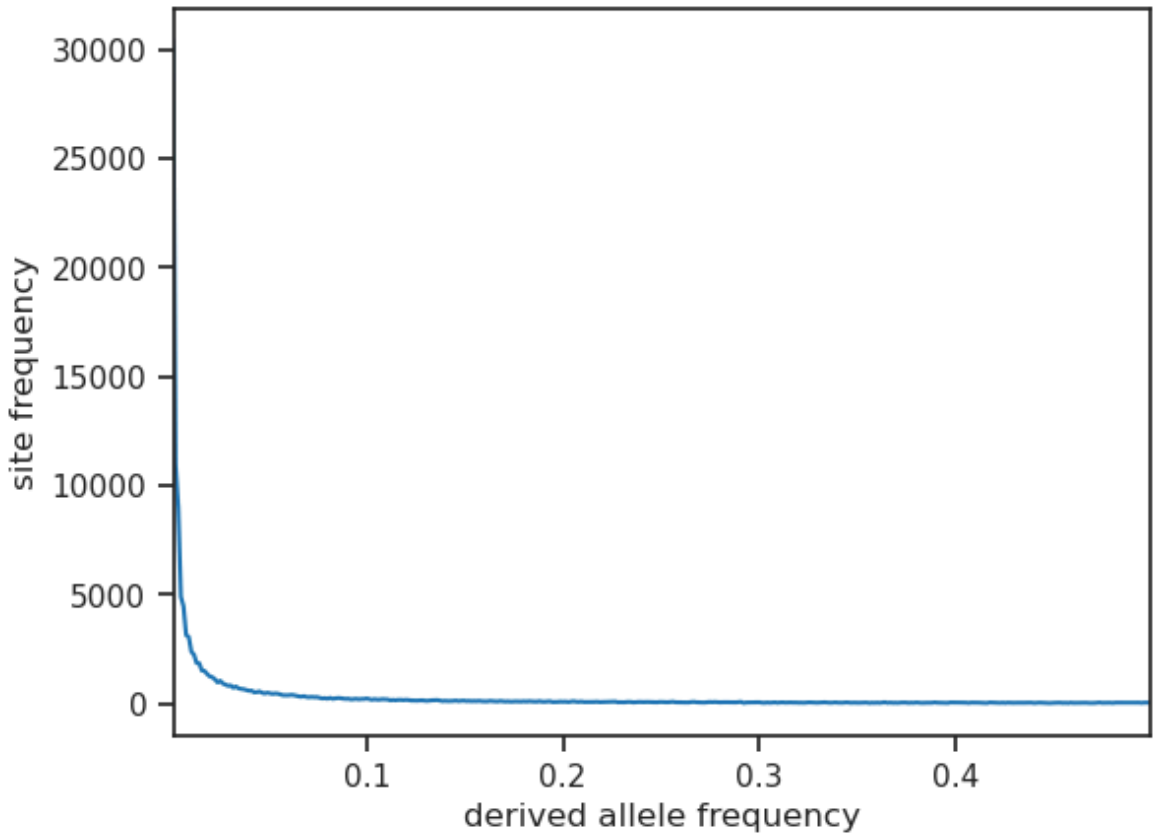
	0	1	2	3
0	698	52	0	0
1	744	6	0	0
2	743	7	0	0
...		...		
137111	747	3	0	0
137112	727	21	2	0
137113	749	1	0	0

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([[698,  52],
       [744,   6],
       [743,   7],
       ...,
       [749,   1],
       [747,   3],
       [749,   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=(137114,) dtype=bool chunks=(137114,)
   nbytes=133.9K cbytes=19.1K cratio=7.0
   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=(132498, 375, 2) dtype=int8 chunks=(1036, 375, 2) nbytes=94.8M cbytes=9.6M cratio=9.9 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1	2	3	4	...	370	371	372	373	374
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
...						...					
132495	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
132496	0/0	0/0	0/0	0/0	0/0	...	0/0	0/0	0/0	0/0	0/0
132497	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]: 132498

In [58]:
```
pc_missing = gt_biallelic.count_missing(axis=0)[:] * 100 / n_variants
pc_het = gt_biallelic.count_het(axis=0)[:] * 100 / n_variants
```

Samples

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

Out[60]:
```
[b'AUT00207-001',
 b'AUT00207-002',
 b'AUT00207-003',
 b'AUT00207-004',
 b'AUT00207-005',
 b'AUT00207-006',
 b'AUT00207-007',
 b'AUT00207-008',
 b'AUT00207-009',
 b'AUT00207-010',
 b'AUT00207-011',
 b'AUT00207-012',
 b'AUT00207-013',
 b'AUT00207-014',
 b'AUT00207-015',
 b'AUT00207-016',
 b'AUT00207-017',
 b'AUT00207-018',
 b'AUT00207-019',
 b'AUT00207-020',
 b'AUT00207-021',
 b'AUT00207-022',
 b'AUT00207-023',
 b'AUT00207-024',
 b'AUT00207-025',
 b'DEU00071-001',
 b'DEU00071-002',
 b'DEU00071-003',
 b'DEU00071-004',
 b'DEU00071-005',
 b'DEU00071-006',
 b'DEU00071-007',
 b'DEU00071-008',
 b'DEU00071-009',
 b'DEU00071-010',
 b'DEU00071-011',
 b'DEU00071-012',
 b'DEU00071-013',
 b'DEU00071-014',
 b'DEU00071-015',
 b'DEU00071-016',
 b'DEU00071-017',
 b'DEU00071-018',
 b'DEU00071-019',
 b'DEU00071-020',
 b'DEU00071-021',
 b'DEU00071-022',
 b'DEU00071-023',
 b'DEU00071-024',
 b'DEU00071-025',
 b'ESP00179-001',
 b'ESP00179-002',
 b'ESP00179-003',
 b'ESP00179-004',
 b'ESP00179-005',
 b'ESP00179-006',
 b'ESP00179-007',
 b'ESP00179-008',
 b'ESP00179-009',
 b'ESP00179-010',
```

```
b'ESP00179-011',
b'ESP00179-012',
b'ESP00179-013',
b'ESP00179-014',
b'ESP00179-015',
b'ESP00179-016',
b'ESP00179-017',
b'ESP00179-018',
b'ESP00179-019',
b'ESP00179-020',
b'ESP00179-021',
b'ESP00179-022',
b'ESP00179-023',
b'ESP00179-024',
b'ESP00179-025',
b'ESP00225-001',
b'ESP00225-002',
b'ESP00225-003',
b'ESP00225-004',
b'ESP00225-005',
b'ESP00225-006',
b'ESP00225-007',
b'ESP00225-008',
b'ESP00225-009',
b'ESP00225-010',
b'ESP00225-011',
b'ESP00225-012',
b'ESP00225-013',
b'ESP00225-014',
b'ESP00225-015',
b'ESP00225-016',
b'ESP00225-017',
b'ESP00225-018',
b'ESP00225-019',
b'ESP00225-020',
b'ESP00225-021',
b'ESP00225-022',
b'ESP00225-023',
b'ESP00225-024',
b'ESP00225-025',
b'ESP00263-001',
b'ESP00263-002',
b'ESP00263-003',
b'ESP00263-004',
b'ESP00263-005',
b'ESP00263-006',
b'ESP00263-007',
b'ESP00263-008',
b'ESP00263-009',
b'ESP00263-010',
b'ESP00263-011',
b'ESP00263-012',
b'ESP00263-013',
b'ESP00263-014',
b'ESP00263-015',
b'ESP00263-016',
b'ESP00263-017',
b'ESP00263-018',
b'ESP00263-019',
b'ESP00263-020',
```

```
b'ESP00263-021',
b'ESP00263-022',
b'ESP00263-023',
b'ESP00263-024',
b'ESP00263-025',
b'FRA00029-001',
b'FRA00029-002',
b'FRA00029-003',
b'FRA00029-004',
b'FRA00029-005',
b'FRA00029-006',
b'FRA00029-007',
b'FRA00029-008',
b'FRA00029-009',
b'FRA00029-010',
b'FRA00029-011',
b'FRA00029-012',
b'FRA00029-013',
b'FRA00029-014',
b'FRA00029-015',
b'FRA00029-016',
b'FRA00029-017',
b'FRA00029-018',
b'FRA00029-019',
b'FRA00029-020',
b'FRA00029-021',
b'FRA00029-022',
b'FRA00029-023',
b'FRA00029-024',
b'FRA00029-025',
b'FRA00042-001',
b'FRA00042-002',
b'FRA00042-003',
b'FRA00042-004',
b'FRA00042-005',
b'FRA00042-006',
b'FRA00042-007',
b'FRA00042-008',
b'FRA00042-009',
b'FRA00042-010',
b'FRA00042-011',
b'FRA00042-012',
b'FRA00042-013',
b'FRA00042-014',
b'FRA00042-015',
b'FRA00042-016',
b'FRA00042-017',
b'FRA00042-018',
b'FRA00042-019',
b'FRA00042-020',
b'FRA00042-021',
b'FRA00042-022',
b'FRA00042-023',
b'FRA00042-024',
b'FRA00042-025',
b'FRA00045-004',
b'FRA00045-039',
b'FRA00045-040',
b'FRA00045-088',
b'FRA00045-122',
```

```
b'FRA00045-173',
b'FRA00045-187',
b'FRA00045-195',
b'FRA00045-207',
b'FRA00045-218',
b'FRA00045-231',
b'FRA00045-278',
b'FRA00045-295',
b'FRA00045-318',
b'FRA00045-323',
b'FRA00045-330',
b'FRA00045-339',
b'FRA00045-352',
b'FRA00045-369',
b'FRA00045-384',
b'FRA00045-393',
b'FRA00045-399',
b'FRA00045-416',
b'FRA00045-442',
b'FRA00045-450',
b'FRA00046-001',
b'FRA00046-002',
b'FRA00046-003',
b'FRA00046-004',
b'FRA00046-005',
b'FRA00046-006',
b'FRA00046-007',
b'FRA00046-008',
b'FRA00046-009',
b'FRA00046-010',
b'FRA00046-011',
b'FRA00046-012',
b'FRA00046-013',
b'FRA00046-014',
b'FRA00046-015',
b'FRA00046-016',
b'FRA00046-017',
b'FRA00046-018',
b'FRA00046-019',
b'FRA00046-020',
b'FRA00046-021',
b'FRA00046-022',
b'FRA00046-023',
b'FRA00046-024',
b'FRA00046-025',
b'ITA00178-001',
b'ITA00178-002',
b'ITA00178-003',
b'ITA00178-004',
b'ITA00178-005',
b'ITA00178-006',
b'ITA00178-007',
b'ITA00178-008',
b'ITA00178-009',
b'ITA00178-010',
b'ITA00178-011',
b'ITA00178-012',
b'ITA00178-013',
b'ITA00178-014',
b'ITA00178-015',
```

```
b'ITA00178-016',
b'ITA00178-017',
b'ITA00178-018',
b'ITA00178-019',
b'ITA00178-020',
b'ITA00178-021',
b'ITA00178-022',
b'ITA00178-023',
b'ITA00178-024',
b'ITA00178-025',
b'NOR00005-001',
b'NOR00005-002',
b'NOR00005-003',
b'NOR00005-006',
b'NOR00005-008',
b'NOR00005-009',
b'NOR00005-010',
b'NOR00005-011',
b'NOR00005-012',
b'NOR00005-013',
b'NOR00005-014',
b'NOR00005-015',
b'NOR00005-018',
b'NOR00005-019',
b'NOR00005-020',
b'NOR00005-021',
b'NOR00005-022',
b'NOR00005-024',
b'NOR00005-025',
b'NOR00005-026',
b'NOR00005-027',
b'NOR00005-028',
b'NOR00005-029',
b'NOR00005-030',
b'NOR00005-031',
b'ROU00077-001',
b'ROU00077-002',
b'ROU00077-003',
b'ROU00077-004',
b'ROU00077-005',
b'ROU00077-006',
b'ROU00077-007',
b'ROU00077-008',
b'ROU00077-009',
b'ROU00077-010',
b'ROU00077-011',
b'ROU00077-012',
b'ROU00077-013',
b'ROU00077-014',
b'ROU00077-015',
b'ROU00077-016',
b'ROU00077-017',
b'ROU00077-018',
b'ROU00077-019',
b'ROU00077-020',
b'ROU00077-021',
b'ROU00077-022',
b'ROU00077-023',
b'ROU00077-024',
b'ROU00077-025',
```

```
b'ROU00467-001',
b'ROU00467-002',
b'ROU00467-003',
b'ROU00467-005',
b'ROU00467-007',
b'ROU00467-008',
b'ROU00467-009',
b'ROU00467-010',
b'ROU00467-011',
b'ROU00467-012',
b'ROU00467-013',
b'ROU00467-014',
b'ROU00467-015',
b'ROU00467-016',
b'ROU00467-017',
b'ROU00467-018',
b'ROU00467-019',
b'ROU00467-020',
b'ROU00467-021',
b'ROU00467-022',
b'ROU00467-023',
b'ROU00467-024',
b'ROU00467-025',
b'ROU00467-027',
b'ROU00467-028',
b'SVN00047-002',
b'SVN00047-023',
b'SVN00047-032',
b'SVN00047-050',
b'SVN00047-068',
b'SVN00047-074',
b'SVN00047-080',
b'SVN00047-127',
b'SVN00047-155',
b'SVN00047-169',
b'SVN00047-184',
b'SVN00047-200',
b'SVN00047-222',
b'SVN00047-235',
b'SVN00047-245',
b'SVN00047-316',
b'SVN00047-341',
b'SVN00047-362',
b'SVN00047-372',
b'SVN00047-381',
b'SVN00047-393',
b'SVN00047-464',
b'SVN00047-476',
b'SVN00047-491',
b'SVN00047-498',
b'TUR00264-001',
b'TUR00264-002',
b'TUR00264-003',
b'TUR00264-004',
b'TUR00264-005',
b'TUR00264-006',
b'TUR00264-007',
b'TUR00264-008',
b'TUR00264-009',
b'TUR00264-010',
```

```
 b'TUR00264-011',
 b'TUR00264-012',
 b'TUR00264-013',
 b'TUR00264-014',
 b'TUR00264-015',
 b'TUR00264-016',
 b'TUR00264-017',
 b'TUR00264-018',
 b'TUR00264-019',
 b'TUR00264-020',
 b'TUR00264-021',
 b'TUR00264-022',
 b'TUR00264-023',
 b'TUR00264-024',
 b'TUR00264-025']
```

In [61]:
```
samples_fn = '~/scratch/data/Fsylvatica/Fagus_sylvatica_sample_list_sciki
samples = pandas.read_csv(samples_fn, sep='\t')
samples
```

Out[61]:

	ID	Population
0	AUT00207-001	AUT00207
1	AUT00207-002	AUT00207
2	AUT00207-003	AUT00207
3	AUT00207-004	AUT00207
4	AUT00207-005	AUT00207
...	...	...
370	TUR00264-021	TUR00264
371	TUR00264-022	TUR00264
372	TUR00264-023	TUR00264
373	TUR00264-024	TUR00264
374	TUR00264-025	TUR00264

375 rows × 2 columns

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

Out[62]:
```
Population
AUT00207    25
DEU00071    25
ESP00179    25
ESP00225    25
ESP00263    25
FRA00029    25
FRA00042    25
FRA00045    25
FRA00046    25
ITA00178    25
NOR00005    25
ROU00077    25
ROU00467    25
SVN00047    25
TUR00264    25
Name: count, dtype: int64
```

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

Out[63]:
```
array(['AUT00207', 'DEU00071', 'ESP00179', 'ESP00225', 'ESP00263',
       'FRA00029', 'FRA00042', 'FRA00045', 'FRA00046', 'ITA00178',
       'NOR00005', 'ROU00077', 'ROU00467', 'SVN00047', 'TUR00264'],
      dtype=object)
```

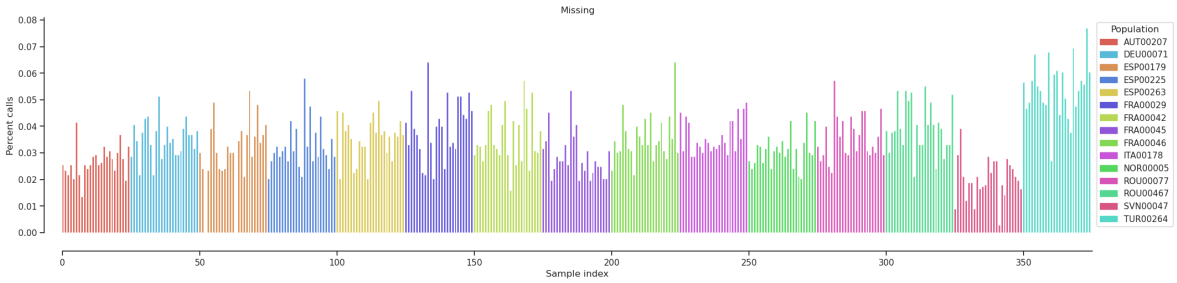
Gt frequency function

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", 15)
    pop2color = {'AUT00207': palette[0],
                 'DEU00071': palette[8],
                 'ESP00179': palette[1],
                 'ESP00225': palette[9],
                 'ESP00263': palette[2],
                 'FRA00029': palette[10],
                 'FRA00042': palette[3],
                 'FRA00045': palette[11],
                 'FRA00046': palette[4],
                 'ITA00178': palette[12],
                 'NOR00005': palette[5],
                 'ROU00077': palette[13],
                 'ROU00467': palette[6],
                 'SVN00047': palette[14],
                 'TUR00264': palette[7]}
    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])]
    ax.legend(handles=handles, labels=['AUT00207', 'DEU00071', 'ESP00179'
       'FRA00029', 'FRA00042', 'FRA00045', 'FRA00046', 'ITA00178',
       'NOR00005', 'ROU00077', 'ROU00467', 'SVN00047', 'TUR00264'], title
              bbox_to_anchor=(1, 1), loc='upper left')
```

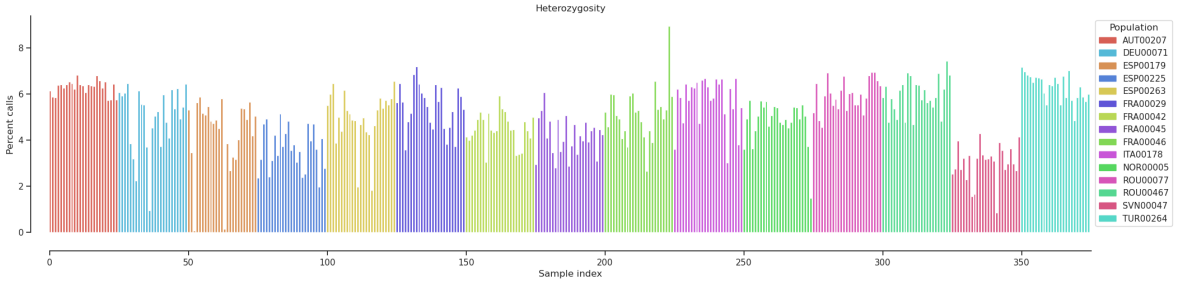
Plot missing

In [65]:
```
plot_genotype_frequency(pc_missing, 'Missing')
```



Plot heterozygosity

In [66]:
```
plot_genotype_frequency(pc_het, 'Heterozygosity')
```



PCA

In [67]:
```
palette = sns.color_palette("hls",15)
pop_colours = {
                'AUT00207': palette[0],
                 'DEU00071': palette[8],
                 'ESP00179': palette[1],
                 'ESP00225': palette[9],
                 'ESP00263': palette[2],
                 'FRA00029': palette[10],
                 'FRA00042': palette[3],
                 'FRA00045': palette[11],
```

```
                'FRA00046': palette[4],
                'ITA00178': palette[12],
                'NOR00005': palette[5],
                'ROU00077': palette[13],
                'ROU00467': palette[6],
                'SVN00047': palette[14],
                'TUR00264': palette[7]
}
```

In [68]:
```
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 [69]:
```
ac2 = gt_biallelic.count_alleles()
ac2
```

Out[69]:
<AlleleCountsChunkedArray shape=(132498, 2) dtype=int32 chunks=(33125, 2) nbytes=1.0M cbytes=273.4K cratio=3.8 compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shuffle': 1, 'blocksize': 0} values=zarr.core.Array>

	0	1
0	698	52
1	744	6
2	743	7
...	...	
132495	749	1
132496	747	3
132497	749	1

In [70]:
```
flt = (ac2[:, :2].min(axis=1) > 1)
gf = gt_biallelic.compress(flt, axis=0)
```

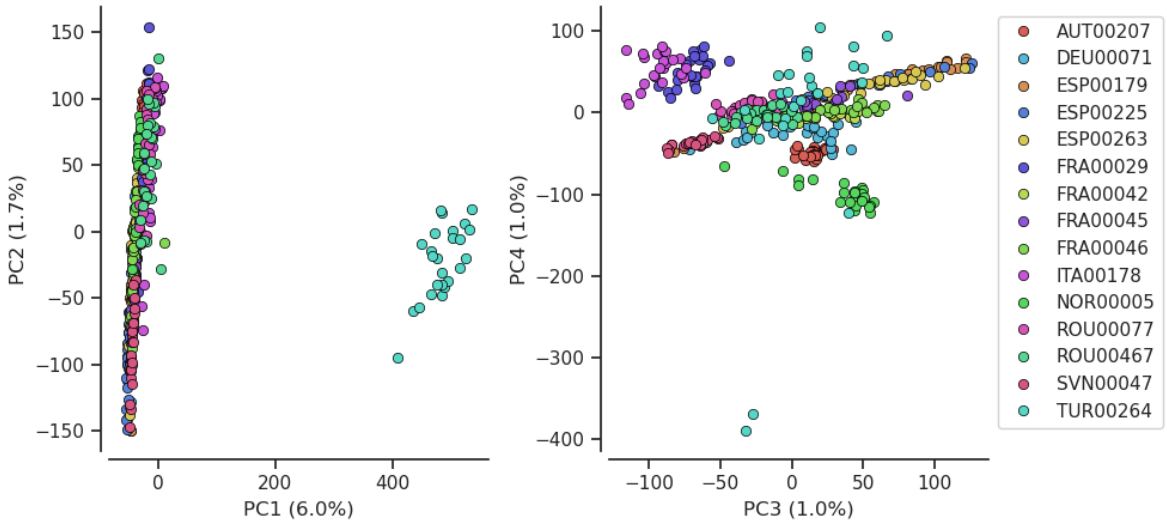
```
gn = gf.to_n_alt()
gn
```

Out[70]:
```
<ChunkedArrayWrapper shape=(102130, 375) dtype=int8 chunks=(1596, 375)
   nbytes=36.5M cbytes=6.2M cratio=5.9
   compression=blosc compression_opts={'cname': 'lz4', 'clevel': 5, 'shu
ffle': 1, 'blocksize': 0}
   values=zarr.core.Array>
```

In [71]:
```
coords1, model1 = allel.pca(gn, n_components=10, scaler='patterson')
```

In [72]:
```
fig_pca(coords1, model1, 'Figure 1. Conventional PCA.')
```

Figure 1. Conventional PCA.



In [73]:
```
outliers = coords1[:,3]<-200
samples[outliers]
```

Out[73]:

	ID	Population
373	TUR00264-024	TUR00264
374	TUR00264-025	TUR00264