snps Documentation

Release 2.12.1

Andrew Riha

Jan 28, 2026

CONTENTS

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

CHAPTER

ONE

SNPS

tools for reading, writing, generating, 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, historically validated using data from openSNP
- Generate synthetic genotype data for testing and examples

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
- 23Mofang
- Ancestry
- CircleDNA
- Código 46
- DNA.Land
- Family Tree DNA
- Genes for Good
- LivingDNA
- Mapmygenome
- MyHeritage
- PLINK
- Sano Genetics
- SelfDecode
- tellmeGen

Additionally, `snps` can read a variety of "generic" CSV and TSV files.

1.3 Dependencies

`snps` requires Python 3.9+ 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

To try these examples, first generate some sample data:

```
>>> from snps.resources import Resources
>>> paths = Resources().create_example_datasets()
```

1.5.1 Load a Raw Data File

Load a raw data file exported from a DNA testing source (e.g., 23andMe, AncestryDNA, Family Tree DNA):

```
>>> from snps import SNPs
>>> s = SNPs("resources/sample1.23andme.txt.gz")
```

snps automatically detects the source format and normalizes the data:

```
>>> s.source
'23andMe'
>>> s.count
991767
>>> s.build
37
>>> s.assembly
'GRCh37'
```

The SNPs are available as a pandas.DataFrame:

```
>>> df = s.snps
>>> df.columns.tolist()
['chrom', 'pos', 'genotype']
>>> len(df)
991767
```

1.5.2 Merge Raw Data Files

Combine SNPs from multiple files (e.g., combine data from different testing companies):

```
>>> results = s.merge([SNPs("resources/sample2.ftdna.csv.gz")])
>>> s.count
1006949
```

SNPs are compared during the merge. Position and genotype discrepancies are identified and can be inspected via properties of the SNPs object:

```
>>> len(s.discrepant_merge_positions)
27
>>> len(s.discrepant_merge_genotypes)
156
```

1.5.3 Remap SNPs

Convert SNPs between genome assemblies (Build 36/NCBI36, Build 37/GRCh37, Build 38/GRCh38):

```
>>> chromosomes_remapped, chromosomes_not_remapped = s.remap(38)
>>> s.assembly
'GRCh38'
```

1.5.4 Save SNPs

Save SNPs to common file formats:

```
>>> _ = s.to_tsv("output.txt")
>>> _ = s.to_csv("output.csv")
```

To save as VCF, `snps` automatically downloads the required reference sequences for the assembly. This ensures the REF alleles in the VCF are accurate:

```
>>> _ = s.to_vcf("output.vcf")
```

All output files are saved to the output directory.

1.5.5 Generate Synthetic Data

Generate synthetic genotype data for testing, examples, or demonstrations:

```
>>> from snps.io import SyntheticSNPGenerator
>>> gen = SyntheticSNPGenerator(build=37, seed=123)
>>> gen.save_as_23andme("synthetic_23andme.txt.gz", num_snps=10000)
'synthetic_23andme.txt.gz'
```

The generator supports multiple output formats (23andMe, AncestryDNA, FTDNA) and automatically injects build-specific marker SNPs to ensure accurate build detection.

1.6 Documentation

Documentation is available here.

1.7 Acknowledgements

Thanks to Mike Agostino, Padma Reddy, Kevin Arvai, Open Humans, and Sano Genetics. This project was historically validated using data from openSNP.

`snps` incorporates code and concepts generated with the assistance of various generative AI tools (including but not limited to ChatGPT, Grok, and Claude).

1.8 License

`snps` is licensed under the BSD 3-Clause License.

CHAPTER

TWO

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
rsid	SNP ID
chromosome	Chromosome of SNP
position	Position of SNP
genotype	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.

CHAPTER

THREE

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 $
```

CHAPTER

FOUR

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/

CHAPTER

FIVE

CHANGELOG

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

CHAPTER

SIX

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 `main` branch:

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

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 README.md
```

> **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. Perform code linting and formatting:

```
$ pipenv run ruff check --fix
$ pipenv run ruff format
```

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.md` 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.

CHAPTER

SEVEN

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	

CHAPTER

EIGHT

API REFERENCE

This section documents the complete API for the `snps` library.

8.1 Core Classes

8.1.1 SNPs

The main class for reading, writing, and analyzing genotype data. `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 source

Summary of the SNP data source(s).

Returns
Data source(s) for this `SNPs` object, separated by ", ".

Return type
`str`

property snps

Normalized SNPs.

Notes

Throughout `snps`, the "normalized `snps` dataframe" is defined as follows:

Column	Description	*pandas* dtype
rsid*[0]	SNP ID	object (string)
chrom	Chromosome of SNP	object (string)
pos	Position of SNP (relative to build)	uint32
genotype†[0]	Genotype of SNP	object (string)

Returns
normalized `snps` dataframe

Return type
`pandas.DataFrame`

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`

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`

[0] Dataframe index

[0] 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 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 heterozygous_MT

Heterozygous SNPs on the MT chromosome found during deduplication.

Returns
normalized `snps` dataframe

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 low_quality

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

Returns
normalized `snps` dataframe

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

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)

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)

Return type

pandas.DataFrame

property build

Build of SNPs.

Return type

int

property build_detected

Status indicating if build of SNPs was detected.

Return type

bool

property build_original

Original build of SNPs, before any remapping.

Return type
int

property assembly

Assembly of SNPs.

Return type
str

property count

Count of SNPs.

Return type
int

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 sex

Sex derived from SNPs.

Returns
'Male' or 'Female' if detected, else empty str

Return type
str

property unannotated_vcf

Indicates if VCF file is unannotated.

Return type
bool

property phased

Indicates if genotype is phased.

Return type
bool

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

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

heterozygous(*chrom=''*)

Get heterozygous SNPs.

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

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

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 summary

Summary of SNPs.

Returns
summary info if SNPs is valid, else {}

Return type
dict

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

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 4.3 Specification, 27 Nov 2022, https://samtools.github.io/hts-specs/VCFv4.3.pdf

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/

get_count(*chrom=''*)

Count of SNPs.

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

Return type
`int`

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`

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

sort()

Sort SNPs based on ordered chromosome list and position.

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
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

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 indicated 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.

predict_ancestry(*output_directory=None*, *write_predictions=False*, *models_directory=None*, *aisnps_directory=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_percent **(float)**
predicted probability for the max predicted population

superpopulation_code **(str)**
max predicted super population (continental) for the sample

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_ancestry_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***, ***MSL***, ***MXL***, ***PEL***, ***PJL***, ***PUR***, ***STU***, ***TSI***, ***YRI***
Predicted probabilities for each of the populations. These sum to 1.0.

predicted_ancestry_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.

Return type
`dict`

`compute_cluster_overlap`(*cluster_overlap_threshold=0.95*)

Compute overlap with chip clusters.

Chip clusters, which are defined in[1], are associated with deduced genotype / chip arrays and DTC companies.

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[1]

[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.

chip_base_deduced
deduced genotype / chip array of associated cluster from[Page 30, 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

`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 30, 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()*.

8.2 I/O Operations

Modules for reading, writing, and generating SNP data files.

8.2.1 snps.io

Classes for reading, writing, and generating SNPs.

`class snps.io.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

`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

`classmethod read_file(file, only_detect_source, resources, rsids)`

`static is_zip(bytes_data)`

Check whether or not a bytes_data file is a valid Zip file.

`static is_gzip(bytes_data)`

Check whether or not a bytes_data file is a valid gzip file.

`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_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_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_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_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_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*, *comments*)

Read and parse Mapmygenome file.

https://mapmygenome.in

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_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_gsa(*data_or_filename*, *compresion*, *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_dnaland(*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_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_sano_dtc(*file*, *compression*)

Read and parse Sano Genetics DTC file.

https://sanogenetics.com

Parameters
file (str) – path to file

Returns
result of *read_helper*

Return type
dict

read_selfdecode(*file*, *compression*)
Read and parse SelfDecode file.

https://selfdecode.com/

Parameters
file (str) – path to file

Returns
result of *read_helper*

Return type
dict

read_23Mofang(*file*, *compression*)
Read and parse 23Mofang file.

https://www.23mofang.com/

Parameters
file (str) – path to file

Returns
result of *read_helper*

Return type
dict

read_plink(*file*, *compression*)
Read and parse plink file.

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_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_vcf(*file*, *compression*, *provider*, *rsids=()*, *comments=''*)

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
> - Precise 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

class snps.io.**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

classmethod **write_file**(*snps=None*, *filename=''*, *vcf=False*, *atomic=True*, *vcf_alt_unavailable='.'*, *vcf_qc_only=False*, *vcf_qc_filter=False*, ***kwargs*)

class snps.io.**SyntheticSNPGenerator**(*build=37*, *seed=None*)

Bases: object

Generate realistic synthetic genotype data.

This class generates synthetic SNP data that mimics real genotype files from various DNA testing companies. The generated data is suitable for testing, examples, and documentation.

Parameters

- **build** (int) – Genome build (36, 37, or 38), default is 37
- **seed** (int, *optional*) – Random seed for reproducibility

Examples

```
>>> gen = SyntheticSNPGenerator(build=37, seed=123)
>>> gen.save_as_23andme("output.txt", num_snps=10000)
'output.txt'
```

__init__(*build=37*, *seed=None*)

Parameters

- **build** (*int*)
- **seed** (*int | None*)

Return type
None

generate_snps(*num_snps=10000*, *chromosomes=None*, *missing_rate=0.01*, *inject_build_markers=True*)

Generate a DataFrame of synthetic SNPs.

Parameters

- **num_snps** (int) – Approximate number of SNPs to generate
- **chromosomes** (list of str, *optional*) – Chromosomes to include (default: all autosomes plus X, Y, MT)
- **missing_rate** (float) – Proportion of SNPs with missing genotypes (default: 0.01)
- **inject_build_markers** (bool) – Inject known marker SNPs for build detection (default: True)

Returns
DataFrame with columns: rsid (index), chrom, pos, genotype

Return type
pd.DataFrame

create_example_dataset_pair(*output_dir='.'*)

Create a pair of realistic example datasets suitable for merging.

Generates two correlated genotype files that share a large number of common SNPs, with some discrepancies to demonstrate merge functionality.

Parameters
output_dir (str) – Directory for output files

Returns
Paths to (file1_23andme, file2_ftdna)

Return type
tuple of (str, str)

save_as_23andme(*output_path*, *num_snps=991786*, ***kwargs*)

Save SNPs in 23andMe format.

Parameters

- **output_path** (*str*)
- **num_snps** (*int*)
- **kwargs** (*Any*)

Return type
str

save_as_ancestry(*output_path*, *num_snps=700000*, ***kwargs*)

Save SNPs in AncestryDNA format.

Parameters

- **output_path** (*str*)
- **num_snps** (*int*)
- **kwargs** (*Any*)

Return type
str

save_as_ftdna(*output_path*, *num_snps=715194*, ***kwargs*)

Save SNPs in Family Tree DNA (FTDNA) format.

Parameters

- **output_path** (*str*)
- **num_snps** (*int*)
- **kwargs** (*Any*)

Return type
str

save_as_generic(*output_path*, *format='csv'*, *num_snps=10000*, ***kwargs*)

Save SNPs in generic CSV or TSV format.

Parameters

- **output_path** (*str*)
- **format** (*str*)
- **num_snps** (*int*)
- **kwargs** (*Any*)

Return type
str

snps.io.**get_empty_snps_dataframe**()

Get empty dataframe normalized for usage with snps.

Return type
pd.DataFrame

8.2.2 snps.io.reader

File format readers for various genotype data sources. Class for reading SNPs.

snps.io.reader.**get_empty_snps_dataframe**()

Get empty dataframe normalized for usage with snps.

Return type
pd.DataFrame

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

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

static is_zip(*bytes_data*)

Check whether or not a bytes_data file is a valid Zip file.

static is_gzip(*bytes_data*)

Check whether or not a bytes_data file is a valid gzip file.

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_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_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_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_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_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*, *comments*)

Read and parse Mapmygenome file.

https://mapmygenome.in

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_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_gsa(*data_or_filename*, *compresion*, *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_dnaland(*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_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_sano_dtc(*file*, *compression*)

Read and parse Sano Genetics DTC file.

https://sanogenetics.com

Parameters
file (str) – path to file

Returns
result of *read_helper*

Return type
dict

read_selfdecode(*file*, *compression*)

Read and parse SelfDecode file.

https://selfdecode.com/

Parameters
file (str) – path to file

Returns
result of *read_helper*

Return type
dict

read_23Mofang(*file*, *compression*)

Read and parse 23Mofang file.

https://www.23mofang.com/

Parameters
file (str) – path to file

Returns
result of *read_helper*

Return type
dict

read_plink(*file*, *compression*)

Read and parse plink file.

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_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_vcf(*file*, *compression*, *provider*, *rsids=()*, *comments=''*)

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
- Precise 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

8.2.3 snps.io.writer

File format writers for exporting genotype data. 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

8.2.4 snps.io.generator

Synthetic SNP data generation utilities. Generate synthetic genotype data for testing and examples.

class snps.io.generator.**SyntheticSNPGenerator**(*build=37*, *seed=None*)

Bases: `object`

Generate realistic synthetic genotype data.

This class generates synthetic SNP data that mimics real genotype files from various DNA testing companies. The generated data is suitable for testing, examples, and documentation.

Parameters

- **build** (`int`) – Genome build (36, 37, or 38), default is 37
- **seed** (`int`, *optional*) – Random seed for reproducibility

Examples

```
>>> gen = SyntheticSNPGenerator(build=37, seed=123)
>>> gen.save_as_23andme("output.txt", num_snps=10000)
'output.txt'
```

__init__(*build=37*, *seed=None*)

Parameters

- **build** (*int*)
- **seed** (*int | None*)

Return type
None

generate_snps(*num_snps=10000*, *chromosomes=None*, *missing_rate=0.01*, *inject_build_markers=True*)

Generate a DataFrame of synthetic SNPs.

Parameters

- **num_snps** (int) – Approximate number of SNPs to generate
- **chromosomes** (list of str, *optional*) – Chromosomes to include (default: all autosomes plus X, Y, MT)
- **missing_rate** (float) – Proportion of SNPs with missing genotypes (default: 0.01)
- **inject_build_markers** (bool) – Inject known marker SNPs for build detection (default: True)

Returns
DataFrame with columns: rsid (index), chrom, pos, genotype

Return type
pd.DataFrame

create_example_dataset_pair(*output_dir='.'*)

Create a pair of realistic example datasets suitable for merging.

Generates two correlated genotype files that share a large number of common SNPs, with some discrepancies to demonstrate merge functionality.

Parameters
output_dir (str) – Directory for output files

Returns
Paths to (file1_23andme, file2_ftdna)

Return type
tuple of (str, str)

save_as_23andme(*output_path*, *num_snps=991786*, ***kwargs*)

Save SNPs in 23andMe format.

Parameters

- **output_path** (*str*)
- **num_snps** (*int*)

- **kwargs** (*Any*)

Return type
str

save_as_ancestry(*output_path*, *num_snps=700000*, ***kwargs*)

Save SNPs in AncestryDNA format.

Parameters

- **output_path** (*str*)
- **num_snps** (*int*)
- **kwargs** (*Any*)

Return type
str

save_as_ftdna(*output_path*, *num_snps=715194*, ***kwargs*)

Save SNPs in Family Tree DNA (FTDNA) format.

Parameters

- **output_path** (*str*)
- **num_snps** (*int*)
- **kwargs** (*Any*)

Return type
str

save_as_generic(*output_path*, *format='csv'*, *num_snps=10000*, ***kwargs*)

Save SNPs in generic CSV or TSV format.

Parameters

- **output_path** (*str*)
- **format** (*str*)
- **num_snps** (*int*)
- **kwargs** (*Any*)

Return type
str

8.3 Data Resources

8.3.1 snps.ensembl

Interface to Ensembl REST API for genomic data. 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*)

8.3.2 snps.resources

Resource management for reference data and assembly mappings. 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.**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

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

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

create_example_datasets(*output_dir=None*)

Create synthetic example datasets for demonstrations.

Generates two correlated genotype files in different formats and builds, suitable for demonstrating merging and remapping functionality. The files share ~700K common SNPs with intentional discrepancies to demonstrate merge conflict detection.

Parameters
output_dir (str, *optional*) – Directory for output files (default: resources directory)

Returns
paths – Paths to created example datasets

Return type
list of str

Examples

```
>>> from snps.resources import Resources
>>> r = Resources()
>>> paths = r.create_example_datasets()
Creating resources/sample1.23andme.txt.gz
Creating resources/sample2.ftdna.csv.gz
```

get_all_resources()

Get / download all resources used throughout *snps*.

Notes

This function does not download reference sequences due to their large sizes.

Returns
dict of resources

Return type
dict

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_gsa_resources()

Get resources for reading Global Screening Array files.

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

Return type
dict

get_chip_clusters()

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

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.
2. Lu, Tzovaras, & Gough. (2021). OpenSNP data-freeze of 5,393 (19.10.2020) [Data set]. In Computational and Structural Biotechnology Journal. Zenodo. https://doi.org/10.1016/j.csbj.2021.06.040

get_low_quality_snps()

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

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.
2. Lu, Tzovaras, & Gough. (2021). OpenSNP data-freeze of 5,393 (19.10.2020) [Data set]. In Computational and Structural Biotechnology Journal. Zenodo. 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.

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_rsid()

Get and load GSA RSID map.

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

Return type
pandas.DataFrame

get_gsa_chrpos()

Get and load GSA chromosome position map.

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

Return type
pandas.DataFrame

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

Bases: object

Object used to represent and interact with a reference sequence.

__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 4.3 Specification, 27 Nov 2022, https://samtools.github.io/hts-specs/VCFv4.3.pdf

property ID
Get reference sequence chromosome.

Return type
str

property chrom
Get reference sequence chromosome.

Return type
str

property url
Get URL to Ensembl reference sequence.

Return type
str

property path
Get path to local reference sequence.

Return type
str

property assembly
Get reference sequence assembly.

Return type
str

property build
Get reference sequence build.

Returns
e.g., "B37"

Return type
str

property species
Get reference sequence species.

Return type
str

property taxonomy
Get reference sequence taxonomy.

Return type
str

property sequence
Get reference sequence.

Return type
np.array(dtype=np.uint8)

property md5
Get reference sequence MD5 hash.

Return type
str

property start

Get reference sequence start position (1-based).

Return type
int

property end

Get reference sequence end position (1-based).

Return type
int

property length

Get reference sequence length.

Return type
int

clear()

Clear reference sequence.

8.4 Utilities

8.4.1 snps.utils

Helper functions and utilities. 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

__call__(*f*, *tasks*)

Optionally parallelize execution of a function.

Parameters

- **f** (func) – function to execute
- **tasks** (list of dict) – tasks to pass to *f*

Returns
results of each call to *f*

Return type
list

class snps.utils.**Singleton**

Bases: type

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.**get_utc_now**()
Get current UTC time.

Return type
datetime.datetime

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.**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.**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

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

8.4.2 snps.testing

Shared test utilities for snps.

snps.testing.**get_complement**(*base*)

Get the complement of a DNA base.

Parameters
base (str) – A single DNA base (A, C, G, or T)

Returns
The complementary base (A<->T, C<->G), or the original if not a valid base

Return type
str

snps.testing.**complement_genotype**(*genotype*)

Get the complement of a genotype (both alleles).

Parameters
genotype (str) – A two-character genotype string (e.g., "AT", "CG")

Returns
The complemented genotype, or np.nan if input is null

Return type
str

snps.testing.**complement_one_allele**(*genotype*)

Get the complement of only the first allele of a genotype.

The second allele is preserved unchanged. This is useful for simulating partial strand complementation in test data.

Parameters
genotype (str) – A two-character genotype string (e.g., "AT", "CG")

Returns
Genotype with first allele complemented, or np.nan if input is null

Return type
str

snps.testing.**create_snp_df**(*rsid*, *chrom*, *pos*, *genotype*)

Create a normalized SNP DataFrame.

Parameters

- **rsid** (list of str) – SNP identifiers (becomes the index)
- **chrom** (list of str) – Chromosome values
- **pos** (list of int) – Position values
- **genotype** (list of str) – Genotype values

Returns
DataFrame with rsid index and chrom, pos, genotype columns

Return type
DataFrame

snps.testing.**create_simulated_snp_df**(*chrom='1'*, *pos_start=1*, *pos_max=248140902*, *pos_step=100*, *pos_dtype=<class 'numpy.uint32'>*, *genotype='AA'*, *insert_nulls=True*, *null_snp_step=101*, *complement_genotype_one_allele=False*, *complement_genotype_two_alleles=False*, *complement_snp_step=50*)

Create a simulated SNP DataFrame for testing.

This is the core logic for creating simulated SNP data. Each project can wrap this to assign to their specific object types.

Parameters

- **chrom** (str) – Chromosome value for all SNPs (default: "1")
- **pos_start** (int) – Starting position (default: 1)
- **pos_max** (int) – Maximum position (default: 248140902)
- **pos_step** (int) – Step between positions (default: 100)
- **pos_dtype** (type) – Numpy dtype for positions (default: np.uint32)
- **genotype** (str) – Default genotype for all SNPs (default: "AA")
- **insert_nulls** (bool) – Whether to insert null genotypes (default: True)
- **null_snp_step** (int) – Insert null every N SNPs (default: 101)
- **complement_genotype_one_allele** (bool) – Complement first allele at intervals (default: False)
- **complement_genotype_two_alleles** (bool) – Complement both alleles at intervals (default: False)
- **complement_snp_step** (int) – Apply complement every N SNPs (default: 50)

Returns
DataFrame with rsid index and chrom, pos, genotype columns

Return type
DataFrame

snps.testing.**assert_series_equal_with_string_dtype**(*left*, *right*, *test_case=None*, ***kwargs*)

Assert Series are equal, accepting both object and StringDtype for string data.

In Python 3.14+, pandas infers StringDtype for string data instead of object. This function compares Series without strict dtype matching for string data.

Parameters

- **left** (Series) – First Series to compare

- **right** (Series) – Second Series to compare
- **test_case** (object, *optional*) – Object with assertTrue method for assertions (uses assert if None)
- ****kwargs** (dict) – Additional arguments passed to pd.testing.assert_series_equal

Return type
None

snps.testing.**assert_frame_equal_with_string_index**(*left*, *right*, *test_case=None*, ***kwargs*)

Assert DataFrames are equal, accepting both object and StringDtype for string columns.

In Python 3.14+, pandas infers StringDtype for string columns/indices instead of object. This function validates that string columns have string types, then compares the DataFrames without strict dtype matching for object/string columns.

Parameters

- **left** (DataFrame) – First DataFrame to compare
- **right** (DataFrame) – Second DataFrame to compare
- **test_case** (object, *optional*) – Object with assertTrue method for assertions (uses assert if None)
- ****kwargs** (dict) – Additional arguments passed to pd.testing.assert_frame_equal

Return type
None

class snps.testing.**SNPsTestMixin**

Bases: object

Mixin class providing common test assertions and utilities for SNP DataFrames.

This mixin can be combined with unittest.TestCase to add convenient assertion methods for comparing SNP DataFrames with flexible string dtype handling, plus common test utilities like creating test DataFrames.

Example

```
>>> class MyTestCase(SNPsTestMixin, TestCase):
...     def test_something(self):
...         df = self.generic_snps()
...         self.assert_frame_equal_with_string_index(df, expected_df)
```

property downloads_enabled: bool

Check if external downloads are enabled for tests.

Only download from external resources when an environment variable named "DOWNLOADS_ENABLED" is set to "true".

Return type
bool

static get_complement(*base*)

Get the complement of a DNA base.

See *get_complement()* for details.

Parameters
base (*str*)

Return type
str

complement_genotype(*genotype*)

Get the complement of a genotype (both alleles).

See *complement_genotype()* for details.

Parameters
genotype (*str*)

Return type
str

complement_one_allele(*genotype*)

Get the complement of only the first allele of a genotype.

See *complement_one_allele()* for details.

Parameters
genotype (*str*)

Return type
str

static create_snp_df(*rsid*, *chrom*, *pos*, *genotype*)

Create a normalized SNP DataFrame.

See *create_snp_df()* for details.

Parameters

- **rsid** (*list[str]*)
- **chrom** (*list[str]*)
- **pos** (*list[int]*)
- **genotype** (*list[str]*)

Return type
DataFrame

generic_snps()

Create a generic SNP DataFrame for testing.

Returns
DataFrame with 8 SNPs (rs1-rs8) on chromosome 1

Return type
DataFrame

assert_series_equal_with_string_dtype(*left*, *right*, ***kwargs*)

Assert Series are equal, accepting both object and StringDtype for string data.

See *assert_series_equal_with_string_dtype()* for details.

assert_frame_equal_with_string_index(*left*, *right*, ***kwargs*)

Assert DataFrames are equal, accepting both object and StringDtype for string columns.

See *assert_frame_equal_with_string_index()* for details.

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

S

INDEX

Symbols

A

B

C

D

S

T

U

V

W

Z