lineage Documentation

Release 4.3.1.post12+ge90058a

Andrew Riha

Mar 20, 2024

CONTENTS

tools for genetic genealogy and the analysis of consumer DNA test results

CHAPTER

ONE

LINEAGE

`lineage` provides a framework for analyzing genotype (raw data) files from direct-to-consumer (DTC) DNA testing companies, primarily for the purposes of genetic genealogy.

1.1 Capabilities

- Find shared DNA and genes between individuals
- Compute centiMorgans (cMs) of shared DNA using a variety of genetic maps (e.g., HapMap Phase II, 1000 Genomes Project)
- Plot shared DNA between individuals
- Find discordant SNPs between child and parent(s)
- Read, write, merge, and remap SNPs for an individual via the snps package

1.2 Supported Genotype Files

`lineage` supports all genotype files supported by snps.

1.3 Installation

`lineage` is available on the Python Package Index. Install `lineage` (and its required Python dependencies) via `pip`:

```
$ pip install lineage
```

Also see the installation documentation.

1.4 Dependencies

`lineage` requires Python 3.8+ and the following Python packages:

- numpy
- pandas
- matplotlib
- atomicwrites
- snps

1.5 Examples

1.5.1 Initialize the lineage Framework

Import `Lineage` and instantiate a `Lineage` object:

```
>>> from lineage import Lineage
>>> l = Lineage()
```

1.5.2 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:

```
>>> paths = l.download_example_datasets()
Downloading resources/662.23andme.340.txt.gz
Downloading resources/662.ftdna-illumina.341.csv.gz
Downloading resources/663.23andme.305.txt.gz
Downloading resources/4583.ftdna-illumina.3482.csv.gz
Downloading resources/4584.ftdna-illumina.3483.csv.gz
```

We'll call these datasets `User662`, `User663`, `User4583`, and `User4584`.

1.5.3 Load Raw Data

Create an `Individual` in the context of the `lineage` framework to interact with the `User662` dataset:

```
>>> user662 = l.create_individual('User662', ['resources/662.23andme.340.txt.gz',
↪'resources/662.ftdna-illumina.341.csv.gz'])
Loading SNPs('662.23andme.340.txt.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
```

Here we created `user662` with the name `User662`. In the process, we merged two raw data files for this individual. Specifically:

- `662.23andme.340.txt.gz` was loaded.
- Then, `662.ftdna-illumina.341.csv.gz` was merged. In the process, it was found to have Build 36. So, it was automatically remapped to Build 37 (downloading the remapping data in the process) to match the build of the SNPs already loaded. After this merge, 27 SNP positions and 151 SNP genotypes were found to be discrepant.

`user662` is represented by an `Individual` object, which inherits from `snps.SNPs`. Therefore, all of the properties and methods available to a `SNPs` object are available here; for example:

```
>>> len(user662.discrepant_merge_genotypes)
151
>>> user662.build
37
>>> user662.build_detected
True
>>> user662.assembly
'GRCh37'
>>> user662.count
1006960
```

As such, SNPs can be saved, remapped, merged, etc. See the snps package for further examples.

1.5.4 Compare Individuals

Let's create another `Individual` for the `User663` dataset:

```
>>> user663 = l.create_individual('User663', 'resources/663.23andme.305.txt.gz')
Loading SNPs('663.23andme.305.txt.gz')
```

Now we can perform some analysis between the `User662` and `User663` datasets.

Find Discordant SNPs

First, let's find discordant SNPs (i.e., SNP data that is not consistent with Mendelian inheritance):

```
>>> discordant_snps = l.find_discordant_snps(user662, user663, save_output=True)
Saving output/discordant_snps_User662_User663_GRCh37.csv
```

All output files are saved to the output directory (a parameter to `Lineage`).

This method also returns a `pandas.DataFrame`, and it can be inspected interactively at the prompt, although the same output is available in the CSV file.

```
>>> len(discordant_snps.loc[discordant_snps['chrom'] != 'MT'])
37
```

Not counting mtDNA SNPs, there are 37 discordant SNPs between these two datasets.

Find Shared DNA

`lineage` uses the probabilistic recombination rates throughout the human genome from the International HapMap Project and the 1000 Genomes Project to compute the shared DNA (in centiMorgans) between two individuals. Additionally, `lineage` denotes when the shared DNA is shared on either one or both chromosomes in a pair. For example, when siblings share a segment of DNA on both chromosomes, they inherited the same DNA from their mother and father for that segment.

With that background, let's find the shared DNA between the `User662` and `User663` datasets, calculating the centiMorgans of shared DNA and plotting the results:

```
>>> results = l.find_shared_dna([user662, user663], cM_threshold=0.75, snp_
↪threshold=1100)
Downloading resources/genetic_map_HapMapII_GRCh37.tar.gz
Downloading resources/cytoBand_hg19.txt.gz
Saving output/shared_dna_User662_User663_0p75cM_1100snps_GRCh37_HapMap2.png
Saving output/shared_dna_one_chrom_User662_User663_0p75cM_1100snps_GRCh37_HapMap2.csv
```

Notice that the centiMorgan and SNP thresholds for each DNA segment can be tuned. Additionally, notice that two files were downloaded to facilitate the analysis and plotting - future analyses will use the downloaded files instead of downloading the files again. Finally, notice that a list of individuals is passed to `find_shared_dna`... This list can contain an arbitrary number of individuals, and `lineage` will find shared DNA across all individuals in the list (i.e., where all individuals share segments of DNA on either one or both chromosomes).

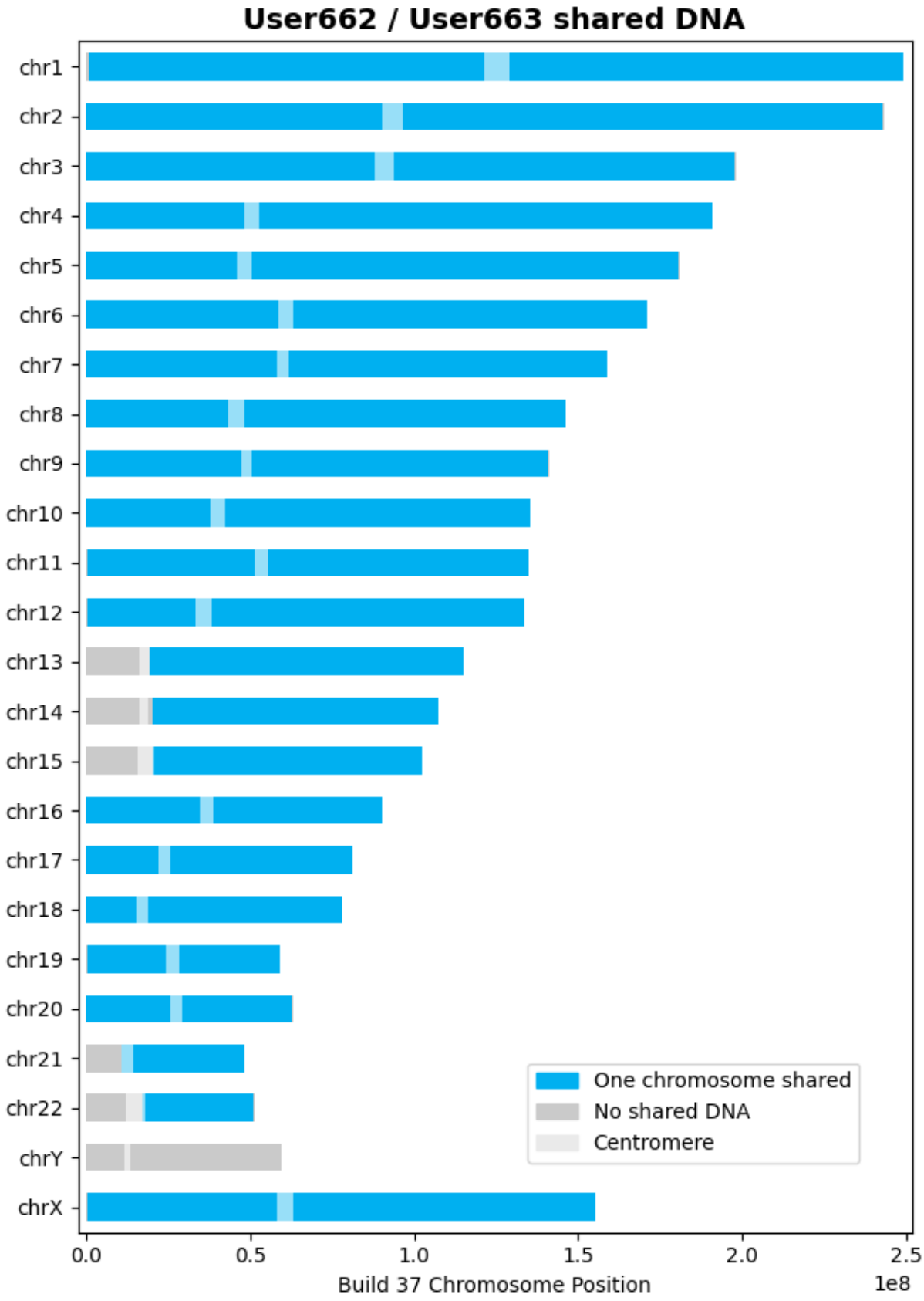
Output is returned as a dictionary with the following keys (`pandas.DataFrame` and `pandas.Index` items):

```
>>> sorted(results.keys())
['one_chrom_discrepant_snps', 'one_chrom_shared_dna', 'one_chrom_shared_genes', 'two_
↪chrom_discrepant_snps', 'two_chrom_shared_dna', 'two_chrom_shared_genes']
```

In this example, there are 27 segments of shared DNA:

```
>>> len(results['one_chrom_shared_dna'])
27
```

Also, output files are created; these files are detailed in the documentation and their generation can be disabled with a `save_output=False` argument. In this example, the output files consist of a CSV file that details the shared segments of DNA on one chromosome and a plot that illustrates the shared DNA:


User662 / User663 shared DNA
chr1
chr2
chr3
chr4
chr5
chr6
chr7
chr8
chr9
chr10
chr11
chr12
chr13
chr14
chr15
chr16
chr17
chr18
chr19
chr20
chr21
chr22
chrY
chrX
One chromosome shared
No shared DNA
Centromere
0.0
0.5
1.0
1.5
2.0
2.5
Build 37 Chromosome Position
1e8

Find Shared Genes

The Central Dogma of Molecular Biology states that genetic information flows from DNA to mRNA to proteins: DNA is transcribed into mRNA, and mRNA is translated into a protein. It's more complicated than this (it's biology after all), but generally, one mRNA produces one protein, and the mRNA / protein is considered a gene.

Therefore, it would be interesting to understand not just what DNA is shared between individuals, but what *genes* are shared between individuals *with the same variations*. In other words, what genes are producing the *same* proteins?*[0] Since `lineage` can determine the shared DNA between individuals, it can use that information to determine what genes are also shared on either one or both chromosomes.

For this example, let's create two more `Individuals` for the `User4583` and `User4584` datasets:

```
>>> user4583 = l.create_individual('User4583', 'resources/4583.ftdna-illumina.3482.csv.gz
↪')
Loading SNPs('4583.ftdna-illumina.3482.csv.gz')
```

```
>>> user4584 = l.create_individual('User4584', 'resources/4584.ftdna-illumina.3483.csv.gz
↪')
Loading SNPs('4584.ftdna-illumina.3483.csv.gz')
```

Now let's find the shared genes, specifying a population-specific 1000 Genomes Project genetic map (e.g., as predicted by ezancestry!):

```
>>> results = l.find_shared_dna([user4583, user4584], shared_genes=True, genetic_map="CEU
↪")
Downloading resources/CEU_omni_recombination_20130507.tar
Downloading resources/knownGene_hg19.txt.gz
Downloading resources/kgXref_hg19.txt.gz
Saving output/shared_dna_User4583_User4584_0p75cM_1100snps_GRCh37_CEU.png
Saving output/shared_dna_one_chrom_User4583_User4584_0p75cM_1100snps_GRCh37_CEU.csv
Saving output/shared_dna_two_chroms_User4583_User4584_0p75cM_1100snps_GRCh37_CEU.csv
Saving output/shared_genes_one_chrom_User4583_User4584_0p75cM_1100snps_GRCh37_CEU.csv
Saving output/shared_genes_two_chroms_User4583_User4584_0p75cM_1100snps_GRCh37_CEU.csv
```

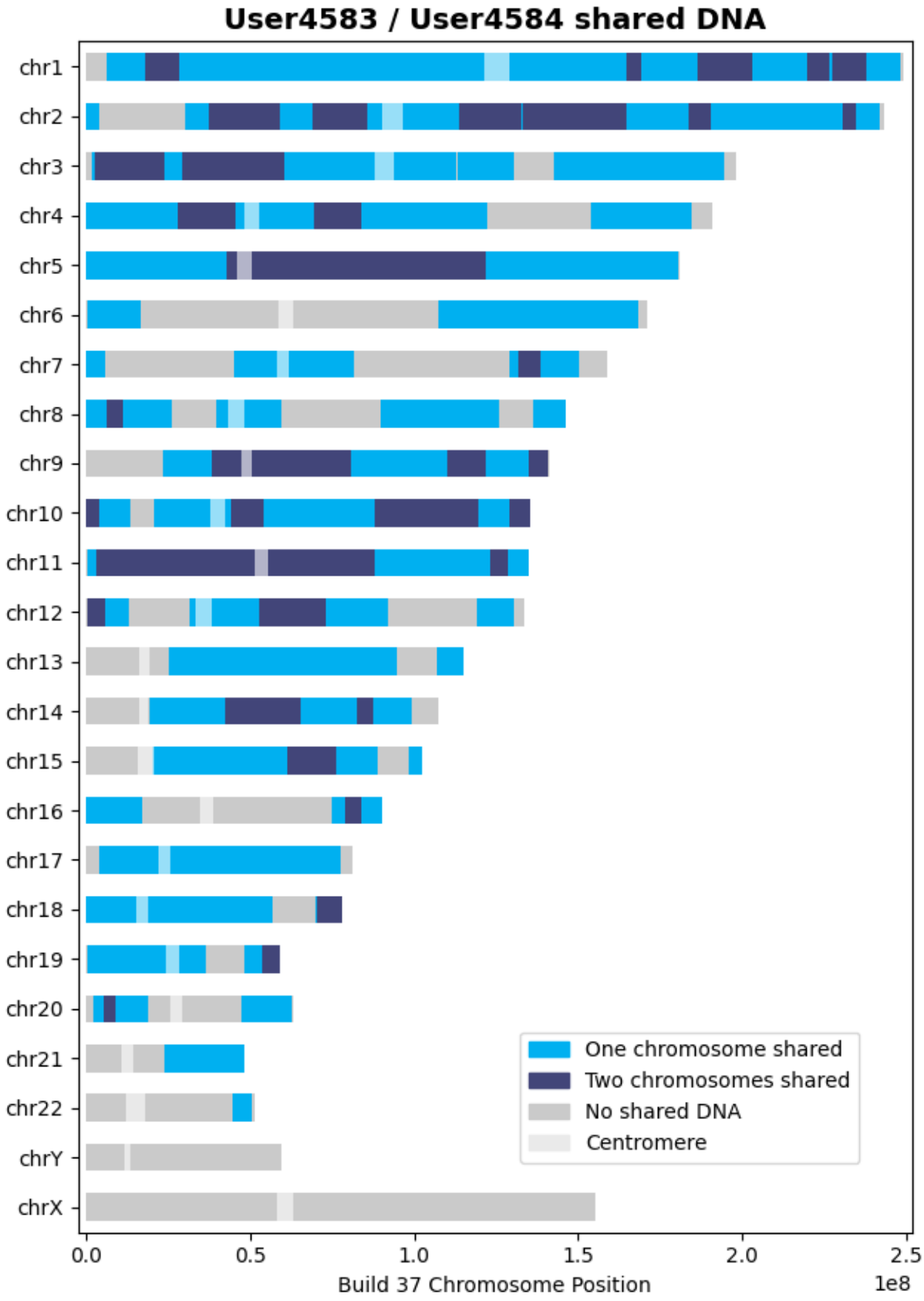
The plot that illustrates the shared DNA is shown below. Note that in addition to outputting the shared DNA segments on either one or both chromosomes, the shared genes on either one or both chromosomes are also output.

Note: Shared DNA is not computed on the X chromosome with the 1000 Genomes Project genetic maps since the X chromosome is not included in these genetic maps.

In this example, there are 15,976 shared genes on both chromosomes transcribed from 36 segments of shared DNA:

```
>>> len(results['two_chrom_shared_genes'])
15976
>>> len(results['two_chrom_shared_dna'])
36
```

[0] In theory, shared segments of DNA should be producing the same proteins, but there are many complexities, such as copy number variation (CNV), gene expression, etc.


User4583 / User4584 shared DNA
chr1
chr2
chr3
chr4
chr5
chr6
chr7
chr8
chr9
chr10
chr11
chr12
chr13
chr14
chr15
chr16
chr17
chr18
chr19
chr20
chr21
chr22
chrY
chrX
One chromosome shared
Two chromosomes shared
No shared DNA
Centromere
0.0
0.5
1.0
1.5
2.0
2.5
Build 37 Chromosome Position
1e8

1.6 Documentation

Documentation is available here.

1.7 Acknowledgements

Thanks to Whit Athey, Ryan Dale, Binh Bui, Jeff Gill, Gopal Vashishtha, CS50, and openSNP.

`lineage` incorporates code and concepts generated with the assistance of OpenAI's ChatGPT (GPT-3.5).

CHAPTER

TWO

OUTPUT FILES

The various output files produced by `lineage` are detailed below. Output files are saved in the output directory, which is defined at the instantiation of the *Lineage* object. Generation of output files can usually be enabled or disabled via a `save_output` argument to the associated method.

2.1 Save SNPs

See here.

2.2 Find Discordant SNPs

Discordant SNPs between two or three individuals can be identified with *find_discordant_snps()*. One CSV file is optionally output when `save_output=True`.

2.2.1 discordant_snps_<name1>_<name2>_GRCh37.csv

Where `name1` is the name of the first *Individual* and `name2` is the name of the second *Individual*.

Column	Description
rsid	SNP ID
chrom	Chromosome of SNP
pos	Position of SNP
genotype_<name1>	Genotype of first individual
genotype_<name2>	Genotype of second individual

2.2.2 discordant_snps_<name1>_<name2>_<name3>_GRCh37.csv

Where `name1` is the name of the first *Individual*, `name2` is the name of the second *Individual*, and `name3` is the name of the third *Individual*.

Column	Description
rsid	SNP ID
chrom	Chromosome of SNP
pos	Position of SNP
genotype_<name1>	Genotype of first individual
genotype_<name2>	Genotype of second individual
genotype_<name3>	Genotype of third individual

2.3 Find Shared DNA

Shared DNA between two or more individuals can be identified with *find_shared_dna()*. One PNG file and up to two CSV files are output when `save_output=True`.

In the filenames below,

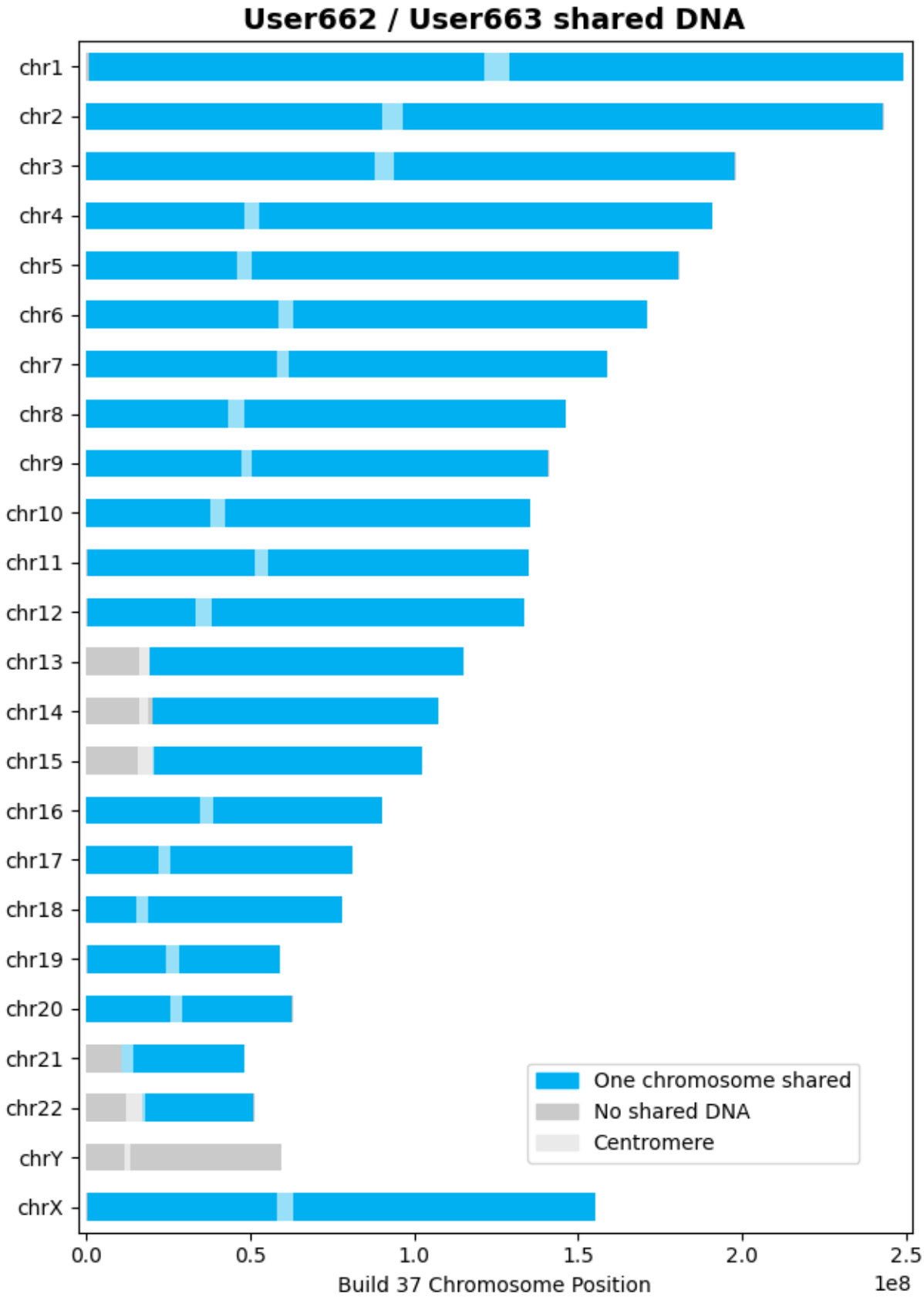
- `name1` is the name of the first *Individual*
- `name2` is the name of the second *Individual*
- `cM_threshold` corresponds to the same named parameter of *find_shared_dna()*; "." is replaced by "p" with precision of 2, e.g., "0p75"
- `snp_threshold` corresponds to the same named parameter of *find_shared_dna()*
- `genetic_map` corresponds to the same named parameter of *find_shared_dna()*.

Note: If more than two individuals are compared, all *Individual* names will be included in the filenames and plot titles using the same conventions.

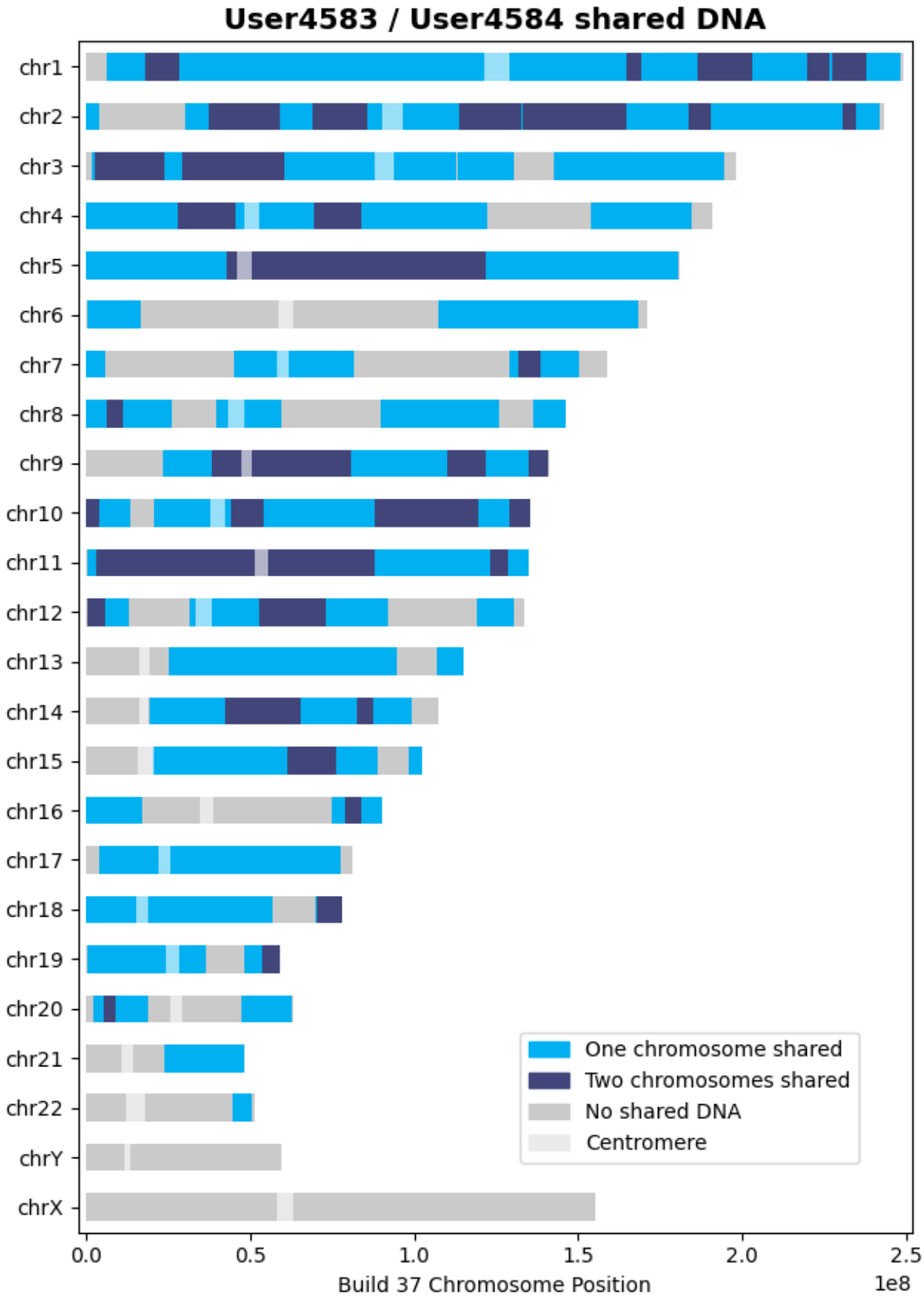
Note: Genetic maps do not have recombination rates for the Y chromosome since the Y chromosome does not recombine. Therefore, shared DNA will not be shown on the Y chromosome.

2.3.1 shared_dna_<name1>_<name2>_<cM_threshold>cM_<snp_threshold>snps_GRCh37_<ge

This plot illustrates shared DNA (i.e., no shared DNA, shared DNA on one chromosome, and shared DNA on both chromosomes). The centromere for each chromosome is also detailed. Two examples of this plot are shown below.


User662 / User663 shared DNA
chr1
chr2
chr3
chr4
chr5
chr6
chr7
chr8
chr9
chr10
chr11
chr12
chr13
chr14
chr15
chr16
chr17
chr18
chr19
chr20
chr21
chr22
chrY
chrX
One chromosome shared
No shared DNA
Centromere
0.0
0.5
1.0
1.5
2.0
2.5
Build 37 Chromosome Position
1e8

In the above plot, note that the two individuals only share DNA on one chromosome. In this plot, the larger regions where "No shared DNA" is indicated are due to SNPs not being available in those regions (i.e., SNPs were not tested in those regions).


User4583 / User4584 shared DNA
chr1
chr2
chr3
chr4
chr5
chr6
chr7
chr8
chr9
chr10
chr11
chr12
chr13
chr14
chr15
chr16
chr17
chr18
chr19
chr20
chr21
chr22
chrY
chrX
One chromosome shared
Two chromosomes shared
No shared DNA
Centromere
0.0
0.5
1.0
1.5
2.0
2.5
Build 37 Chromosome Position
1e8

In the above plot, the areas where "No shared DNA" is indicated are the regions where SNPs were not tested or where DNA is not shared. The areas where "One chromosome shared" is indicated are regions where the individuals share DNA on one chromosome. The areas where "Two chromosomes shared" is indicated are regions where the individuals share DNA on both chromosomes in the pair (i.e., the individuals inherited the same DNA from their father and mother for those regions). Note that the regions where DNA is shared on both chromosomes is a subset of the regions where one chromosome is shared.

2.3.2 shared_dna_one_chrom_<name1>_<name2>_<cM_threshold>cM_<snp_threshold>snps_

If DNA is shared on one chromosome, a CSV file details the shared segments of DNA.

Column	Description
segment	Shared DNA segment number
chrom	Chromosome with matching DNA segment
start	Start position of matching DNA segment
end	End position of matching DNA segment
cMs	CentiMorgans of matching DNA segment
snps	Number of SNPs in matching DNA segment

2.3.3 shared_dna_two_chroms_<name1>_<name2>_<cM_threshold>cM_<snp_threshold>snps

If DNA is shared on two chromosomes, a CSV file details the shared segments of DNA.

Column	Description
segment	Shared DNA segment number
chrom	Pair of chromosomes with matching DNA segment
start	Start position of matching DNA segment on each chromosome
end	End position of matching DNA segment on each chromosome
cMs	CentiMorgans of matching DNA segment on each chromosome
snps	Number of SNPs in matching DNA segment on each chromosome

2.4 Find Shared Genes

Shared genes (with the *same genetic variations*) between two or more individuals can be identified with *find_shared_dna()*, with the parameter `shared_genes=True`. In addition to the outputs produced by *Find Shared DNA*, up to two additional CSV files are output that detail the shared genes when `save_output=True`.

In the filenames below, `name1` is the name of the first *Individual* and `name2` is the name of the second *Individual*. (If more individuals are compared, all *Individual* names will be included in the filenames using the same convention.)

2.4.1 shared_genes_one_chrom_<name1>_<name2>_<cM_threshold>cM_<snp_threshold>snp

If DNA is shared on one chromosome, this file details the genes shared between the individuals on at least one chromosome; these genes are located in the shared DNA segments specified in *shared_dna_one_chrom_<name1>_<name2>_<cM_threshold>cM_<snp_threshold>snps_GRCh37_<genetic_map>.csv*.

Column*	Description*
name	Name of gene
geneSymbol	Gene symbol
chrom	Reference sequence chromosome or scaffold
strand	+ or - for strand
txStart	Transcription start position (or end position for minus strand item)
txEnd	Transcription end position (or start position for minus strand item)
refseq	RefSeq ID
proteinID	UniProt display ID, UniProt accession, or RefSeq protein ID
description	Description

* UCSC Genome Browser / UCSC Table Browser

2.4.2 shared_genes_two_chroms_<name1>_<name2>_<cM_threshold>cM_<snp_threshold>sn

If DNA is shared on both chromosomes in a pair, this file details the genes shared between the individuals on both chromosomes; these genes are located in the shared DNA segments specified in *shared_dna_two_chroms_<name1>_<name2>_<cM_threshold>cM_<snp_threshold>snps_GRCh37_<genetic_map>.csv*.

The file has the same columns as *shared_genes_one_chrom_<name1>_<name2>_<cM_threshold>cM_<snp_threshold>snps_GRCh37*

CHAPTER

THREE

INSTALLATION

`lineage` is available on the Python Package Index. Install `lineage` (and its required Python dependencies) via `pip`:

```
$ pip install lineage
```

3.1 Installation and Usage on a Raspberry Pi

The instructions below provide the steps to install `lineage` 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 `lineage` 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 `lineage`.

4. Install Pillow dependencies:

   ```
   pi@raspberrypi:~ $ sudo apt install libjbig0 liblcms2-2 libopenjp2-7 libtiff5␣
   ↪libwebp6 libwebpdemux2 libwebpmux3
   ```

   Press "y" followed by "enter" to continue. This is required for Matplotlib, a dependency of `lineage`.

5. Create a directory for `lineage` and change working directory:

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

6. Create a virtual environment for `lineage`:

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

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

7. Activate the virtual environment:

```
pi@raspberrypi:~/lineage $ 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:~/lineage $ which python
/home/pi/lineage/.venv/bin/python
```

8. Install `lineage`:

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

9. Start Python:

```
(.venv) pi@raspberrypi:~/lineage $ 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.
>>>
```

10. Use `lineage`; examples shown in the README should now work.

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

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

3.2 Installation on Linux

On Linux systems, the following system-level installs may also be required:

```
$ sudo apt install python3-tk
$ sudo apt install gfortran
$ sudo apt install python-dev
$ sudo apt install python-devel
$ sudo apt install python3.X-dev # (where X == Python minor version)
```

CHAPTER

FOUR

CHANGELOG

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

CHAPTER

FIVE

CONTRIBUTING

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

5.1 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.

5.2 Documentation improvements

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

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/apriha/lineage/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 :)

5.4 Development

To set up `lineage` for local development:

1. Fork lineage (look for the "Fork" button).
2. Clone your fork locally:

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

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

```
$ cd lineage
$ 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=lineage 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.

5.4.1 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.

5.5 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

SIX

CONTRIBUTORS

Contributors to `lineage` are listed below.

6.1 Core Developers

Name	GitHub
Andrew Riha	@apriha

6.2 Other Contributors

Listed in alphabetical order.

Name	GitHub
Anatoli Babenia	@abitrolly
Kevin Arvai	@arvkevi
Will Jones	@willgdjones
Yoan Bouzin	

CHAPTER

SEVEN

CODE DOCUMENTATION

7.1 lineage

lineage

tools for genetic genealogy and the analysis of consumer DNA test results

class lineage.**Lineage**(*output_dir='output'*, *resources_dir='resources'*, *parallelize=False*, *processes=2*)

Bases: `object`

Object used to interact with the *lineage* framework.

__init__(*output_dir='output'*, *resources_dir='resources'*, *parallelize=False*, *processes=2*)

Initialize a `Lineage` object.

Parameters

- **output_dir** (*str*) – name / path of output directory
- **resources_dir** (*str*) – name / path of resources directory
- **parallelize** (*bool*) – utilize multiprocessing to speedup calculations
- **processes** (*int*) – processes to launch if multiprocessing

create_individual(*name*, *raw_data=()*, ***kwargs*)

Initialize an individual in the context of the *lineage* framework.

Parameters

- **name** (*str*) – name of the individual
- **raw_data** (str, bytes, `SNPs` (or list or tuple thereof)) – path(s) to file(s), bytes, or `SNPs` object(s) with raw genotype data
- ****kwargs** – parameters to `snps.SNPs` and/or `snps.SNPs.merge`

Returns

`Individual` initialized in the context of the *lineage* framework

Return type

Individual

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

`find_discordant_snps`(*individual1*, *individual2*, *individual3=None*, *save_output=False*)

Find discordant SNPs between two or three individuals.

Parameters

- **individual1** (*Individual*) – reference individual (child if *individual2* and *individual3* are parents)
- **individual2** (*Individual*) – comparison individual
- **individual3** (*Individual*) – other parent if *individual1* is child and *individual2* is a parent
- **save_output** (*bool*) – specifies whether to save output to a CSV file in the output directory

Returns
discordant SNPs and associated genetic data

Return type
pandas.DataFrame

References

1. David Pike, "Search for Discordant SNPs in Parent-Child Raw Data Files," David Pike's Utilities, http://www.math.mun.ca/~dapike/FF23utils/pair-discord.php
2. David Pike, "Search for Discordant SNPs when given data for child and both parents," David Pike's Utilities, http://www.math.mun.ca/~dapike/FF23utils/trio-discord.php

`find_shared_dna`(*individuals=()*, *cM_threshold=0.75*, *snp_threshold=1100*, *shared_genes=False*, *save_output=True*, *genetic_map='HapMap2'*)

Find the shared DNA between individuals.

Computes the genetic distance in centiMorgans (cMs) between SNPs using the specified genetic map. Applies thresholds to determine the shared DNA. Plots shared DNA. Optionally determines shared genes (i.e., genes transcribed from the shared DNA).

All output is saved to the output directory as *CSV* or *PNG* files.

Notes

The code is commented throughout to help describe the algorithm and its operation.

To summarize, the algorithm first computes the genetic distance in cMs between SNPs common to all individuals using the specified genetic map.

Then, individuals are compared for whether they share one or two alleles for each SNP in common; in this manner, where all individuals share one chromosome, for example, there will be several SNPs in a row where at least one allele is shared between individuals for each SNP. The `cM_threshold` is then applied

to each of these "matching segments" to determine whether the segment could be a potential shared DNA segment (i.e., whether each segment has a cM value greater than the threshold).

The matching segments that passed the `cM_threshold` are then checked to see if they are adjacent to another matching segment, and if so, the segments are stitched together, and the single SNP separating the segments is flagged as potentially discrepant. (This means that multiple smaller matching segments passing the `cM_threshold` could be stitched, identifying the SNP between each segment as discrepant.)

Next, the `snp_threshold` is applied to each segment to ensure there are enough SNPs in the segment and the segment is not only a few SNPs in a region with a high recombination rate; for each segment that passes this test, we have a segment of shared DNA, and the total cMs for this segment are computed.

Finally, discrepant SNPs are checked to ensure that only SNPs internal to a shared DNA segment are reported as discrepant (i.e., don't report a discrepant SNP if it was part of a segment that didn't pass the `snp_threshold`). Currently, no action other than reporting is taken on discrepant SNPs.

Parameters

- **individuals** (*iterable of Individuals*)
- **cM_threshold** (*float*) – minimum centiMorgans for each shared DNA segment
- **snp_threshold** (*int*) – minimum SNPs for each shared DNA segment
- **shared_genes** (*bool*) – determine shared genes
- **save_output** (*bool*) – specifies whether to save output files in the output directory
- **genetic_map** (*{'HapMap2', 'ACB', 'ASW', 'CDX', 'CEU', 'CHB', 'CHS', 'CLM', 'FIN', 'GBR', 'GIH', 'IBS', 'JPT', 'KHV', 'LWK', 'MKK', 'MXL', 'PEL', 'PUR', 'TSI', 'YRI'}*) – genetic map to use for computation of shared DNA; *HapMap2* corresponds to the HapMap Phase II genetic map from the International HapMap Project and all others correspond to the population-specific genetic maps generated from the 1000 Genomes Project phased OMNI data. Note that shared DNA is not computed on the X chromosome with the 1000 Genomes Project genetic maps since the X chromosome is not included in these genetic maps.

Returns

dict with the following items:

one_chrom_shared_dna (pandas.DataFrame)
segments of shared DNA on one chromosome

two_chrom_shared_dna (pandas.DataFrame)
segments of shared DNA on two chromosomes

one_chrom_shared_genes (pandas.DataFrame)
shared genes on one chromosome

two_chrom_shared_genes (pandas.DataFrame)
shared genes on two chromosomes

one_chrom_discrepant_snps (pandas.Index)
discrepant SNPs discovered while finding shared DNA on one chromosome

two_chrom_discrepant_snps (pandas.Index)
discrepant SNPs discovered while finding shared DNA on two chromosomes

Return type
dict

7.2 lineage.individual

Class for representing individuals within the *lineage* framework.

class lineage.individual.**Individual**(*name*, *raw_data=()*, ***kwargs*)

Bases: SNPs

Object used to represent and interact with an individual.

The Individual object maintains information about an individual. The object provides methods for loading an individual's genetic data (SNPs) and normalizing it for use with the *lineage* framework.

Individual inherits from snps.SNPs. See here for details about the SNPs object: https://snps.readthedocs.io/en/latest/snps.html

__init__(*name*, *raw_data=()*, ***kwargs*)

Initialize an Individual object.

Parameters

- **name** (*str*) – name of the individual
- **raw_data** (str, bytes, SNPs (or list or tuple thereof)) – path(s) to file(s), bytes, or SNPs object(s) with raw genotype data
- ****kwargs** – parameters to snps.SNPs and/or snps.SNPs.merge

get_var_name()

property name

Get this Individual's name.

Return type
str

7.3 lineage.resources

Class for downloading and loading required external resources.

lineage uses tables and data from UCSC's Genome Browser:

- http://genome.ucsc.edu/
- http://genome.ucsc.edu/cgi-bin/hgTables

References

1. Karolchik D, Hinrichs AS, Furey TS, Roskin KM, Sugnet CW, Haussler D, Kent WJ. The UCSC Table Browser data retrieval tool. Nucleic Acids Res. 2004 Jan 1;32(Database issue):D493-6. PubMed PMID: 14681465; PubMed Central PMCID: PMC308837. https://www.ncbi.nlm.nih.gov/pubmed/14681465
2. Tyner C, Barber GP, Casper J, Clawson H, Diekhans M, Eisenhart C, Fischer CM, Gibson D, Gonzalez JN, Guruvadoo L, Haeussler M, Heitner S, Hinrichs AS, Karolchik D, Lee BT, Lee CM, Nejad P, Raney BJ, Rosenbloom KR, Speir ML, Villarreal C, Vivian J, Zweig AS, Haussler D, Kuhn RM, Kent WJ. The UCSC Genome Browser database: 2017 update. Nucleic Acids Res. 2017 Jan 4;45(D1):D626-D634. doi: 10.1093/nar/gkw1134. Epub 2016 Nov 29. PubMed PMID: 27899642; PubMed Central PMCID: PMC5210591. https://www.ncbi.nlm.nih.gov/pubmed/27899642

3. 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
4. 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
5. Yates et. al. (doi:10.1093/bioinformatics/btu613), http://europepmc.org/search/?query=DOI:10.1093/bioinformatics/btu613
6. Zerbino et. al. (doi.org/10.1093/nar/gkx1098), https://doi.org/10.1093/nar/gkx1098

class lineage.resources.**Resources**(**args*, ***kwargs*)

Bases: Resources

Object used to manage resources required by *lineage*.

__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 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_resources()

Get / download all resources (except reference sequences) used throughout *lineage*.

Returns
dict of resources

Return type
dict

get_cytoBand_hg19()

Get UCSC cytoBand table for Build 37.

Returns
cytoBand table if loading was successful, else empty DataFrame

Return type
pandas.DataFrame

References

1. Ryan Dale, GitHub Gist, https://gist.github.com/daler/c98fc410282d7570efc3#file-ideograms-py

`get_genetic_map`(*genetic_map*)

Get specified genetic map.

Parameters

genetic_map (*{'HapMap2', 'ACB', 'ASW', 'CDX', 'CEU', 'CHB', 'CHS', 'CLM', 'FIN', 'GBR', 'GIH', 'IBS', 'JPT', 'KHV', 'LWK', 'MKK', 'MXL', 'PEL', 'PUR', 'TSI', 'YRI'}*) – *HapMap2* corresponds to the HapMap Phase II genetic map from the International HapMap Project and all others correspond to the population-specific genetic maps generated from the 1000 Genomes Project phased OMNI data.

Returns

dict of pandas.DataFrame genetic maps if loading was successful, else {}

Return type

dict

`get_genetic_map_1000G_GRCh37`(*pop*)

Get population-specific 1000 Genomes Project genetic map.[1][2]

Notes

From README_omni_recombination_20130507[1] :

Genetic maps generated from the 1000G phased OMNI data.

[Build 37] OMNI haplotypes were obtained from the Phase 1 dataset (/vol1/ftp/phase1/analysis_results/supporting/omni_haplotypes/).

Genetic maps were generated for each population separately using LDhat (http://ldhat.sourceforge.net/). Haplotypes were split into 2000 SNP windows with an overlap of 200 SNPs between each window. The recombination rate was estimated for each window independently, using a block penalty of 5 for a total of 22.5 million iterations with a sample being taken from the MCMC chain every 15,000 iterations. The first 7.5 million iterations were discarded as burn in. Once rates were estimated, windows were merged by splicing the estimates at the mid-point of the overlapping regions.

LDhat estimates the population genetic recombination rate, rho = 4Ner. In order to convert to per-generation rates (measured in cM/Mb), the LDhat rates were compared to pedigree-based rates from Kong et al. (2010). Specifically, rates were binned at the 5Mb scale, and a linear regression performed between the two datasets. The gradient of this line gives an estimate of 4Ne, allowing the population based rates to be converted to per-generation rates.

Returns

dict of pandas.DataFrame population-specific 1000 Genomes Project genetic maps if loading was successful, else {}

Return type

dict

[1] Adam Auton, May 7th, 2013

[2] The 1000 Genomes Project Consortium., Corresponding authors., Auton, A. et al. A global reference for human genetic variation. Nature 526, 68–74 (2015). https://doi.org/10.1038/nature15393

References

get_genetic_map_HapMapII_GRCh37()

Get International HapMap Consortium HapMap Phase II genetic map for Build 37.[4][5]

Returns
dict of pandas.DataFrame HapMapII genetic maps if loading was successful, else {}

Return type
dict

References

get_kgXref_hg19()

Get UCSC kgXref table for Build 37.

Returns
kgXref table if loading was successful, else empty DataFrame

Return type
pandas.DataFrame

get_knownGene_hg19()

Get UCSC knownGene table for Build 37.

Returns
knownGene table if loading was successful, else empty DataFrame

Return type
pandas.DataFrame

7.4 lineage.visualization

Chromosome plotting functions.

Notes

Adapted from Ryan Dale's GitHub Gist for plotting chromosome features.[6]

[4] "The International HapMap Consortium (2007). A second generation human haplotype map of over 3.1 million SNPs. Nature 449: 851-861."
[5] "The map was generated by lifting the HapMap Phase II genetic map from build 35 to GRCh37. The original map was generated using LDhat as described in the 2007 HapMap paper (Nature, 18th Sept 2007). The conversion from b35 to GRCh37 was achieved using the UCSC liftOver tool. Adam Auton, 08/12/2010"
[6] Ryan Dale, GitHub Gist, https://gist.github.com/daler/c98fc410282d7570efc3#file-ideograms-py

References

lineage.visualization.**plot_chromosomes**(*one_chrom_match*, *two_chrom_match*, *cytobands*, *path*, *title*, *build*)

Plots chromosomes with designated markers.

Parameters

- **one_chrom_match** (*pandas.DataFrame*) – segments to highlight on the chromosomes representing one shared chromosome
- **two_chrom_match** (*pandas.DataFrame*) – segments to highlight on the chromosomes representing two shared chromosomes
- **cytobands** (*pandas.DataFrame*) – cytobands table loaded with Resources
- **path** (*str*) – path to destination *.png* file
- **title** (*str*) – title for plot
- **build** (*{37}*) – human genome build

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

l

INDEX

Symbols

C

D

F

G

I

L

M

N

P

R