POPgen

Search Engine

POPgen is a user-friendly website that allows the user to retrieve SNP information for a genomic region of interest. Summary statistics can be calculated, and their values can be plotted in a sliding window graph.

Produced by Waleed, Laavanya, Mani, Alexandra, Harry

Documentation

Version 1 – March 2022

Contents

Overview

Software Schematic

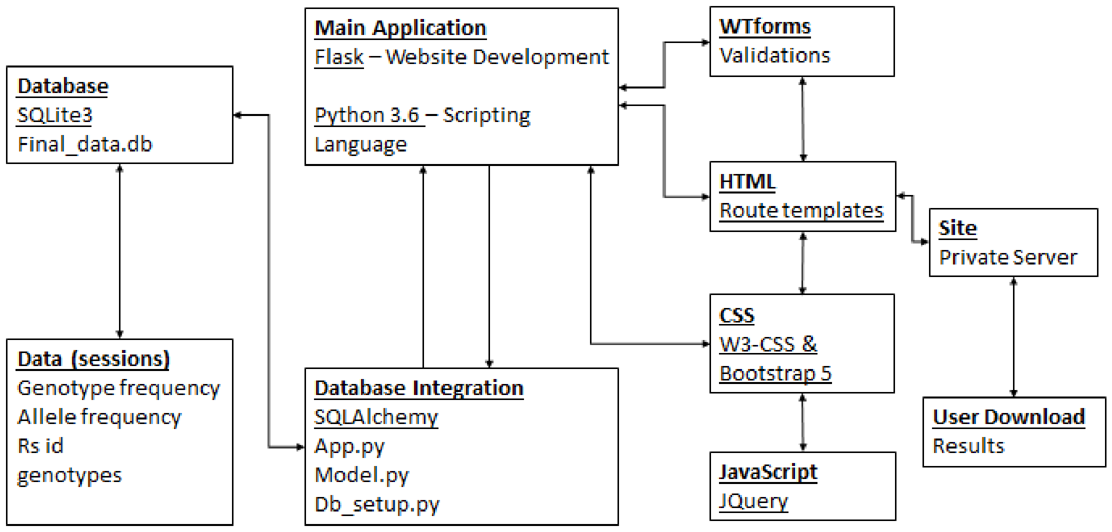


Figure 1. POPgen software schematic and component integration

Figure 1 shows the finalized software structure and all the integrated components used to create the functional POPgen web application, which can search through SNP information within human chromosome 22. This SNP information was downloaded from the IGSR web application and constructed into a functional database using SQLite3, which was then linked to flask using the package SQLAlchemy. SQLAlchemy facilitates communication between the flask python program and the constructed relational SQLite3 database in python. The main application runs on Flask using Jinja2 as a template engine to create HTML and other markup formats which can then be returned to the user in a http response (routes.py is the executable file). Website routes were defined using the HTML language and static files such as CSS (maintain consistency in website design). Together the application renders the final site that also features a user download for the results of their search parameters.

Website Architecture

Figure 2. shows a summary of all the routes within the POPgen website and how they are linked. The Search page outputs the results page from user-input fields, such as the gene name, RS ID and Chromosome location. The results page then outputs the relevant data in tables and provides options to view the summary statistics within and between populations. Once the statistics options within the results page have been selected and searched, the statistics page will show the relevant statistics for the selected populations available within the database.

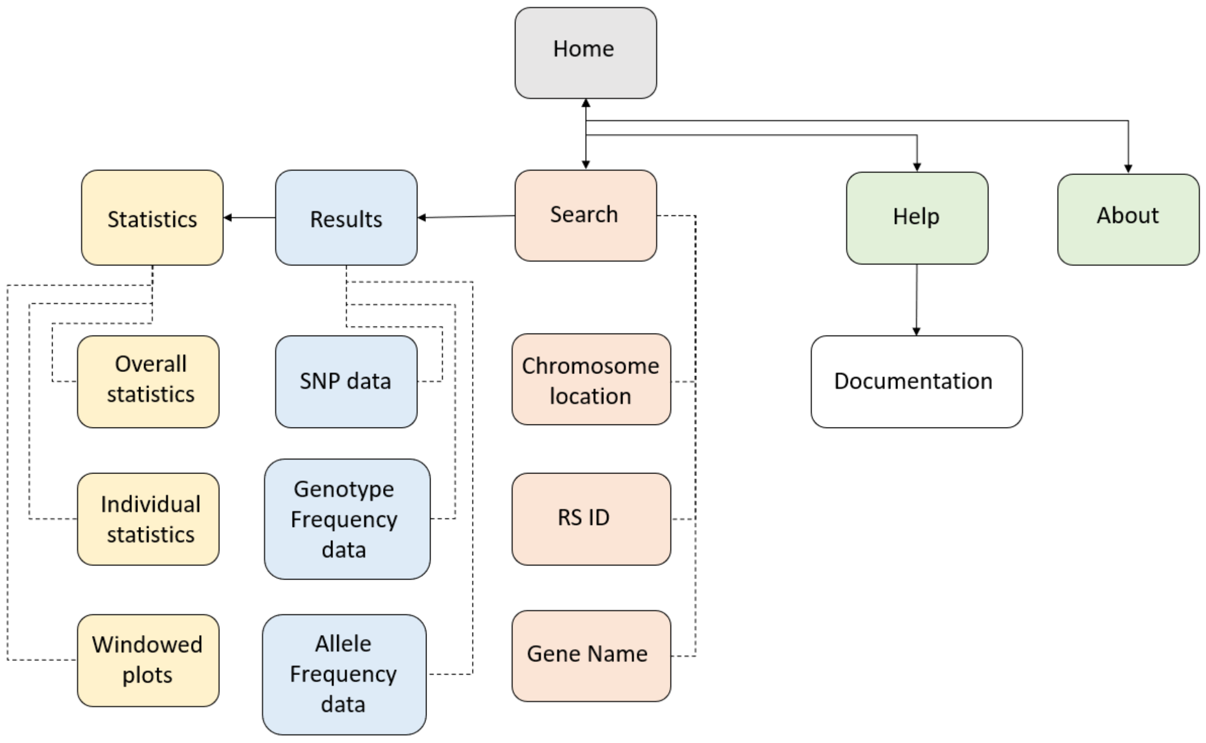


Figure 2. POPgen web routes and links shown by solid black arrows. Web content shown by dotted lines.

Running POPgen

How to run

POPgen has been deployed to the web successfully and the link can be found in the GitHub page.

However, users can run POPgen locally. For POPgen to run, cloning the Web_Project repository from GitHub is needed.

POPgen can run locally using:

```
$ git clone https://github.com/Mani-varma1/Web_project
```

```
$ cd Flask
```

Virtual environment can be set up using:

```
$ py -3 -m venv .venv
$ .venv/Scripts/activate
```

To install necessary packages:

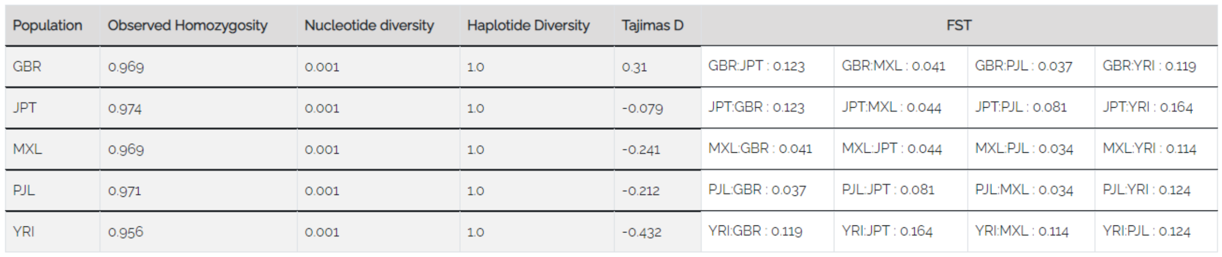
```
$ pip install -r packages.txt
```

A script for populating the database has been provided for those that wish to set up a local database and wish to explore the dataset. The files required for generating the database have also been provided in the form of a zip file with an empty database. All that needs to be done is to run the database_populating.py script in the folder it is located in. This will extract the files to a new data folder and automatically populate the database. The script takes roughly 30 minutes to run.

Finally, to run the application:
$ python application.py

Walkthrough

The website lets you search by chromosome name and position (start-end), RS ID, gene name. It outputs SNPs information (chromosomal position, reference allele, alternate allele, frequencies) and allows you to select the desired statistics for calculation, based on the 5 populations provided. The populations can be selected as well. The statistics page contains the results of the calculated statistics (Table 1), as well as sliding windowed plots (Figure 3), of all statistics for better visualization and user experience.



Population	Observed Homozygosity	Nucleotide diversity	Haplotide Diversity	Tajimas D	FST			
GBR	0.969	0.001	1.0	0.31	GBR:JPT : 0.123	GBR:MXL : 0.041	GBR:PJL : 0.037	GBR:YRI : 0.119
JPT	0.974	0.001	1.0	-0.079	JPT:GBR : 0.123	JPT:MXL : 0.044	JPT:PJL : 0.081	JPT:YRI : 0.164
MXL	0.969	0.001	1.0	-0.241	MXL:GBR : 0.041	MXL:JPT : 0.044	MXL:PJL : 0.034	MXL:YRI : 0.114
PJL	0.971	0.001	1.0	-0.212	PJL:GBR : 0.037	PJL:JPT : 0.081	PJL:MXL : 0.034	PJL:YRI : 0.124
YRI	0.956	0.001	1.0	-0.432	YRI:GBR : 0.119	YRI:JPT : 0.164	YRI:MXL : 0.114	YRI:PJL : 0.124

Table 1. Output of calculation of all statistics and all populations.

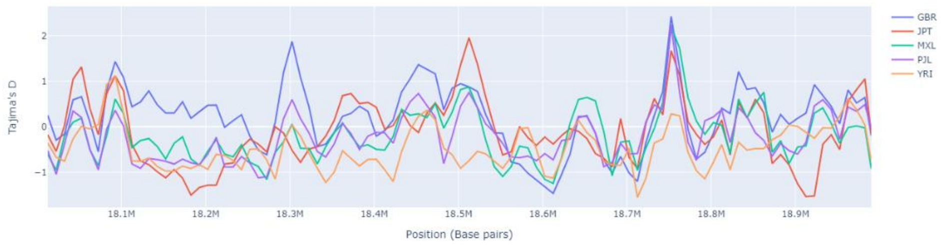


Figure 3. An example of a sliding plot for Tajima's D using a search region of 18,000,000-19,000,000 for 5 populations, window size = 25,000 step size = 10,000.

Technologies

Flask

Flask is a micro web framework written in python for creating a quick and easy web application, implemented on Werkzeug, a Web Server Gateway Interface (WSGI) app that allows requests to be forwarded from the web server and responses to be passed back. Flask also uses Jinja2 templating engine for a simple yet robust use of python code in HTML. This allows HTML to use python related functions such as logical operators like if statements or for loops, represented by "{% code %}", and also call-in assigned variables, represented by "{{ variable }}". Python syntax can be seamlessly integrated into HTML and CSS for a more dynamic and responsive website. Generating HTML code in python code can be challenging and error prone especially for larger HTML documents. Flask uses render template method to call in the HTML files to be used for their respective routes. Creation of multiple routes is as simple as binding a function with a route decorator when flask class is instantiated.

HTML, CSS and JavaScript

HTML and CSS are the core elements of the frontend allowing for a well-organized and easy to understand user interface. HTML Document Object Models (DOM) were coupled with Bootstrap and W3-CSS libraries for creating a responsive and polished website. These libraries contain prebuilt custom classes for professional grade styling of our website. Although JavaScript is a core component of any modern website, we have used pre-existing libraries like JQuery to improve the interactivity of our website.

Forms and Validation

We often require input data types to be in specific format for the downstream analysis to work. User input can often be unpredictable and error prone. It is crucial to provide the user with appropriate feedback when mistakes are made. Although HTML has such attributes, they are limited in nature without extensive use of JavaScript. Extensions such as Flask WTforms, allow developers to easily create forms in an object-oriented way, which can be passed as a variable to the necessary templates. Setting up flask forms is extremely simple. The main advantages of using WTforms is their data validation, which allows the web server to check the input data and is within the constraints set by the application, and cross site request forgery form (CSRF) protection, which prevents malicious attacker to induce users to perform actions that they do not intend to perform. We can also easily create our own custom validation checks that are specific to our website forms without JavaScript. Having form validation can be coupled to CSS with Jinja2 to provide visual feedback to the users on their input errors, which can significantly improve the user experience and allows developers to focus more on the downstream analysis without formatting issues.

Sessions

Web apps often require data to be stored between each request as a user interacts with the app. However, HTTP is a stateless protocol and has no knowledge of other requests that were executed. This data could be stored in cookies which are stored on the user's computer. Due to security concerns and storage limit of 4KB, sessions are often used. Sessions are files that contain information on non-user specific requests they have executed. This allows us to access this data in multiple routes, without having to request user input for the same data on different pages. We use sessions to store information that was retrieved from queries which include SNP information and population specific frequencies. Prior to storage we must first serialize the data as session can only be stored in JSON format. Based on the user input parameters, the data size retrieved from the database could exceed 4KB. Therefore, we decided to store the sessions on the file systems, with no limits mentioned in the documentation. These sessions will automatically clear when either the browser is closed or replaced with data from a new query search if the user decides to change their search parameters. We can also have access to the session data within Jinja2 to create dynamic route links. Sessions can be used by JavaScript to perform various tasks in the front end without relying on the backend. Having sessions allows the user to quickly load the data if they close the tab by accident or navigate around the website, without having to search the database again.

Data Compression

Due to the amount of data within the VCF file it is not practical to store the raw genotype data for everyone as shown in VCF format. Doing so could significantly increase the storage requirements and deteriorate the querying speed. The repetitive nature of our data means there is lot of redundant data that violates the first normal form of databases, so we have decided to compress the data. Although many compressions technologies exist, we took inspiration from Run length encoding (RLE) to create a custom compression method which retains the positions of the sample and minimises the data storage requirement. Allowing us to be more flexible in the future if we decide to include other statistical methods.

Structure and Design

The structure of the website was defined using HTML tags/classes from the stylesheets of both Bootstrap and W3-CSS for each page within the website architecture. These HTML templates were coupled with CSS languages to produce a professional website design that provides a simple and user-friendly experience. The Bootstrap and W3-CSS tags create a framework for responsive elements to the website and help organize where each HTML component will appear on the website. For example, Bootstrap classes were used to center the team information on the about page.

Deployment

Web Deployment

Elastic Beanstalk was used from the Amazon Web Services (AWS).

Data

Data Collection

The data used to populate the database was primarily retrieved from the 1000 Genomes Project and the NCBI FTP server. The data used for this project was sourced from phase 3 data collection and this was done for two primary reasons:

- Phase three data contained information about both indels and multiallelic SNPs, which can be filtered to create a dataset only containing biallelic SNPs. Earlier phases of the project were not able to distinguish and identify multiallelic SNPs and would record them as separate entries within the VCF files (1KGP consortium, 2015).

- Phase three data used a larger sample size with more populations included.

Phase three data used genomic positions based on the ensembl GrCh37.p19 reference build. This is important to note as the current RS ID's found on the dbSNP from ensembl have positions that correspond to both the GrCH37.p19 and the newer GrCH38.p19 reference build. The reasoning for choosing the GrCH37 build is explained in the Data Wrangling section.

Sample data was also required to allow sub-setting based on population, which was also obtained from 1KGP from the respective data collection page.

Information regarding RS ID corresponding to position as well as gene consequence can be obtained from NCBI (Sherry et al, 2001). A VCF for the entire genome would need to be downloaded and then filtered. These files can be found here:

https://ftp.ncbi.nih.gov/snp/latest_release/VCF/.

Both GrCH37 (GCF_000001405.25) and GrCH38 (GCF_000001405.39) builds can be found with their corresponding tabix files. The chromosome fields are populated according RefSeq annotation.

Data regarding gene names which corresponds to RS IDs can be obtained through several ways. NCBI provides a command line tool which can retrieve information from dbSNP using their API (Kans, 2022). This is limited to 3 requests per second per IP address. The command used to retrieve gene names is shown below:

```
<code>for i in `cat x`; do esearch -db snp -query "${i} AND human [orgn]" | esummary | xtract -pattern
DocumentSummary -element SNP_ID,NAME | uniq; done > output.txt</code>
```

Data Wrangling & Pre-processing

To keep storage requirements low, the database was constructed using VCF data from chromosome 22 for five populations: Mexican (MXL), British (GBR), Japanese (JPT), Punjabi (PJL), and Yoruba (YRI). Each of these populations belong to separate superpopulations, therefore information regarding superpopulations was left out as it would make the search options redundant for this build.

Bcftools was used to subset only for biallelic SNPs and the 5 populations. The VCF files available from 1KGP contain empty fields for ID. To populate the database correctly, these would need to be filled. The following command from bcftools was used assuming we have a VCF file with annotated IDs and positions according to the same reference build:

```
<code>bcftools annotate -a 'path/to/dbsnp.vcf.gz' -c ID 'path/to/1KGP.vcf.gz' -o
'path/to/output.vcf.gz </code>
```

The reference VCF would need to be first filtered for only the chromosome of interest, which can also be done in bcftools.

Some positions found on the VCF file which had previously been associated with a RS ID were no longer supported as of the current dbSNP build. As RS ID is required for the database, these SNPs were filtered from the final VCF file. This has led to gaps within the chromosome region, which may affect expected statistical values when searching these regions. These regions are:

1. 20706257-20708536
2. 25722272-25742171

To get the data into a format that would make it easier to populate the database, the scikit-allel package in python was used (Miles & Harding, 2021). This has the advantage of using local storage array solutions to reduce load on memory usage. It also has many features to subset data and read it into custom array-like structures that work in a similar fashion to numpy arrays. The script used to wrangle this data can be found on the GitHub repository.

This script will produce 6 tsv files; one query file and 5 population files. The population data contains genotype arrays and pre-calculated counts for genotypes and alleles per SNP.

The query file requires further wrangling to get gene names inputted. A txt file containing gene names and RS IDs were then loaded into python as pandas dataframes along with the query file and merged on RS ID to produce a final query file. This file is provided along with population data, which can be used to create the database.

Data Storage

SQLite3 and SQLAlchemy were used to create and populate the database. Structures for the tables can be found in the models.py file. All search queries are initially made with the query table, which contains information about the chromosomal position, reference allele, alternate allele, and, where it is applicable, any genes that are implicated. Due to the unique property of RS IDs from the dbSNP, these were used as primary keys for which any searches would need to be matched to. The results page outputs the data stored within the query _search table (as seen in Figure 3). The other tables correspond to each population and contains information about the genotype frequency, allele frequency and phased genotype arrays for each SNP, with RS ID acting as a foreign key referencing the query_ search table. Both genotype frequency and allele frequency are stored in dictionary format within each of these fields.

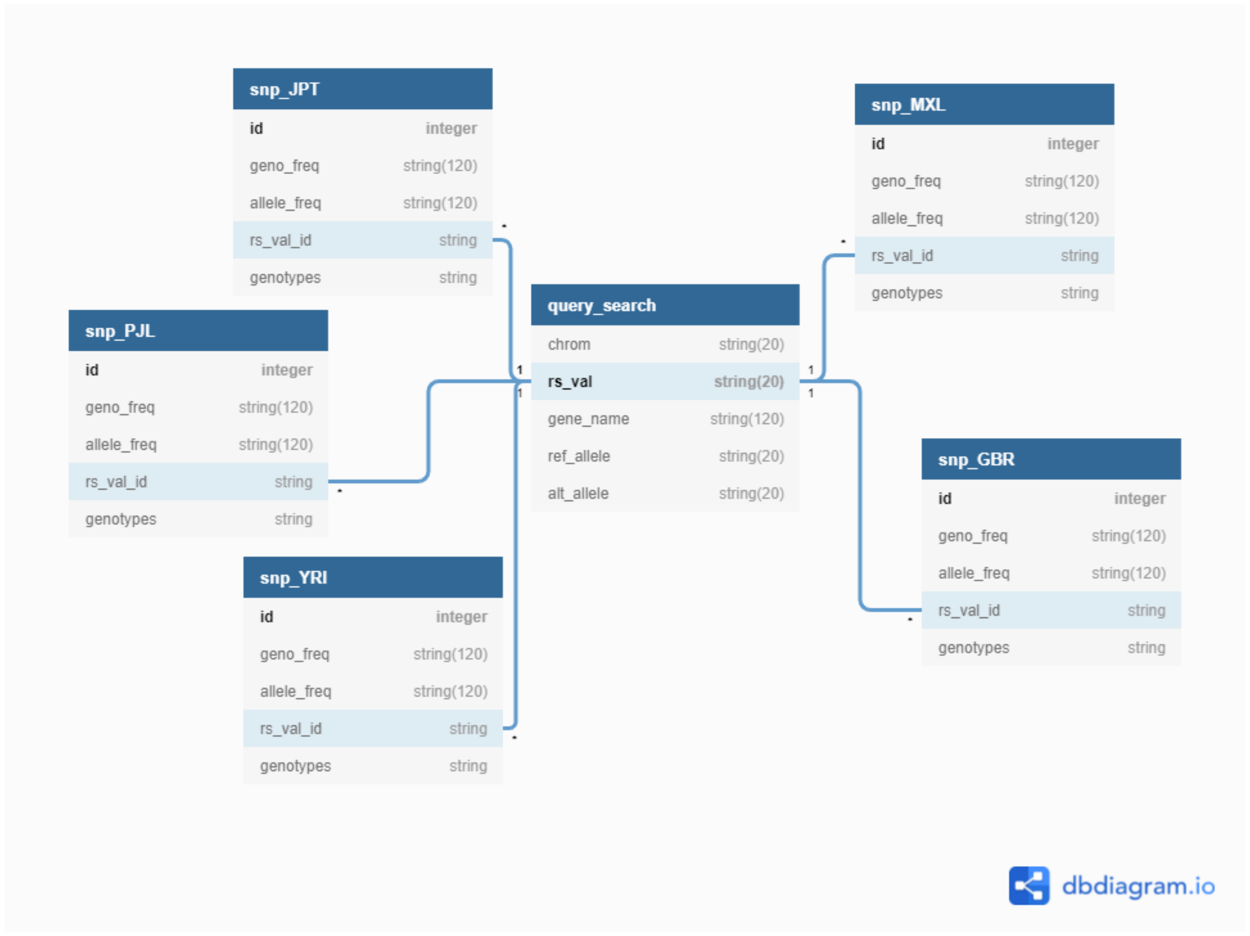


Figure 4. POPgen database schematic: Shown is 6 tables and the links represented by solid lines

<u>Search</u>

Querying was done through SQLAlchemy. Searches can be made based on chromosomal positions, RS ID, and gene names (Using HGNC names). Individual SNPs can be queried by passing a single position in the start field or 1 RS ID. Multiple SNPs can be queried by providing a chromosomal region (start and stop positions) or a list of comma-separated RS IDs or gene names. Gene names have been limited to a maximum of five.

Queries must be submitted in the correct format (especially for multiple SNP searches) and in the case of gene names, are case sensitive.

NOTE: Large search queries can be made however this is dependent on hardware limitations. Significant slowdowns in page responsiveness can occur if 100,000s of SNPs are queried due to browser limitations. Keep this in mind when searching the database. On average, it takes ~1.5 minutes to run a search regardless of number of SNPs queried. This can take longer depending on how many SNPs need to be outputted to the page. It can take around 3 minutes to output a 500,000bp search.

Statistics

Summary Statistics

Statistics analyses are performed to see whether there are significant differences between and within populations, using SNP information and genomic regions. All but one statistical values are calculated through Scikit-allel (Miles & Harding, 2021).

Several summary statistics are available on POPgen. The user can select the summary statistics of interest, and values and plots will be outputted. The download button can be used to download summary statistics.

Window size and step size are determined by the user. Step size is optional, and an empty field will result in window plots with no overlaps. Choosing an appropriate window size with respect to the size of the region searched is important to obtain meaningful analysis.

Window sizes that are two small may result in NaN values due to insufficient number of SNPs in a window. The plots will still be produced, but with breaks where there are missing values. If this occurs, you may need to adjust the window and step size to reduce the occurrence of NaN values. Typically, a window size of 25,000 and step size of 10,000 is recommended for a search of a 100,000 bp region.

As mentioned previously, there are two regions that may cause unexpected NaNs; 20706257-20708536 and 25722272 – 25742171. These regions contain missing SNPs found in the original VCF due to unsupported RS ID. Care needs to be taken when searching these regions, especially the latter.

- Observed Homozygosity

This is a measure of the total sum of homozygous reference and homozygous alternative for each SNP. The value outputted indicates the proportion in which the SNP is homozygous, and a high value indicates the likelihood of the SNP being homozygous. This is based on a direct count method highlighted in Sabatti & Risch, 2002.

$$\widehat{H}_A^{\text{count}} = \frac{\text{No. homoz. genotypes}}{\text{No. genotypes}}$$

- Genetic Diversity

Genetic Diversity can be used to see whether there is variation between selected populations.

Nucleotide Diversity

Nucleotide Diversity (π) is used as a measure of genetic diversity on POPgen. This is defined as the average number of nucleotide differences per site when two sequences are chosen randomly from the sample population. The formula outputs π and a high π indicates high genetic variation.

$$\hat{\pi} = \frac{n}{n-1} \sum_{ij} x_i x_j \pi_{ij}$$

Here, x_i and x_j refer to the population frequency of the ith and jth nucleomorph, and n refer to sample size. π_{ij} refers to the differences in nucleotides between the ith and jth sequences (Nei & Li, 1979).

- Haplotype Diversity

Haplotype Diversity, also known as gene diversity, can be defined as the probability of two alleles randomly being different when sampled (de Jong *et al.*, 2011).

$$H = \frac{N}{N-1}(1 - \sum_i x_i^2)$$

The function estimates haplotype diversity (H), where x_i refers to the relative haplotype diversity in the sample and N refers to the sample size. A value close to 1 indicates that the probability of picking two haplotypes which are the same are extremely high. A value close to 0 indicates that there are lots of haplotypes indicating more variation.

- Neutrality test

Neutrality tests are used in population genetics to assess the goodness-of-fit of standard neutral model (Achaz, 2009) based on the frequency spectrum such as Tajima's or Fu and Li's F.

Tajima's D

Tajima's D (D) is used as a test against neutrality, which is calculated through genotype arrays and counting the alleles. The formula compares the difference in values of Watterson theta (ratio of number of SNPs observed to expected number of SNPs based on neutral model of evolution) and expected number of segregating sites (according to a neutral model of evaluation).

$$D = \frac{d}{\sqrt{\hat{V}(d)}}$$

Interpreting D

$D = 0$: The observed and expected variation are similar. One conclusion is there is no evidence of selection for the observed sequence.

$D > 0$: Observed variation is greater than expected. This may indicate that the alternate alleles are scarce and that there may be a balancing selection.

$D < 0$: Observed variation is less than expected. This may indicate that the alternate alleles are abundant, indicating possibility of a selection pressure/selective sweep.

With single values, it may be hard to ascertain the meaning of these values, as these interpretations are only relevant if the value for D is statistically significant. The sliding window feature can obtain a range of values along an overall searched region, which can then be used to identify regions that may be significantly different from other observed values. A table of confidence limits for D can be found in the original published paper, which can be used to determine significance of individual observed values. (Tajima, 1989).

- Population genetic variation

F_{ST}

Fst is a measure of genetic variance which is explained by population structure. POPgen calculates the Fst values for each pair of populations the user has selected. The function is calculated through the method of Hudson (1992). Values range from 0 (low degree of differentiation) to 1 (High degree of differentiation).

$$\hat{F}_{ST}^{Hudson} = \frac{(\tilde{p}_1 - \tilde{p}_2)^2 - \frac{\tilde{p}_1(1-\tilde{p}_1)}{n_1 - 1} - \frac{\tilde{p}_2(1-\tilde{p}_2)}{n_2 - 1}}{\tilde{p}_1(1-\tilde{p}_2) + \tilde{p}_2(1-\tilde{p}_1)}$$

Here, P1 refers to sample allele frequency in the first population and P2 refers to sample allele frequency in the second population. The function used by POPgen produces the numerator (which defines the mean number of differences within a population) and the denominator (defines the mean number of differences between populations). The sum of the numerators is divided by the sum of the denominators to obtain an average value for Fst over all variants searched. This definition of Fst was chosen as it is independent of sample size and is recommended when doing pair-wise comparisons (Bhatia et al, 2013).

Plots

Sliding Window Plots

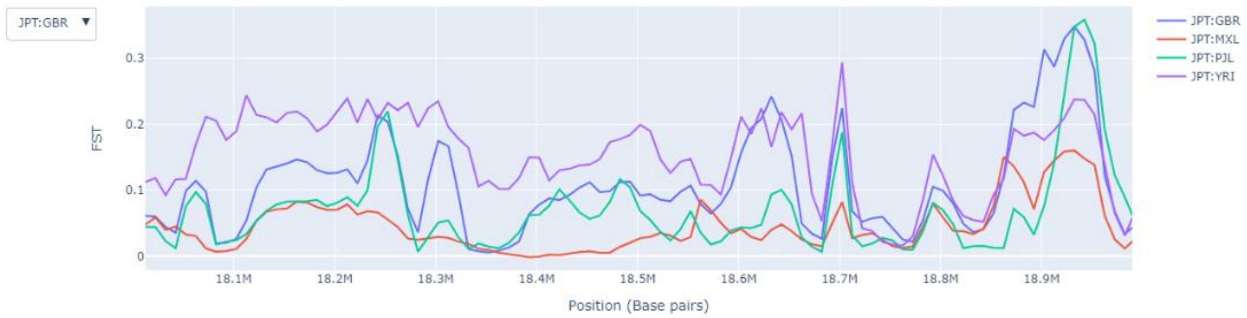


Figure 5. An example of a sliding plot for Fst using a search region of 18,000,000-19,000,000 for 5 populations, window size = 25,000 step size = 10,000.

A sliding window plot is created using a window of a fixed size which is sliding along the genomic region of interest. The sliding window will move in steps of a specified size from the beginning to end position. At each window, the statistical values are calculated and plotted against the average genomic position.

A sliding window plot is calculated and plotted using positions, population data, window sizes and step sizes. Window and step sizes are given in base pairs and can be specified by the user. The window size refers to the number of variants for haplotype diversity. Step size refers to the number of nucleotides for nucleotide diversity.

Plots are created using Plotly package (Nucleotide Diversity, Haplotype Diversity, Tajima's D and Fst). The different populations can be selected either using the lines to the side of the plot.

Limitations

Dataset Limitations

The dataset itself has issues, and no single dataset from the 1KGP was perfect. As the nature of this project involves using datasets that are constantly being updated themselves, there are some discrepancies with the final form of the dataset used. Recent dbSNP builds no longer supports certain SNPs that are found in the dataset used. To satisfy the requirements for the database, these SNPs would need to be removed during the data wrangling process. This will inevitably lead to regions in the VCF files that hold no SNPs. One potential workaround would be to include the non-supported RS IDs as they are still searchable on the dbSNP (the individual pages simply mention that its unsupported), however this would require mixing different builds for dbSNP to populate the missing field and using several reference files, to ensure that no positions were left unannotated. As bcftools was used to annotate the RS IDs, this can become complicated as it tends to override previous annotations.

Database Limitations

The format of the tables, regarding populations, is unnormalised. It would be best practice, especially with larger datasets spanning multiple chromosomes and more populations, to split the genotype and allele frequency into four or five separate columns and storing only 1 value in each field. This would allow the database to satisfy the first normal form (1NF). Another issue is regarding the formatting of phased genotype arrays. Initially, two additional tables were trialed, one table containing sample IDs and information regarding each sample, and another table containing genotype array data, with a column containing RS ID and sample ID. The issue with this approach was the number of rows that would need to be populated for a single chromosome. For chromosome 22 and 500 samples, this would equate to ~500,000,000 rows. For the purposes of this version of the application, keeping array data in the form that we used can simplify the populating of the database, but changes to individual genotypes would prove to be difficult.

Technologies Limitations

- Flask
  Flask is only viable for a small-scale application. More complex applications need support for more built-in packages and libraries. Consequently, there could be an issue when scaling the application. Django would be a good alternative option, as it offers everything Flask has and more, but with a steeper learning curve.
- HTML, CSS and JavaScript
  Most web apps use HTML, CSS and JS. Although we used some built in JS libraries, the use of these was very limited. There is a need for more custom JS. For a full control of our website, use JS with libraries such as react, which also allows us to build our own API and make the website more robust.
- Sessions
  Storage requirements could be an issue if multiple users access the database and search for large regions at the same time.

Future Developments

1. Expanding dataset to include entire human genome – significant storage requirements and data wrangling required. It took around 1 week to get the gene names for the smallest autosome. The entire dbSNP database would need to be accessed to generate gene names for each RS ID. Also, a more efficient method for wrangling the data would need to be employed as the current scripts can take up to 1 hour to go from VCF files to a populated database.
2. Expand to all populations from IGSR and allow for filtering by superpopulations. Would need to create additional tables for each population and use dictionaries to categorise populations by superpopulations.
3. More meaningful stats options (e.g. PCA) using genotype array data – would require restructuring of database to separate 1 genotype per field. Also, would require sample information table for information about sample ID, population and superpopulation.
4. Exploring options of non-relational databases that may be better suited for the type of data we are using. For example, multivalue databases or object databases could be used for data that was stored as lists or dictionaries.
5. Improvements to algorithms for better efficiency – Code relies on some nested for loops that can significantly increase memory usage and time complexity.
6. Cleaner code – streamline repetitive elements by using more classes and functions.

References

Achaz, G. (2009) 'Frequency spectrum neutrality tests: One for all and all for one', *Genetics*, 183(1), pp. 249–258. doi: 10.1534/genetics.109.104042.

de Jong, M. A. *et al.* (2011) 'Mitochondrial DNA signature for range-wide populations of Bicyclus anynana suggests a rapid expansion from recent refugia', *PLoS ONE*, 6(6), pp. 1–5. doi: 10.1371/journal.pone.0021385.

Nei, M., & Li, W. H. (1979). Mathematical model for studying genetic variation in terms of restriction endonucleases. Proceedings of the National Academy of Sciences, 76(10), 5269-5273.

Tajima, F. (1989). Statistical method for testing the neutral mutation hypothesis by DNA polymorphism. Genetics, 123(3), 585-595.

Sabatti, C., & Risch, N. (2002). Homozygosity and linkage disequilibrium. Genetics, 160(4), 1707-1719.

Bhatia G, Patterson N, Sankararaman S, Price AL. Estimating and interpreting FST: the impact of rare variants. Genome Res. 2013;23(9):1514-1521. doi:10.1101/gr.154831.113

Miles, A. & Harding, N. 2021. cggh/scikit-allel: v1.3.3 (Version 1.3.3). Zenodo. http://doi.org/10.5281/zenodo.822784

The 1000 Genomes Project Consortium. A global reference for human genetic variation. Nature 526, 68–74 (2015). https://doi.org/10.1038/nature15393

S. T. Sherry, M.-H. Ward, M. Kholodov, J. Baker, L. Phan, E. M. Smigielski, K. Sirotkin, dbSNP: the NCBI database of genetic variation, Nucleic Acids Research, Volume 29, Issue 1, 1 January 2001, Pages 308–311, https://doi.org/10.1093/nar/29.1.308

Kans, J. (2022). Entrez direct: E-utilities on the UNIX command line. In Entrez Programming Utilities Help [Internet]. National Center for Biotechnology Information (US).