

---

# **snps Documentation**

***Release 2.5.0.post58+g125f809***

**Andrew Riha**

**Oct 10, 2023**



# CONTENTS

<b>1</b>	<b>snps</b>	<b>3</b>
1.1	Features . . . . .	3
1.2	Supported Genotype Files . . . . .	4
1.3	Dependencies . . . . .	4
1.4	Installation . . . . .	4
1.5	Examples . . . . .	5
1.6	Documentation . . . . .	8
1.7	Acknowledgements . . . . .	8
<b>2</b>	<b>Output Files</b>	<b>9</b>
2.1	Save SNPs . . . . .	9
<b>3</b>	<b>Installation</b>	<b>11</b>
3.1	Installation and Usage on a Raspberry Pi . . . . .	11
<b>4</b>	<b>snps Banner</b>	<b>13</b>
4.1	SNPs . . . . .	13
<b>5</b>	<b>Changelog</b>	<b>17</b>
<b>6</b>	<b>Contributing</b>	<b>19</b>
6.1	Bug reports . . . . .	19
6.2	Documentation improvements . . . . .	19
6.3	Feature requests and feedback . . . . .	19
6.4	Development . . . . .	20
6.5	Documentation . . . . .	21
<b>7</b>	<b>Contributors</b>	<b>23</b>
7.1	Core Developers . . . . .	23
7.2	Other Contributors . . . . .	23
<b>8</b>	<b>Code Documentation</b>	<b>25</b>
8.1	SNPs . . . . .	25
8.2	snps.ensembl . . . . .	35
8.3	snps.io . . . . .	36
8.4	snps.resources . . . . .	41
8.5	snps.utils . . . . .	47
<b>9</b>	<b>Indices and tables</b>	<b>49</b>
	<b>Python Module Index</b>	<b>51</b>



*tools for reading, writing, merging, and remapping SNPs*

GATCACAGGTCTATCAC	CCTATTAACCACTCAC	GGGAGCTCTCCATGCAT	TTGGTATTTTCGTCTGG
GGGGTATGCACGCGATA	GCATTGCGAGACGCTG	GAGCCGGAGCACCCTAT	GTGCGAGTATCTGTCTT
TGA	TTC	CTG	CCT
ATTATTTATCGCACCTA	CGT	TCA	ACTTACTAAAGTGTGTT
AATTAATTAATGCTTGT	AGG	ACA	TGTCTGCACAGCCACTT
TCC	ACA	CAG	TCA
TAACAAAAAATTTCCAC	CAA	ACC	CCTCCCCCGCTTCTGGC
CACAGCACTTAAACACA	TCT	CTG	AACCCCAAAAACAAAGA



tools for reading, writing, merging, and remapping SNPs

*snps strives to be an easy-to-use and accessible open-source library for working with genotype data*

## 1.1 Features

### 1.1.1 Input / Output

- Read raw data (genotype) files from a variety of direct-to-consumer (DTC) DNA testing sources with a [SNPs](#) object
- Read and write VCF files (e.g., convert [23andMe](#) to VCF)
- Merge raw data files from different DNA tests, identifying discrepant SNPs in the process
- Read data in a variety of formats (e.g., files, bytes, compressed with *gzip* or *zip*)
- Handle several variations of file types, validated via [openSNP parsing analysis](#)

### 1.1.2 Build / Assembly Detection and Remapping

- Detect the build / assembly of SNPs (supports builds 36, 37, and 38)
- Remap SNPs between builds / assemblies

### 1.1.3 Data Cleaning

- Perform quality control (QC) / filter low quality SNPs based on [chip clusters](#)
- Fix several common issues when loading SNPs
- Sort SNPs based on chromosome and position
- Deduplicate RSIDs
- Deduplicate alleles in the non-PAR regions of the X and Y chromosomes for males
- Deduplicate alleles on MT
- Assign PAR SNPs to the X or Y chromosome

### 1.1.4 Analysis

- Derive sex from SNPs
- Detect deduced genotype / chip array and chip version based on [chip clusters](#)
- Predict ancestry from SNPs (when installed with [ezancestry](#))

## 1.2 Supported Genotype Files

snps supports [VCF](#) files and genotype files from the following DNA testing sources:

- [23andMe](#)
- [Ancestry](#)
- [CircleDNA](#)
- [Código 46](#)
- [DNA.Land](#)
- [Family Tree DNA](#)
- [Genes for Good](#)
- [LivingDNA](#)
- [Mapmygenome](#)
- [MyHeritage](#)
- [Sano Genetics](#)
- [tellmeGen](#)

Additionally, snps can read a variety of “generic” CSV and TSV files.

## 1.3 Dependencies

snps requires [Python 3.7.1+](#) and the following Python packages:

- [numpy](#)
- [pandas](#)
- [atomicwrites](#)

## 1.4 Installation

snps is [available](#) on the [Python Package Index](#). Install snps (and its required Python dependencies) via `pip`:

```
$ pip install snps
```

For [ancestry prediction](#) capability, snps can be installed with [ezancestry](#):

```
$ pip install snps[ezancestry]
```



## 1.5 Examples

### 1.5.1 Download Example Data

First, let's setup logging to get some helpful output:

```
>>> import logging, sys
>>> logger = logging.getLogger()
>>> logger.setLevel(logging.INFO)
>>> logger.addHandler(logging.StreamHandler(sys.stdout))
```

Now we're ready to download some example data from `openSNP`:

```
>>> from snps.resources import Resources
>>> r = Resources()
>>> paths = r.download_example_datasets()
Downloading resources/662.23andme.340.txt.gz
Downloading resources/662.ftdna-illumina.341.csv.gz
```

### 1.5.2 Load Raw Data

Load a `23andMe` raw data file:

```
>>> from snps import SNPs
>>> s = SNPs("resources/662.23andme.340.txt.gz")
>>> s.source
'23andMe'
>>> s.count
991786
```

The `SNPs` class accepts a path to a file or a bytes object. A `Reader` class attempts to infer the data source and load the SNPs. The loaded SNPs are `normalized` and available via a `pandas.DataFrame`:

```
>>> df = s.snps
>>> df.columns.values
array(['chrom', 'pos', 'genotype'], dtype=object)
>>> df.index.name
'rsid'
>>> df.chrom.dtype.name
'object'
>>> df.pos.dtype.name
'uint32'
>>> df.genotype.dtype.name
'object'
>>> len(df)
991786
```

`snps` also attempts to detect the build / assembly of the data:

```
>>> s.build
37
>>> s.build_detected
```

(continues on next page)

(continued from previous page)

```
True
>>> s.assembly
'GRCh37'
```

### 1.5.3 Merge Raw Data Files

The dataset consists of raw data files from two different DNA testing sources - let's combine these files. Specifically, we'll update the SNPs object with SNPs from a [Family Tree DNA](#) file.

```
>>> merge_results = s.merge([SNPs("resources/662.ftdna-illumina.341.csv.gz")])
Merging SNPs('662.ftdna-illumina.341.csv.gz')
SNPs('662.ftdna-illumina.341.csv.gz') has Build 36; remapping to Build 37
Downloading resources/NCBI36_GRCh37.tar.gz
27 SNP positions were discrepant; keeping original positions
151 SNP genotypes were discrepant; marking those as null
>>> s.source
'23andMe, FTDNA'
>>> s.count
1006960
>>> s.build
37
>>> s.build_detected
True
```

If the SNPs being merged have a build that differs from the destination build, the SNPs to merge will be remapped automatically. After this example merge, the build is still detected, since the build was detected for all SNPs objects that were merged.

As the data gets added, it's compared to the existing data, and SNP position and genotype discrepancies are identified. (The discrepancy thresholds can be tuned via parameters.) These discrepant SNPs are available for inspection after the merge via properties of the SNPs object.

```
>>> len(s.discrepant_merge_genotypes)
151
```

Additionally, any non-called / null genotypes will be updated during the merge, if the file being merged has a called genotype for the SNP.

Moreover, `merge` takes a `chrom` parameter - this enables merging of only SNPs associated with the specified chromosome (e.g., "Y" or "MT").

Finally, `merge` returns a list of dict, where each dict has information corresponding to the results of each merge (e.g., SNPs in common).

```
>>> sorted(list(merge_results[0].keys()))
['common_rsids', 'discrepant_genotype_rsids', 'discrepant_position_rsids', 'merged']
>>> merge_results[0]["merged"]
True
>>> len(merge_results[0]["common_rsids"])
692918
```

### 1.5.4 Remap SNPs

Now, let's remap the merged SNPs to change the assembly / build:

```
>>> s.snps.loc["rs3094315"].pos
752566
>>> chromosomes_remapped, chromosomes_not_remapped = s.remap(38)
Downloading resources/GRCh37_GRCh38.tar.gz
>>> s.build
38
>>> s.assembly
'GRCh38'
>>> s.snps.loc["rs3094315"].pos
817186
```

SNPs can be remapped between Build 36 (NCBI36), Build 37 (GRCh37), and Build 38 (GRCh38).

### 1.5.5 Save SNPs

Ok, so far we've merged the SNPs from two files (ensuring the same build in the process and identifying discrepancies along the way). Then, we remapped the SNPs to Build 38. Now, let's save the merged and remapped dataset consisting of 1M+ SNPs to a tab-separated values (TSV) file:

```
>>> saved_snps = s.to_tsv("out.txt")
Saving output/out.txt
>>> print(saved_snps)
output/out.txt
```

Moreover, let's get the reference sequences for this assembly and save the SNPs as a VCF file:

```
>>> saved_snps = s.to_vcf("out.vcf")
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.1.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.2.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.3.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.4.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.5.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.6.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.7.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.8.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.9.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.10.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.11.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.12.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.13.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.14.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.15.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.16.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.17.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.18.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.19.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.20.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.21.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.22.fa.gz
```

(continues on next page)

(continued from previous page)

```
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.X.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.Y.fa.gz
Downloading resources/fasta/GRCh38/Homo_sapiens.GRCh38.dna.chromosome.MT.fa.gz
Saving output/out.vcf
1 SNP positions were found to be discrepant when saving VCF
```

When saving a VCF, if any SNPs have positions outside of the reference sequence, they are marked as discrepant and are available via a property of the SNPs object.

All [output files](#) are saved to the output directory.

## 1.6 Documentation

Documentation is available [here](#).

## 1.7 Acknowledgements

Thanks to Mike Agostino, Padma Reddy, Kevin Arvai, [openSNP](#), [Open Humans](#), and [Sano Genetics](#).

## OUTPUT FILES

The various output files produced by `snps` are detailed below. Output files are saved in the output directory, which is defined at the instantiation of a `SNPs` object.

### 2.1 Save SNPs

SNPs can be saved with `SNPs.save`. By default, one tab-separated `.txt` or `.vcf` file (`vcf=True`) is output when SNPs are saved. If comma is specified as the separator (`sep=","`), the default extension is `.csv`.

The content of non-VCF files (after comment lines, which start with `#`) is as follows:

Column	Description
<code>rsid</code>	SNP ID
<code>chromosome</code>	Chromosome of SNP
<code>position</code>	Position of SNP
<code>genotype</code>	Genotype of SNP

When `filename` is not specified, default filenames are used as described below.

#### 2.1.1 `SNPs.save`

`<source>_<assembly>.txt / <source>_<assembly>.csv`

Where `source` is the detected source(s) of SNPs data and `assembly` is the assembly of the SNPs being saved.



## INSTALLATION

`snps` is available on the [Python Package Index](#). Install `snps` (and its required Python dependencies) via `pip`:

```
$ pip install snps
```

### 3.1 Installation and Usage on a Raspberry Pi

The instructions below provide the steps to install `snps` on a [Raspberry Pi](#) (tested with “[Raspberry Pi OS \(32-bit\) Lite](#)”, release date 2020-08-20). For more details about Python on the Raspberry Pi, see [here](#).

---

**Note:** Text after a prompt (e.g., `$`) is the command to type at the command line. The instructions assume a fresh install of Raspberry Pi OS and that after logging in as the `pi` user, the current working directory is `/home/pi`.

---

1. Install `pip` for Python 3:

```
pi@raspberrypi:~ $ sudo apt install python3-pip
```

Press “y” followed by “enter” to continue. This enables us to install packages from the Python Package Index.

2. Install the `venv` module:

```
pi@raspberrypi:~ $ sudo apt install python3-venv
```

Press “y” followed by “enter” to continue. This enables us to create a [virtual environment](#) to isolate the `snps` installation from other system Python packages.

3. [Install ATLAS](#):

```
pi@raspberrypi:~ $ sudo apt install libatlas-base-dev
```

Press “y” followed by “enter” to continue. This is required for [NumPy](#), a dependency of `snps`.

4. Create a directory for `snps` and change working directory:

```
pi@raspberrypi:~ $ mkdir snps
pi@raspberrypi:~ $ cd snps
```

5. Create a virtual environment for `snps`:

```
pi@raspberrypi:~/snps $ python3 -m venv .venv
```

The virtual environment is located at `/home/pi/snps/.venv`.

6. Activate the virtual environment:

```
pi@raspberrypi:~/snps $ source .venv/bin/activate
```

Now when you invoke Python or pip, the virtual environment's version will be used (as indicated by the `(.venv)` before the prompt). This can be verified as follows:

```
(.venv) pi@raspberrypi:~/snps $ which python
/home/pi/snps/.venv/bin/python
```

7. Install snps:

```
(.venv) pi@raspberrypi:~/snps $ pip install snps
```

8. Start Python:

```
(.venv) pi@raspberrypi:~/snps $ python
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

9. Use snps; examples shown in the README should now work.

10. At completion of usage, the virtual environment can be deactivated:

```
(.venv) pi@raspberrypi:~/snps $ deactivate
pi@raspberrypi:~/snps $
```



## SNPS BANNER

The **snps** banner is composed of nucleotides from [GRCh38 mitochondrial DNA](#). SNPs are represented by lighter colored nucleotides.

The SVG file was modified from a version created with [macSVG](#).

The PNG file was exported from [macSVG](#).

The color scheme was generated by [ColorBrewer](#).

### 4.1 SNPs

The SNPs highlighted in the banner were identified with the [UCSC Genome Browser](#). The dbSNP accessions for the SNPs are as follows:

- rs3883917
- rs370271105
- rs3087742
- rs369034419
- rs147830800
- rs369070397
- rs144402189
- rs139684161
- rs375589100
- rs370482130
- rs62581312
- rs117135796
- rs370716192
- rs41473347
- rs113913230
- rs368807878
- rs371543232
- rs2857291

- rs72619362
- rs372099630
- rs3135032
- rs369669319
- rs368534078
- rs372889209
- rs372439069
- rs41531144
- rs372946833
- rs41323649
- rs368463610
- rs3937037
- rs375896687
- rs145412228
- rs376013487
- rs372529808
- rs41334645
- rs372003323
- rs41400048
- rs2853515
- rs201801609
- rs41528348
- rs3927813
- rs66492218
- rs371975106
- rs373732637
- rs117394573
- rs3883865
- rs28678375

### 4.1.1 References

- Sherry ST, Ward MH, Kholodov M, Baker J, Phan L, Smigielski EM, Sirotkin K. dbSNP: the NCBI database of genetic variation. *Nucleic Acids Res.* 2001 Jan 1; 29(1):308-11.
- Database of Single Nucleotide Polymorphisms (dbSNP). Bethesda (MD): National Center for Biotechnology Information, National Library of Medicine. dbSNP accession: <listed above> (dbSNP Build ID: 142). Available from: <http://www.ncbi.nlm.nih.gov/SNP/>



## CHANGELOG

The changelog is maintained here: <https://github.com/apriha/snps/releases>



## CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

### 6.1 Bug reports

When [reporting a bug](#) please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

### 6.2 Documentation improvements

`snps` could always use more documentation, whether as part of the official `snps` docs, in docstrings, or even on the web in blog posts, articles, and such. See below for info on how to generate documentation.

### 6.3 Feature requests and feedback

The best way to send feedback is to file an issue at <https://github.com/apriha/snps/issues>.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

## 6.4 Development

To set up snps for local development:

1. Fork [snps](#) (look for the “Fork” button).
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/snps.git
```

3. Create a branch for local development from the develop branch:

```
$ cd snps
$ git checkout develop
$ git checkout -b name-of-your-bugfix-or-feature develop
```

4. Setup a development environment:

```
$ pip install pipenv
$ pipenv install --dev
```

5. When you’re done making changes, run all the tests with:

```
$ pipenv run pytest --cov-report=html --cov=snps tests
```

---

**Note:** Downloads during tests are disabled by default. To enable downloads, set the environment variable `DOWNLOADS_ENABLED=true`.

---

---

**Note:** If you receive errors when running the tests, you may need to specify the temporary directory with an environment variable, e.g., `TMPDIR="/path/to/tmp/dir"`.

---

---

**Note:** After running the tests, a coverage report can be viewed by opening `htmlcov/index.html` in a browser.

---

6. Check code formatting:

```
$ pipenv run black --check --diff .
```

7. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

8. Submit a pull request through the GitHub website.



### 6.4.1 Pull request guidelines

If you need some code review or feedback while you're developing the code, just make the pull request.

For merging, you should:

1. Ensure tests pass.
2. Update documentation when there's new API, functionality, etc.
3. Add yourself to `CONTRIBUTORS.rst` if you'd like.

## 6.5 Documentation

After the development environment has been setup, documentation can be generated via the following command:

```
$ pipenv run sphinx-build -T -E -D language=en docs docs/_build
```

Then, the documentation can be viewed by opening `docs/_build/index.html` in a browser.



## CONTRIBUTORS

Contributors to snps are listed below.

### 7.1 Core Developers

Name	GitHub
Andrew Riha	@apriha
Will Jones	@willgdjones

### 7.2 Other Contributors

Listed in alphabetical order.

Name	GitHub
Alan Moffet	@amoffet
Anatoli Babenia	@abitrolly
Castedo Ellerman	@castedo
Gerard Manning	@GerardManning
Julian Runnels	@JulianRunnels
Kevin Arvai	@arvkevi
Phil Palmer	@PhilPalmer
Yoan Bouzin	



## CODE DOCUMENTATION

### 8.1 SNPs

SNPs reads, writes, merges, and remaps genotype / raw data files.

```
class snps.snps.SNPs(file="", only_detect_source=False, assign_par_snps=False, output_dir='output',
                     resources_dir='resources', deduplicate=True, deduplicate_XY_chrom=True,
                     deduplicate_MT_chrom=True, parallelize=False, processes=2, rsids=())
```

Bases: object

```
__init__(file="", only_detect_source=False, assign_par_snps=False, output_dir='output',
          resources_dir='resources', deduplicate=True, deduplicate_XY_chrom=True,
          deduplicate_MT_chrom=True, parallelize=False, processes=2, rsids=())
```

Object used to read, write, and remap genotype / raw data files.

#### Parameters

- **file** (*str or bytes*) – path to file to load or bytes to load
- **only\_detect\_source** (*bool*) – only detect the source of the data
- **assign\_par\_snps** (*bool*) – assign PAR SNPs to the X and Y chromosomes
- **output\_dir** (*str*) – path to output directory
- **resources\_dir** (*str*) – name / path of resources directory
- **deduplicate** (*bool*) – deduplicate RSIDs and make SNPs available as *SNPs.duplicate*
- **deduplicate\_MT\_chrom** (*bool*) – deduplicate alleles on MT; see *SNPs.heterozygous\_MT*
- **deduplicate\_XY\_chrom** (*bool or str*) – deduplicate alleles in the non-PAR regions of X and Y for males; see *SNPs.discrepant\_XY* if a *str* then this is the sex determination method to use X Y or XY
- **parallelize** (*bool*) – utilize multiprocessing to speedup calculations
- **processes** (*int*) – processes to launch if multiprocessing
- **rsids** (*tuple, optional*) – rsids to extract if loading a VCF file

#### property assembly

Assembly of SNPs.

#### Returns

**Return type** str

#### property build

Build of SNPs.

**Returns****Return type** int**property build\_detected**

Status indicating if build of SNPs was detected.

**Returns****Return type** bool**property chip**Detected deduced genotype / chip array, if any, per [compute\\_cluster\\_overlap](#).**Returns** detected chip array, else empty str**Return type** str**property chip\_version**Detected genotype / chip array version, if any, per [compute\\_cluster\\_overlap](#).**Notes**

Chip array version is only applicable to 23andMe (v3, v4, v5) and AncestryDNA (v1, v2) files.

**Returns** detected chip array version, e.g., 'v4', else empty str**Return type** str**property chromosomes**

Chromosomes of SNPs.

**Returns** list of str chromosomes (e.g., ['1', '2', '3', 'MT']), empty list if no chromosomes**Return type** list**property chromosomes\_summary**

Summary of the chromosomes of SNPs.

**Returns** human-readable listing of chromosomes (e.g., '1-3, MT'), empty str if no chromosomes**Return type** str**property cluster**Detected chip cluster, if any, per [compute\\_cluster\\_overlap](#).**Notes**Refer to [compute\\_cluster\\_overlap](#) for more details about chip clusters.**Returns** detected chip cluster, e.g., 'c1', else empty str**Return type** str**compute\_cluster\_overlap(cluster\_overlap\_threshold=0.95)**

Compute overlap with chip clusters.

Chip clusters, which are defined in<sup>1</sup>, are associated with deduced genotype / chip arrays and DTC companies.

---

<sup>1</sup> Chang Lu, Bastian Greshake Tzovaras, Julian Gough, A survey of direct-to-consumer genotype data, and quality control tool (GenomePrep) for research, Computational and Structural Biotechnology Journal, Volume 19, 2021, Pages 3747-3754, ISSN 2001-0370, <https://doi.org/10.1016/j.csbj.2021.06.040>.

This method also sets the values returned by the *cluster*, *chip*, and *chip\_version* properties, based on max overlap, if the specified threshold is satisfied.

**Parameters** *cluster\_overlap\_threshold* (*float*) – threshold for cluster to overlap this SNPs object, and vice versa, to set values returned by the *cluster*, *chip*, and *chip\_version* properties

**Returns**

pandas.DataFrame with the following columns:

*company\_composition* DTC company composition of associated cluster from [Page 26, 1](#)

*chip\_base\_deduced* deduced genotype / chip array of associated cluster from [Page 26, 1](#)

*snps\_in\_cluster* count of SNPs in cluster

*snps\_in\_common* count of SNPs in common with cluster (inner merge with cluster)

*overlap\_with\_cluster* percentage overlap of *snps\_in\_common* with cluster

*overlap\_with\_self* percentage overlap of *snps\_in\_common* with this SNPs object

**Return type** pandas.DataFrame

## References

**property count**

Count of SNPs.

**Returns**

**Return type** int

**detect\_build()**

Detect build of SNPs.

Use the coordinates of common SNPs to identify the build / assembly of a genotype file that is being loaded.

## Notes

- rs3094315 : plus strand in 36, 37, and 38
- rs11928389 : plus strand in 36, minus strand in 37 and 38
- rs2500347 : plus strand in 36 and 37, minus strand in 38
- rs964481 : plus strand in 36, 37, and 38
- rs2341354 : plus strand in 36, 37, and 38
- rs3850290 : plus strand in 36, 37, and 38
- rs1329546 : plus strand in 36, 37, and 38

**Returns** detected build of SNPs, else 0

**Return type** int

## References

1. Yates et. al. (doi:10.1093/bioinformatics/btu613), <http://europepmc.org/search/?query=DOI:10.1093/bioinformatics/btu613>
2. Zerbino et. al. (doi.org/10.1093/nar/gkx1098), <https://doi.org/10.1093/nar/gkx1098>
3. Sherry ST, Ward MH, Kholodov M, Baker J, Phan L, Smigielski EM, Sirotkin K. dbSNP: the NCBI database of genetic variation. Nucleic Acids Res. 2001 Jan 1;29(1):308-11.
4. Database of Single Nucleotide Polymorphisms (dbSNP). Bethesda (MD): National Center for Biotechnology Information, National Library of Medicine. dbSNP accession: rs3094315, rs11928389, rs2500347, rs964481, rs2341354, rs3850290, and rs1329546 (dbSNP Build ID: 151). Available from: <http://www.ncbi.nlm.nih.gov/SNP/>

**determine\_sex**(*heterozygous\_x\_snps\_threshold=0.03, y\_snps\_not\_null\_threshold=0.3, chrom='X'*)

Determine sex from SNPs using thresholds.

### Parameters

- **heterozygous\_x\_snps\_threshold** (*float*) – percentage heterozygous X SNPs; above this threshold, Female is determined
- **y\_snps\_not\_null\_threshold** (*float*) – percentage Y SNPs that are not null; above this threshold, Male is determined
- **chrom** (*{“X”, “Y”}*) – use X or Y chromosome SNPs to determine sex

**Returns** ‘Male’ or ‘Female’ if detected, else empty str

**Return type** str

**property discrepant\_XY**

Discrepant XY SNPs.

A discrepant XY SNP is a heterozygous SNP in the non-PAR region of the X or Y chromosome found during deduplication for a detected male genotype.

**Returns** normalized snps dataframe

**Return type** pandas.DataFrame

**property discrepant\_merge\_genotypes**

SNPs with discrepant genotypes discovered while merging SNPs.

## Notes

Definitions of columns in this dataframe are as follows:

Column	Description
rsid	SNP ID
chrom	Chromosome of existing SNP
pos	Position of existing SNP
genotype	Genotype of existing SNP
chrom_added	Chromosome of added SNP
pos_added	Position of added SNP
genotype_added	Genotype of added SNP (discrepant with genotype)

**Returns**



**Return type** pandas.DataFrame

**property discrepant\_merge\_positions**

SNPs with discrepant positions discovered while merging SNPs.

**Notes**

Definitions of columns in this dataframe are as follows:

Column	Description
rsid	SNP ID
chrom	Chromosome of existing SNP
pos	Position of existing SNP
genotype	Genotype of existing SNP
chrom_added	Chromosome of added SNP
pos_added	Position of added SNP (discrepant with pos)
genotype_added	Genotype of added SNP

**Returns**

**Return type** pandas.DataFrame

**property discrepant\_merge\_positions\_genotypes**

SNPs with discrepant positions and / or genotypes discovered while merging SNPs.

**Notes**

Definitions of columns in this dataframe are as follows:

Column	Description
rsid	SNP ID
chrom	Chromosome of existing SNP
pos	Position of existing SNP
genotype	Genotype of existing SNP
chrom_added	Chromosome of added SNP
pos_added	Position of added SNP (possibly discrepant with pos)
genotype_added	Genotype of added SNP (possibly discrepant with genotype)

**Returns**

**Return type** pandas.DataFrame

**property discrepant\_vcf\_position**

SNPs with discrepant positions discovered while saving VCF.

**Returns** normalized snps dataframe

**Return type** pandas.DataFrame

**property duplicate**

Duplicate SNPs.

A duplicate SNP has the same RSID as another SNP. The first occurrence of the RSID is not considered a duplicate SNP.

**Returns** normalized snps dataframe

**Return type** pandas.DataFrame

**get\_count**(*chrom=""*)

Count of SNPs.

**Parameters** **chrom** (*str, optional*) – chromosome (e.g., “1”, “X”, “MT”)

**Returns**

**Return type** int

**static get\_par\_regions**(*build*)

Get PAR regions for the X and Y chromosomes.

**Parameters** **build** (*int*) – build of SNPs

**Returns** PAR regions for the given build

**Return type** pandas.DataFrame

## References

1. Genome Reference Consortium, <https://www.ncbi.nlm.nih.gov/grc/human>
2. Yates et. al. (doi:10.1093/bioinformatics/btu613), <http://europepmc.org/search/?query=DOI:10.1093/bioinformatics/btu613>
3. Zerbino et. al. (doi.org/10.1093/nar/gkx1098), <https://doi.org/10.1093/nar/gkx1098>

**heterozygous**(*chrom=""*)

Get heterozygous SNPs.

**Parameters** **chrom** (*str, optional*) – chromosome (e.g., “1”, “X”, “MT”)

**Returns** normalized snps dataframe

**Return type** pandas.DataFrame

**property heterozygous\_MT**

Heterozygous SNPs on the MT chromosome found during deduplication.

**Returns** normalized snps dataframe

**Return type** pandas.DataFrame

**homozygous**(*chrom=""*)

Get homozygous SNPs.

**Parameters** **chrom** (*str, optional*) – chromosome (e.g., “1”, “X”, “MT”)

**Returns** normalized snps dataframe

**Return type** pandas.DataFrame

**identify\_low\_quality\_snps**()

Identify low quality SNPs based on chip clusters.

Any low quality SNPs are removed from the *snps\_qc* dataframe and are made available as *low\_quality*.

## Notes

Chip clusters, which are defined in [Page 26, 1](#), are associated with low quality SNPs. As such, low quality SNPs will only be identified when this SNPs object corresponds to a cluster per `compute_cluster_overlap()`.

### property `low_quality`

SNPs identified as low quality, if any, per `identify_low_quality_snps()`.

**Returns** normalized snps dataframe

**Return type** pandas.DataFrame

**merge**(*snps\_objects=()*, *discrepant\_positions\_threshold=100*, *discrepant\_genotypes\_threshold=500*, *remap=True*, *chrom=""*)

Merge other SNPs objects into this SNPs object.

### Parameters

- **snps\_objects** (list or tuple of SNPs) – other SNPs objects to merge into this SNPs object
- **discrepant\_positions\_threshold** (*int*) – threshold for discrepant SNP positions between existing data and data to be loaded; a large value could indicate mismatched genome assemblies
- **discrepant\_genotypes\_threshold** (*int*) – threshold for discrepant genotype data between existing data and data to be loaded; a large value could indicate mismatched individuals
- **remap** (*bool*) – if necessary, remap other SNPs objects to have the same build as this SNPs object before merging
- **chrom** (*str*, *optional*) – chromosome to merge (e.g., “1”, “Y”, “MT”)

### Returns

for each SNPs object to merge, a dict with the following items:

**merged** (**bool**) whether SNPs object was merged

**common\_rsids** (**pandas.Index**) SNPs in common

**discrepant\_position\_rsids** (**pandas.Index**) SNPs with discrepant positions

**discrepant\_genotype\_rsids** (**pandas.Index**) SNPs with discrepant genotypes

**Return type** list of dict

## References

1. Fluent Python by Luciano Ramalho (O’Reilly). Copyright 2015 Luciano Ramalho, 978-1-491-94600-8.

**notnull**(*chrom=""*)

Get not null genotype SNPs.

**Parameters** **chrom** (*str*, *optional*) – chromosome (e.g., “1”, “X”, “MT”)

**Returns** normalized snps dataframe

**Return type** pandas.DataFrame

### property `phased`

Indicates if genotype is phased.

**Returns**

**Return type** bool

**predict\_ancestry**(*output\_directory=None, write\_predictions=False, models\_directory=None, aisnps\_directory=None, n\_components=None, k=None, thousand\_genomes\_directory=None, samples\_directory=None, algorithm=None, aisnps\_set=None*)

Predict genetic ancestry for SNPs.

Predictions by [ezancestry](#).

## Notes

Populations below are described [here](#).

**Parameters** **various** (*optional*) – See the available settings for *predict* at [ezancestry](#).

## Returns

dict with the following keys:

**population\_code** (**str**) max predicted population for the sample

**population\_description** (**str**) descriptive name of the population

**population\_percent** (**float**) predicted probability for the max predicted population

**superpopulation\_code** (**str**) max predicted super population (continental) for the sample

**superpopulation\_description** (**str**) descriptive name of the super population

**superpopulation\_percent** (**float**) predicted probability for the max predicted super population

**ezancestry\_df** (**pandas.DataFrame**) **pandas.DataFrame** with the following columns:

**component1, component2, component3** The coordinates of the sample in the dimensionality-reduced component space. Can be used as (x, y, z,) coordinates for plotting in a 3d scatter plot.

**predicted\_population\_population** The max predicted population for the sample.

**ACB, ASW, BEB, CDX, CEU, CHB, CHS, CLM, ESN, FIN, GBR, GIH, GWD, IBS, ITU, JPT, KHV, LWK, M**  
Predicted probabilities for each of the populations. These sum to 1.0.

**predicted\_population\_superpopulation** The max predicted super population (continental) for the sample.

**AFR, AMR, EAS, EUR, SAS** Predicted probabilities for each of the super populations. These sum to 1.0.

**population\_description, superpopulation\_name** Descriptive names of the population and super population.

**Return type** dict

**remap**(*target\_assembly, complement\_bases=True*)

Remap SNP coordinates from one assembly to another.

This method uses the assembly map endpoint of the Ensembl REST API service (via `Resources's EnsemblRestClient`) to convert SNP coordinates / positions from one assembly to another. After remapping, the coordinates / positions for the SNPs will be that of the target assembly.

If the SNPs are already mapped relative to the target assembly, remapping will not be performed.

## Parameters

- **target\_assembly** (*{'NCBI36', 'GRCh37', 'GRCh38', 36, 37, 38}*) – assembly to remap to
- **complement\_bases** (*bool*) – complement bases when remapping SNPs to the minus strand

#### Returns

- **chromosomes\_remapped** (*list of str*) – chromosomes remapped
- **chromosomes\_not\_remapped** (*list of str*) – chromosomes not remapped

#### Notes

An assembly is also know as a “build.” For example:

Assembly NCBI36 = Build 36 Assembly GRCh37 = Build 37 Assembly GRCh38 = Build 38

See <https://www.ncbi.nlm.nih.gov/assembly> for more information about assemblies and remapping.

#### References

1. Ensembl, Assembly Map Endpoint, [http://rest.ensembl.org/documentation/info/assembly\\_map](http://rest.ensembl.org/documentation/info/assembly_map)
2. Yates et. al. (doi:10.1093/bioinformatics/btu613), <http://europepmc.org/search/?query=DOI:10.1093/bioinformatics/btu613>
3. Zerbino et. al. (doi.org/10.1093/nar/gkx1098), <https://doi.org/10.1093/nar/gkx1098>

#### property sex

Sex derived from SNPs.

**Returns** ‘Male’ or ‘Female’ if detected, else empty str

**Return type** str

#### property snps

Normalized SNPs.

#### Notes

Throughout snps, the “normalized snps dataframe” is defined as follows:

Column	Description	<i>pandas</i> dtype
rsid* <sup>0</sup>	SNP ID	object (string)
chrom	Chromosome of SNP	object (string)
pos	Position of SNP (relative to build)	uint32
genotype† <sup>0</sup>	Genotype of SNP	object (string)

**Returns** normalized snps dataframe

**Return type** pandas.DataFrame

<sup>0</sup> Dataframe index

<sup>0</sup> Genotype can be null, length 1, or length 2. Specifically, genotype is null if not called or unavailable. Otherwise, for autosomal chromosomes, genotype is two alleles. For the X and Y chromosomes, male genotypes are one allele in the non-PAR regions (assuming *deduplicate\_XY\_chrom*). For the MT chromosome, genotypes are one allele (assuming *deduplicate\_MT\_chrom*).

**property snps\_qc**

Normalized SNPs, after quality control.

Any low quality SNPs, identified per `identify_low_quality_snps()`, are not included in the result.

**Returns** normalized snps dataframe

**Return type** pandas.DataFrame

**sort()**

Sort SNPs based on ordered chromosome list and position.

**property source**

Summary of the SNP data source(s).

**Returns** Data source(s) for this SNPs object, separated by “;”.

**Return type** str

**property summary**

Summary of SNPs.

**Returns** summary info if SNPs is valid, else { }

**Return type** dict

**to\_csv(filename="", atomic=True, \*\*kwargs)**

Output SNPs as comma-separated values.

**Parameters**

- **filename** (*str or buffer*) – filename for file to save or buffer to write to
- **atomic** (*bool*) – atomically write output to a file on local filesystem
- **\*\*kwargs** – additional parameters to `pandas.DataFrame.to_csv`

**Returns** path to file in output directory if SNPs were saved, else empty str

**Return type** str

**to\_tsv(filename="", atomic=True, \*\*kwargs)**

Output SNPs as tab-separated values.

Note that this results in the same default output as `save`.

**Parameters**

- **filename** (*str or buffer*) – filename for file to save or buffer to write to
- **atomic** (*bool*) – atomically write output to a file on local filesystem
- **\*\*kwargs** – additional parameters to `pandas.DataFrame.to_csv`

**Returns** path to file in output directory if SNPs were saved, else empty str

**Return type** str

**to\_vcf(filename="", atomic=True, alt\_unavailable='.', chrom\_prefix="", qc\_only=False, qc\_filter=False, \*\*kwargs)**

Output SNPs as Variant Call Format.

**Parameters**

- **filename** (*str or buffer*) – filename for file to save or buffer to write to
- **atomic** (*bool*) – atomically write output to a file on local filesystem
- **alt\_unavailable** (*str*) – representation of ALT allele when ALT is not able to be determined

- **chrom\_prefix** (*str*) – prefix for chromosomes in VCF CHROM column
- **qc\_only** (*bool*) – output only SNPs that pass quality control
- **qc\_filter** (*bool*) – populate FILTER column based on quality control results
- **\*\*kwargs** – additional parameters to *pandas.DataFrame.to\_csv*

**Returns** path to file in output directory if SNPs were saved, else empty str

**Return type** str

### Notes

Parameters *qc\_only* and *qc\_filter*, if true, will identify low quality SNPs per *identify\_low\_quality\_snps()*, if not done already. Moreover, these parameters have no effect if this SNPs object does not map to a cluster per *compute\_cluster\_overlap()*.

### References

1. The Variant Call Format (VCF) Version 4.2 Specification, 8 Mar 2019, <https://samtools.github.io/hts-specs/VCFv4.2.pdf>

#### **property unannotated\_vcf**

Indicates if VCF file is unannotated.

**Returns**

**Return type** bool

#### **property valid**

Determine if SNPs is valid.

SNPs is valid when the input file has been successfully parsed.

**Returns** True if SNPs is valid

**Return type** bool

## 8.2 snps.ensembl

Ensembl REST client.

### Notes

Modified from <https://github.com/Ensembl/ensembl-rest/wiki/Example-Python-Client>.

## References

1. Yates et. al. (doi:10.1093/bioinformatics/btu613), <http://europepmc.org/search/?query=DOI:10.1093/bioinformatics/btu613>
2. Zerbino et. al. (doi.org/10.1093/nar/gkx1098), <https://doi.org/10.1093/nar/gkx1098>

```
class snps.ensembl.EnsemblRestClient(server='https://rest.ensembl.org', reqs_per_sec=15)
    Bases: object
    __init__(server='https://rest.ensembl.org', reqs_per_sec=15)
    perform_rest_action(endpoint, hdrs=None, params=None)
```

## 8.3 snps.io

Classes for reading and writing SNPs.

### 8.3.1 snps.io.reader

Class for reading SNPs.

```
class snps.io.reader.Reader(file="", only_detect_source=False, resources=None, rsids=())
    Bases: object
    Class for reading and parsing raw data / genotype files.
    __init__(file="", only_detect_source=False, resources=None, rsids=())
        Initialize a Reader.
```

#### Parameters

- **file** (*str or bytes*) – path to file to load or bytes to load
- **only\_detect\_source** (*bool*) – only detect the source of the data
- **resources** (*Resources*) – instance of Resources
- **rsids** (*tuple, optional*) – rsids to extract if loading a VCF file

```
static is_gzip(bytes_data)
    Check whether or not a bytes_data file is a valid gzip file.
```

```
static is_zip(bytes_data)
    Check whether or not a bytes_data file is a valid Zip file.
```

```
read()
    Read and parse a raw data / genotype file.
```

#### Returns

dict with the following items:

**snps** (**pandas.DataFrame**) dataframe of parsed SNPs

**source** (**str**) detected source of SNPs

**phased** (**bool**) flag indicating if SNPs are phased

**Return type** dict



**read\_23andme**(*file*, *compression*, *joined=True*)

Read and parse 23andMe file.

<https://www.23andme.com>

**Parameters** **file** (*str*) – path to file

**Returns** result of *read\_helper*

**Return type** dict

**read\_ancestry**(*file*, *compression*)

Read and parse Ancestry.com file.

<http://www.ancestry.com>

**Parameters** **file** (*str*) – path to file

**Returns** result of *read\_helper*

**Return type** dict

**read\_circledna**(*file*, *compression*)

Read and parse CircleDNA file.

<https://circledna.com/>

## Notes

This method attempts to read and parse a whole exome file, optionally compressed with gzip or zip. Some assumptions are made throughout this process:

- SNPs that are not annotated with an RSID are skipped
- Insertions and deletions are skipped

**Parameters** **file** (*str or bytes*) – path to file or bytes to load

**Returns** result of *read\_helper*

**Return type** dict

**read\_dna.land**(*file*, *compression*)

Read and parse DNA.land files.

<https://dna.land/>

**Parameters** **data** (*str*) – data string

**Returns** result of *read\_helper*

**Return type** dict

**read\_ftdna**(*file*, *compression*)

Read and parse Family Tree DNA (FTDNA) file.

<https://www.familytreedna.com>

**Parameters** **file** (*str*) – path to file

**Returns** result of *read\_helper*

**Return type** dict

**read\_ftdna\_famfinder**(*file, compression*)

Read and parse Family Tree DNA (FTDNA) “famfinder” file.

<https://www.familytreedna.com>

**Parameters** **file** (*str*) – path to file

**Returns** result of *read\_helper*

**Return type** dict

**read\_generic**(*file, compression, skip=1*)

Read and parse generic CSV or TSV file.

### Notes

Assumes columns are ‘rsid’, ‘chrom’ / ‘chromosome’, ‘pos’ / ‘position’, and ‘genotype’; values are comma separated; unreported genotypes are indicated by ‘-’; and one header row precedes data. For example:

rsid,chromosome,position,genotype rs1,1,1,AA rs2,1,2,CC rs3,1,3,-

**Parameters** **file** (*str*) – path to file

**Returns** result of *read\_helper*

**Return type** dict

**read\_genes\_for\_good**(*file, compression*)

Read and parse Genes For Good file.

<https://genesforgood.sph.umich.edu/readme/readme1.2.txt>

**Parameters** **file** (*str*) – path to file

**Returns** result of *read\_helper*

**Return type** dict

**read\_gsa**(*data\_or\_filename, compression, comments*)

Read and parse Illumina Global Screening Array files

**Parameters** **data\_or\_filename** (*str or bytes*) – either the filename to read from or the bytes data itself

**Returns** result of *read\_helper*

**Return type** dict

**read\_helper**(*source, parser*)

Generic method to help read files.

#### Parameters

- **source** (*str*) – name of data source
- **parser** (*func*) – parsing function, which returns a tuple with the following items:
  - 0 (pandas.DataFrame)** dataframe of parsed SNPs (empty if only detecting source)
  - 1 (bool), optional** flag indicating if SNPs are phased
  - 2 (int), optional** detected build of SNPs

#### Returns

dict with the following items:

**snps** (**pandas.DataFrame**) dataframe of parsed SNPs

**source** (**str**) detected source of SNPs

**phased** (**bool**) flag indicating if SNPs are phased

**build** (**int**) detected build of SNPs

**Return type** dict

## References

1. Fluent Python by Luciano Ramalho (O'Reilly). Copyright 2015 Luciano Ramalho, 978-1-491-94600-8.

**read\_livingdna**(*file, compression*)

Read and parse LivingDNA file.

<https://livingdna.com/>

**Parameters** **file** (*str*) – path to file

**Returns** result of *read\_helper*

**Return type** dict

**read\_mapmygenome**(*file, compression, header*)

Read and parse Mapmygenome file.

<https://mapmygenome.in>

**Parameters** **file** (*str*) – path to file

**Returns** result of *read\_helper*

**Return type** dict

**read\_myheritage**(*file, compression*)

Read and parse MyHeritage file.

<https://www.myheritage.com>

**Parameters** **file** (*str*) – path to file

**Returns** result of *read\_helper*

**Return type** dict

**read\_snps\_csv**(*file, comments, compression*)

Read and parse CSV file generated by snps.

<https://pypi.org/project/snps/>

**Parameters**

- **file** (*str or buffer*) – path to file or buffer to read
- **comments** (*str*) – comments at beginning of file

**Returns** result of *read\_helper*

**Return type** dict

**read\_tellmegen**(*file, compression*)

Read and parse tellmeGen files.

<https://www.tellmegen.com/>

**Parameters** *data* (*str*) – data string

**Returns** result of *read\_helper*

**Return type** dict

**read\_vcf**(*file*, *compression*, *provider*, *rsids*=())

Read and parse VCF file.

### Notes

This method attempts to read and parse a VCF file or buffer, optionally compressed with gzip. Some assumptions are made throughout this process:

- SNPs that are not annotated with an RSID are skipped
- If the VCF contains multiple samples, only the first sample is used to lookup the genotype
- Insertions and deletions are skipped
- If a sample allele is not specified, the genotype is reported as NaN
- If a sample allele refers to a REF or ALT allele that is not specified, the genotype is reported as NaN

### Parameters

- **file** (*str* or *bytes*) – path to file or bytes to load
- **rsids** (*tuple*, *optional*) – rsids to extract if loading a VCF file

**Returns** result of *read\_helper*

**Return type** dict

**snps.io.reader.get\_empty\_snps\_dataframe()**

Get empty dataframe normalized for usage with snps.

**Returns**

**Return type** pd.DataFrame

## 8.3.2 snps.io.writer

Class for writing SNPs.

**class** snps.io.writer.**Writer**(*snps=None*, *filename=""*, *vcf=False*, *atomic=True*, *vcf\_alt\_unavailable='.'*, *vcf\_chrom\_prefix=""*, *vcf\_qc\_only=False*, *vcf\_qc\_filter=False*, *\*\*kwargs*)

Bases: object

Class for writing SNPs to files.

**\_\_init\_\_**(*snps=None*, *filename=""*, *vcf=False*, *atomic=True*, *vcf\_alt\_unavailable='.'*, *vcf\_chrom\_prefix=""*, *vcf\_qc\_only=False*, *vcf\_qc\_filter=False*, *\*\*kwargs*)

Initialize a *Writer*.

### Parameters

- **snps** (*SNPs*) – SNPs to save to file or write to buffer
- **filename** (*str* or *buffer*) – filename for file to save or buffer to write to
- **vcf** (*bool*) – flag to save file as VCF

- **atomic** (*bool*) – atomically write output to a file on local filesystem
- **vcf\_alt\_unavailable** (*str*) – representation of VCF ALT allele when ALT is not able to be determined
- **vcf\_chrom\_prefix** (*str*) – prefix for chromosomes in VCF CHROM column
- **vcf\_qc\_only** (*bool*) – for VCF, output only SNPs that pass quality control
- **vcf\_qc\_filter** (*bool*) – for VCF, populate VCF FILTER column based on quality control results
- **\*\*kwargs** – additional parameters to *pandas.DataFrame.to\_csv*

**write()**

Write SNPs to file or buffer.

#### Returns

- *str* – path to file in output directory if SNPs were saved, else empty *str*
- **discrepant\_vcf\_position** (*pd.DataFrame*) – SNPs with discrepant positions discovered while saving VCF

## 8.4 snps.resources

Class for downloading and loading required external resources.

### References

1. International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. Nature. 2001 Feb 15;409(6822):860-921. <http://dx.doi.org/10.1038/35057062>
2. hg19 (GRCh37): Hiram Clawson, Brooke Rhead, Pauline Fujita, Ann Zweig, Katrina Learned, Donna Karolchik and Robert Kuhn, <https://genome.ucsc.edu/cgi-bin/hgGateway?db=hg19>
3. Yates et. al. (doi:10.1093/bioinformatics/btu613), <http://europepmc.org/search/?query=DOI:10.1093/bioinformatics/btu613>
4. Zerbino et. al. (doi.org/10.1093/nar/gkx1098), <https://doi.org/10.1093/nar/gkx1098>

**class** `snps.resources.ReferenceSequence`(*ID*="", *url*="", *path*="", *assembly*="", *species*="", *taxonomy*="")

Bases: `object`

Object used to represent and interact with a reference sequence.

#### property **ID**

Get reference sequence chromosome.

#### Returns

**Return type** `str`

**\_\_init\_\_**(*ID*="", *url*="", *path*="", *assembly*="", *species*="", *taxonomy*="")

Initialize a `ReferenceSequence` object.

#### Parameters

- **ID** (*str*) – reference sequence chromosome
- **url** (*str*) – url to Ensembl reference sequence
- **path** (*str*) – path to local reference sequence

- **assembly** (*str*) – reference sequence assembly (e.g., “GRCh37”)
- **species** (*str*) – reference sequence species
- **taxonomy** (*str*) – reference sequence taxonomy

## References

1. The Variant Call Format (VCF) Version 4.2 Specification, 8 Mar 2019, <https://samtools.github.io/hts-specs/VCFv4.2.pdf>

### **property assembly**

Get reference sequence assembly.

**Returns**

**Return type** `str`

### **property build**

Get reference sequence build.

**Returns** e.g., “B37”

**Return type** `str`

### **property chrom**

Get reference sequence chromosome.

**Returns**

**Return type** `str`

### **clear()**

Clear reference sequence.

### **property end**

Get reference sequence end position (1-based).

**Returns**

**Return type** `int`

### **property length**

Get reference sequence length.

**Returns**

**Return type** `int`

### **property md5**

Get reference sequence MD5 hash.

**Returns**

**Return type** `str`

### **property path**

Get path to local reference sequence.

**Returns**

**Return type** `str`

### **property sequence**

Get reference sequence.

**Returns****Return type** np.array(dtype=np.uint8)**property species**

Get reference sequence species.

**Returns****Return type** str**property start**

Get reference sequence start position (1-based).

**Returns****Return type** int**property taxonomy**

Get reference sequence taxonomy.

**Returns****Return type** str**property url**

Get URL to Ensembl reference sequence.

**Returns****Return type** str**class** snps.resources.**Resources**(\*args, \*\*kwargs)

Bases: object

Object used to manage resources required by *snps*.**\_\_init\_\_**(resources\_dir='resources')

Initialize a Resources object.

**Parameters** resources\_dir (str) – name / path of resources directory**download\_example\_datasets()**Download example datasets from [openSNP](#).Per openSNP, “the data is donated into the public domain using [CC0 1.0](#).”**Returns** paths – paths to example datasets**Return type** list of str or empty str**References**

1. Greshake B, Bayer PE, Rausch H, Reda J (2014), “openSNP-A Crowdsourced Web Resource for Personal Genomics,” PLOS ONE, 9(3): e89204, <https://doi.org/10.1371/journal.pone.0089204>

**get\_all\_reference\_sequences**(\*\*kwargs)

Get Homo sapiens reference sequences for Builds 36, 37, and 38 from Ensembl.

## Notes

This function can download over 2.5GB of data.

**Returns** dict of ReferenceSequence, else { }

**Return type** dict

### **get\_all\_resources()**

Get / download all resources used throughout *snps*.

## Notes

This function does not download reference sequences and the openSNP datadump, due to their large sizes.

**Returns** dict of resources

**Return type** dict

### **get\_assembly\_mapping\_data(source\_assembly, target\_assembly)**

Get assembly mapping data.

#### **Parameters**

- **source\_assembly** ({'NCBI36', 'GRCh37', 'GRCh38'}) – assembly to remap from
- **target\_assembly** ({'NCBI36', 'GRCh37', 'GRCh38'}) – assembly to remap to

**Returns** dict of json assembly mapping data if loading was successful, else { }

**Return type** dict

### **get\_chip\_clusters()**

Get resource for identifying deduced genotype / chip array based on chip clusters.

**Returns**

**Return type** pandas.DataFrame

## References

1. Chang Lu, Bastian Greshake Tzovaras, Julian Gough, A survey of direct-to-consumer genotype data, and quality control tool (GenomePrep) for research, Computational and Structural Biotechnology Journal, Volume 19, 2021, Pages 3747-3754, ISSN 2001-0370, <https://doi.org/10.1016/j.csbj.2021.06.040>.

### **get\_dbsnp\_151\_37\_reverse()**

Get and load RSIDs that are on the reference reverse (-) strand in dbSNP 151 and lower.

**Returns**

**Return type** pandas.DataFrame



## References

1. Sherry ST, Ward MH, Kholodov M, Baker J, Phan L, Smigielski EM, Sirotkin K. dbSNP: the NCBI database of genetic variation. *Nucleic Acids Res.* 2001 Jan 1; 29(1):308-11.
2. Database of Single Nucleotide Polymorphisms (dbSNP). Bethesda (MD): National Center for Biotechnology Information, National Library of Medicine. (dbSNP Build ID: 151). Available from: <http://www.ncbi.nlm.nih.gov/SNP/>

### **get\_gsa\_chrpos()**

Get and load GSA chromosome position map.

<https://support.illumina.com/downloads/infinium-global-screening-array-v2-0-product-files.html>

#### **Returns**

**Return type** pandas.DataFrame

### **get\_gsa\_resources()**

Get resources for reading Global Screening Array files.

<https://support.illumina.com/downloads/infinium-global-screening-array-v2-0-product-files.html>

#### **Returns**

**Return type** dict

### **get\_gsa\_rsid()**

Get and load GSA RSID map.

<https://support.illumina.com/downloads/infinium-global-screening-array-v2-0-product-files.html>

#### **Returns**

**Return type** pandas.DataFrame

### **get\_low\_quality\_snps()**

Get listing of low quality SNPs for quality control based on chip clusters.

#### **Returns**

**Return type** pandas.DataFrame

## References

1. Chang Lu, Bastian Greshake Tzovaras, Julian Gough, A survey of direct-to-consumer genotype data, and quality control tool (GenomePrep) for research, *Computational and Structural Biotechnology Journal*, Volume 19, 2021, Pages 3747-3754, ISSN 2001-0370, <https://doi.org/10.1016/j.csbj.2021.06.040>.

### **get\_opensnp\_datadump\_filenames()**

Get filenames internal to the [openSNP](#) datadump zip.

Per openSNP, “the data is donated into the public domain using [CC0 1.0](#).”

## Notes

This function can download over 27GB of data. If the download is not successful, try using a different tool like *wget* or *curl* to download the file and move it to the resources directory (see *\_get\_path\_opensnp\_datadump*).

**Returns** **filenames** – filenames internal to the openSNP datadump

**Return type** list of str

## References

1. Greshake B, Bayer PE, Rausch H, Reda J (2014), “openSNP-A Crowdsourced Web Resource for Personal Genomics,” PLOS ONE, 9(3): e89204, <https://doi.org/10.1371/journal.pone.0089204>

**get\_reference\_sequences**(*assembly*='GRCh37', *chroms*=('1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', 'X', 'Y', 'MT'))

Get Homo sapiens reference sequences for *chroms* of *assembly*.

## Notes

This function can download over 800MB of data for each assembly.

### Parameters

- **assembly** ({'NCBI36', 'GRCh37', 'GRCh38'}) – reference sequence assembly
- **chroms** (list of str) – reference sequence chromosomes

**Returns** dict of ReferenceSequence, else { }

**Return type** dict

**load\_opensnp\_datadump\_file**(*filename*)

Load the specified file from the openSNP datadump.

Per openSNP, “the data is donated into the public domain using [CC0 1.0](#).”

**Parameters** **filename** (str) – filename internal to the openSNP datadump

**Returns** content of specified file internal to the openSNP datadump

**Return type** bytes

## References

1. Greshake B, Bayer PE, Rausch H, Reda J (2014), “openSNP-A Crowdsourced Web Resource for Personal Genomics,” PLOS ONE, 9(3): e89204, <https://doi.org/10.1371/journal.pone.0089204>

## 8.5 snps.utils

Utility classes and functions.

**class** `snps.utils.Parallelizer`(*parallelize=False, processes=2*)

Bases: `object`

**\_\_init\_\_**(*parallelize=False, processes=2*)

Initialize a *Parallelizer*.

**Parameters**

- **parallelize** (*bool*) – utilize multiprocessing to speedup calculations
- **processes** (*int*) – processes to launch if multiprocessing

**class** `snps.utils.Singleton`

Bases: `type`

`snps.utils.clean_str`(*s*)

Clean a string so that it can be used as a Python variable name.

**Parameters** *s* (*str*) – string to clean

**Returns** string that can be used as a Python variable name

**Return type** `str`

`snps.utils.create_dir`(*path*)

Create directory specified by *path* if it doesn't already exist.

**Parameters** *path* (*str*) – path to directory

**Returns** True if *path* exists

**Return type** `bool`

`snps.utils.gzip_file`(*src, dest*)

Gzip a file.

**Parameters**

- **src** (*str*) – path to file to gzip
- **dest** (*str*) – path to output gzip file

**Returns** path to gzipped file

**Return type** `str`

`snps.utils.save_df_as_csv`(*df, path, filename, comment="", prepend\_info=True, atomic=True, \*\*kwargs*)

Save dataframe to a CSV file.

**Parameters**

- **df** (*pandas.DataFrame*) – dataframe to save
- **path** (*str*) – path to directory where to save CSV file
- **filename** (*str or buffer*) – filename for file to save or buffer to write to
- **comment** (*str*) – header comment(s); one or more lines starting with '#'
- **prepend\_info** (*bool*) – prepend file generation information as comments
- **atomic** (*bool*) – atomically write output to a file on local filesystem
- **\*\*kwargs** – additional parameters to *pandas.DataFrame.to\_csv*

**Returns** path to saved file or buffer (empty str if error)

**Return type** str or buffer

`snps.utils.zip_file(src, dest, arcname)`

Zip a file.

**Parameters**

- **src** (*str*) – path to file to zip
- **dest** (*str*) – path to output zip file
- **arcname** (*str*) – name of file in zip archive

**Returns** path to zipped file

**Return type** str

## INDICES AND TABLES

- `genindex`
- `modindex`



## PYTHON MODULE INDEX

### S

`snps.ensembl`, 35  
`snps.io`, 36  
`snps.io.reader`, 36  
`snps.io.writer`, 40  
`snps.resources`, 41  
`snps.snps`, 25  
`snps.utils`, 47





## Symbols

`__init__()` (*snps.ensembl.EnsemblRestClient* method), 36  
`__init__()` (*snps.io.reader.Reader* method), 36  
`__init__()` (*snps.io.writer.Writer* method), 40  
`__init__()` (*snps.resources.ReferenceSequence* method), 41  
`__init__()` (*snps.resources.Resources* method), 43  
`__init__()` (*snps.snps.SNPs* method), 25  
`__init__()` (*snps.utils.Parallelizer* method), 47

## A

`assembly` (*snps.resources.ReferenceSequence* property), 42  
`assembly` (*snps.snps.SNPs* property), 25

## B

`build` (*snps.resources.ReferenceSequence* property), 42  
`build` (*snps.snps.SNPs* property), 25  
`build_detected` (*snps.snps.SNPs* property), 26

## C

`chip` (*snps.snps.SNPs* property), 26  
`chip_version` (*snps.snps.SNPs* property), 26  
`chrom` (*snps.resources.ReferenceSequence* property), 42  
`chromosomes` (*snps.snps.SNPs* property), 26  
`chromosomes_summary` (*snps.snps.SNPs* property), 26  
`clean_str()` (in module *snps.utils*), 47  
`clear()` (*snps.resources.ReferenceSequence* method), 42  
`cluster` (*snps.snps.SNPs* property), 26  
`compute_cluster_overlap()` (*snps.snps.SNPs* method), 26  
`count` (*snps.snps.SNPs* property), 27  
`create_dir()` (in module *snps.utils*), 47

## D

`detect_build()` (*snps.snps.SNPs* method), 27  
`determine_sex()` (*snps.snps.SNPs* method), 28  
`discrepant_merge_genotypes` (*snps.snps.SNPs* property), 28  
`discrepant_merge_positions` (*snps.snps.SNPs* property), 29

`discrepant_merge_positions_genotypes` (*snps.snps.SNPs* property), 29  
`discrepant_vcf_position` (*snps.snps.SNPs* property), 29  
`discrepant_XY` (*snps.snps.SNPs* property), 28  
`download_example_datasets()` (*snps.resources.Resources* method), 43  
`duplicate` (*snps.snps.SNPs* property), 29

## E

`end` (*snps.resources.ReferenceSequence* property), 42  
`EnsemblRestClient` (class in *snps.ensembl*), 36

## G

`get_all_reference_sequences()` (*snps.resources.Resources* method), 43  
`get_all_resources()` (*snps.resources.Resources* method), 44  
`get_assembly_mapping_data()` (*snps.resources.Resources* method), 44  
`get_chip_clusters()` (*snps.resources.Resources* method), 44  
`get_count()` (*snps.snps.SNPs* method), 30  
`get_dbsnp_151_37_reverse()` (*snps.resources.Resources* method), 44  
`get_empty_snps_dataframe()` (in module *snps.io.reader*), 40  
`get_gsa_chrpos()` (*snps.resources.Resources* method), 45  
`get_gsa_resources()` (*snps.resources.Resources* method), 45  
`get_gsa_rsid()` (*snps.resources.Resources* method), 45  
`get_low_quality_snps()` (*snps.resources.Resources* method), 45  
`get_opensnp_datadump_filenames()` (*snps.resources.Resources* method), 45  
`get_par_regions()` (*snps.snps.SNPs* static method), 30  
`get_reference_sequences()` (*snps.resources.Resources* method), 46  
`gzip_file()` (in module *snps.utils*), 47

## H

heterozygous() (*snps.snps.SNPs* method), 30  
heterozygous\_MT (*snps.snps.SNPs* property), 30  
homozygous() (*snps.snps.SNPs* method), 30

## I

ID (*snps.resources.ReferenceSequence* property), 41  
identify\_low\_quality\_snps() (*snps.snps.SNPs* method), 30  
is\_gzip() (*snps.io.reader.Reader* static method), 36  
is\_zip() (*snps.io.reader.Reader* static method), 36

## L

length (*snps.resources.ReferenceSequence* property), 42  
load\_opensnp\_datadump\_file() (*snps.resources.Resources* method), 46  
low\_quality (*snps.snps.SNPs* property), 31

## M

md5 (*snps.resources.ReferenceSequence* property), 42  
merge() (*snps.snps.SNPs* method), 31  
module  
    *snps.ensembl*, 35  
    *snps.io*, 36  
    *snps.io.reader*, 36  
    *snps.io.writer*, 40  
    *snps.resources*, 41  
    *snps.snps*, 25  
    *snps.utils*, 47

## N

notnull() (*snps.snps.SNPs* method), 31

## P

Parallelizer (*class in snps.utils*), 47  
path (*snps.resources.ReferenceSequence* property), 42  
perform\_rest\_action() (*snps.ensembl.EnsemblRestClient* method), 36  
phased (*snps.snps.SNPs* property), 31  
predict\_ancestry() (*snps.snps.SNPs* method), 32

## R

read() (*snps.io.reader.Reader* method), 36  
read\_23andme() (*snps.io.reader.Reader* method), 36  
read\_ancestry() (*snps.io.reader.Reader* method), 37  
read\_circledna() (*snps.io.reader.Reader* method), 37  
read\_dnaland() (*snps.io.reader.Reader* method), 37  
read\_ftdna() (*snps.io.reader.Reader* method), 37  
read\_ftdna\_famfinder() (*snps.io.reader.Reader* method), 37  
read\_generic() (*snps.io.reader.Reader* method), 38

read\_genes\_for\_good() (*snps.io.reader.Reader* method), 38  
read\_gsa() (*snps.io.reader.Reader* method), 38  
read\_helper() (*snps.io.reader.Reader* method), 38  
read\_livingdna() (*snps.io.reader.Reader* method), 39  
read\_mapmygenome() (*snps.io.reader.Reader* method), 39  
read\_myheritage() (*snps.io.reader.Reader* method), 39  
read\_snps\_csv() (*snps.io.reader.Reader* method), 39  
read\_tellmegen() (*snps.io.reader.Reader* method), 39  
read\_vcf() (*snps.io.reader.Reader* method), 40  
*Reader* (*class in snps.io.reader*), 36  
*ReferenceSequence* (*class in snps.resources*), 41  
remap() (*snps.snps.SNPs* method), 32  
*Resources* (*class in snps.resources*), 43

## S

save\_df\_as\_csv() (*in module snps.utils*), 47  
sequence (*snps.resources.ReferenceSequence* property), 42  
sex (*snps.snps.SNPs* property), 33  
*Singleton* (*class in snps.utils*), 47  
*SNPs* (*class in snps.snps*), 25  
*snps* (*snps.snps.SNPs* property), 33  
*snps.ensembl*  
    module, 35  
*snps.io*  
    module, 36  
*snps.io.reader*  
    module, 36  
*snps.io.writer*  
    module, 40  
*snps.resources*  
    module, 41  
*snps.snps*  
    module, 25  
*snps.utils*  
    module, 47  
*snps\_qc* (*snps.snps.SNPs* property), 33  
sort() (*snps.snps.SNPs* method), 34  
source (*snps.snps.SNPs* property), 34  
species (*snps.resources.ReferenceSequence* property), 43  
start (*snps.resources.ReferenceSequence* property), 43  
summary (*snps.snps.SNPs* property), 34

## T

taxonomy (*snps.resources.ReferenceSequence* property), 43  
to\_csv() (*snps.snps.SNPs* method), 34  
to\_tsv() (*snps.snps.SNPs* method), 34  
to\_vcf() (*snps.snps.SNPs* method), 34

## U

`unannotated_vcf` (*snps.snps.SNPs* property), [35](#)

`url` (*snps.resources.ReferenceSequence* property), [43](#)

## V

`valid` (*snps.snps.SNPs* property), [35](#)

## W

`write()` (*snps.io.writer.Writer* method), [41](#)

`Writer` (class in *snps.io.writer*), [40](#)

## Z

`zip_file()` (in module *snps.utils*), [48](#)