Team Mint

SOFTWARE DEVELOPMENT GROUP PROJECT 2024

v0.1.0

Documentation

Amin Shire, Bishan Rai, Christina Yiangou, Namit Nagar, Peng Zhou

Intro to our app

This web application provides a user-friendly interface to explore the SNPs found on chromosome 1 in 3929 samples from different populations around the world. Specifically, it allows users to perform two population genetic structure analyses on the data set provided; clustering using Principal Component Analysis and Admixture analysis. In these two analyses the user is able to select which populations or superpopulations to analyse each time. Additionally, the user can use a search feature to retrieve allele and genotype frequencies as well as other relevant information, such as clinical information for the SNPs of interest in the populations of interest. The user also has the option to search for genes of interest in the populations he chooses and all SNPs associated with those genes along with the relevant information will be shown. This can also be done by choosing a region of interest on chromosome 1. If multiple populations are selected in the latter three analyses an Fst matrix is automatically created showing how much the selected populations differ and the user has the option to download the matrix as a text file to be used in further analyses.

Software architecture

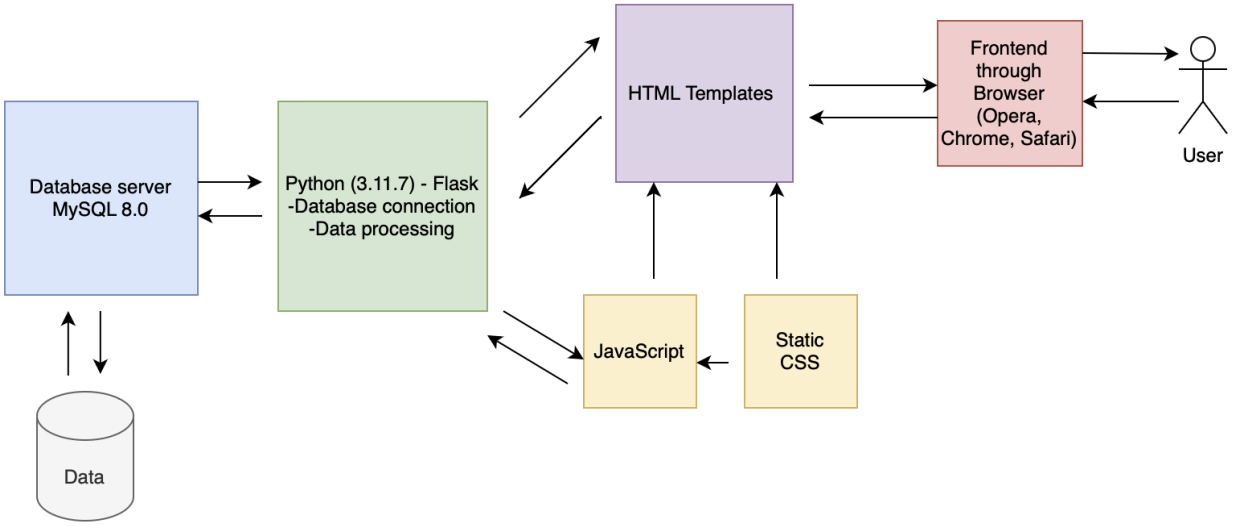

Figure 1: Graphic representation of the application's software architecture. This application is based on a MySQL 8.0 database that contains the relevant information about SNPs on chromosome 1 in 3929 individuals. The software was developed using Flask through Python 3.11.7. The database and Flask interact through the mysql.connector package to run the code of the app based on what the user chooses. Flask uses Javascript, HTML templates and CSS to define the visual representation of the website. This image was created using Draw.io.

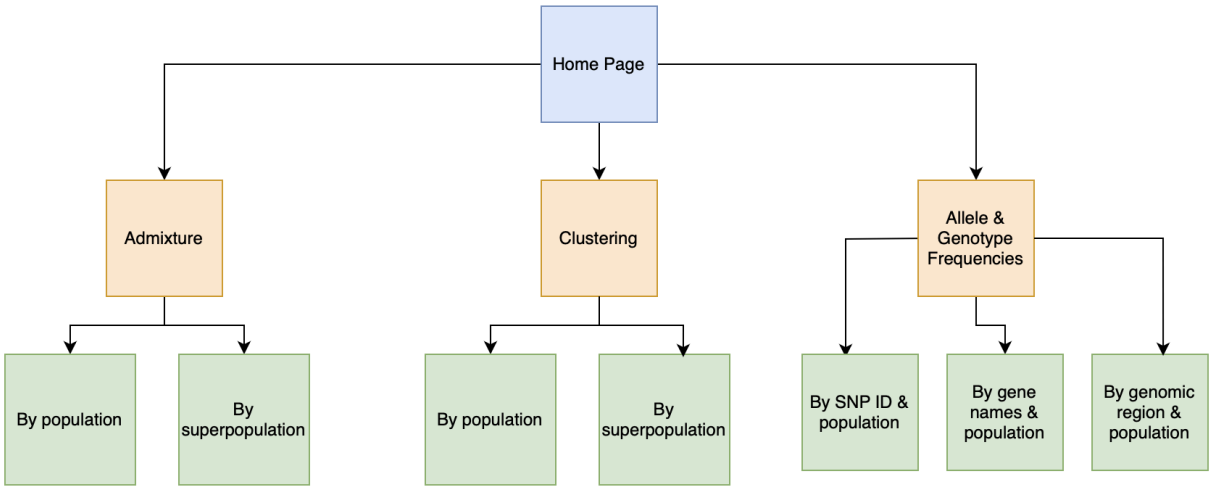

Figure 2: Detailed view of frontend of the web application showing all the possible analyses available to the user. Admixture and Clustering analysis can be done based on populations or superpopulations. Allele and genotype frequencies can be retrieved for SNPs of interest by directly choosing the SNPs of interest, the gene names or a genomic region on chromosome 1. This image was created using Draw.io.

How to run Team Mint application

- Use python 64bit (v3.11.7 64bit).
- Download the 'Website' folder and 'main.py'. Run main.py using your choice of Python IDE or open up a terminal and type cd /path/to/Website and python main.py. Ensure MYSQL 8.0.36 database is created via downloading all the files in "Make the database" and running make_database.ipynb or if you want to start from scratch to download all the files in "Initial Data Processing".
- For Mac ensure on the top where the imports matplotlib.use("AGG") is coded.

Installation

There are installations required to run the web application:

Python

- The Python version used in 3.11.7 64bit download from https://www.python.org/downloads/release/python-3117/. To use flask and other libraries, Python will be needed. Pip manager is already installed and will be needed to download the other libraries listed down below.
- Download a suitable code editor. Visual Studio Code (VS Code) was used and recommended due to its lightweight features and additional extension. It can be downloaded from https://code.visualstudio.com/download/. For VS code remember to install the Python extension.

Dependencies

Important to keep in mind to run Mint web applications that when you are in VS Code you are installing the libraries in the correct path for your local device. It is ideal for when opening a new project that you create a new virtual environment either using VS Code terminal "py -3 -m venv (pick a name)" or ctrl+shift+p --> select python environment --> create virtual environment.

Flask

- Flask is a web application framework written in Python to allow the code to run as a web application. For the following library pip install “name of package” on VS Code terminal will be the default installation method. Plus, a link to the website for further information.
- Install flask by opening the terminal in VS Code and type “pip install flask” and press enter.
- From flask the imported functions required are Flask, render_templates, redirect, url_for, request, send_file.
- Another install needed is flask_wtf and import Flaskform which is needed to create a class to allow selection of population and various variables for one of the allele and genotype frequencies page.
- More information can be found here: https://flask.palletsprojects.com/en/2.2.x/

pandas

The python library is used for creation of data frame for the SQL database. Pip install pandas. https://pandas.pydata.org/docs/getting_started/index.html#getting-started/

mySQL.connector

This package is used to connect to the SQL database created with the make_database.ipynb. https://pypi.org/project/mysql-connector/

NumPy

Python library used for multiple numerical computing for example editing a NumPy array to replace missing values with zero.
https://numpy.org/install/

Matplotlib & Matplotlib.pyplot

A library used to generate the graphs for PCA and admixture.
https://matplotlib.org/stable/index.html
https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html

io & BytesIO

The BytesIO class provides a convenient and versatile way to work with in-memory binary data allowing the image that needs to be processed and manipulated can be stored as binary data without the need for temporary files.
https://www.digitalocean.com/community/tutorials/python-io-bytesio-stringio

Base64

Imported from base python is used to encode the binary data into base64-encoded ASCII text format.

wtforms

- This module mainly serves the purpose of web development, in particular building forms using Flask.
- From wtforms import IntegerField, SubmitField, SelectMultipleField, StringField
- In summary these classes allow for various specific types of input field that can be used to collect different types of user input in the web application.

https://pypi.org/project/WTForms/

wtforms.validators

- As the name suggests this module functions around validating inputs to be what is expected (also used as error handling).
- From wtforms.validators import Optional, NumberRange.

https://pypi.org/project/wtforms-validators/

Itertools

From this module we imported combinations so that for the fst matrix calculation to iterate over pairs of indices in def calculate_fst.

Packages required to recreate the SQL database:

PyMySQL

This package allows users to connect to the MySQL server in the python interface. It's used in make_database.ipynb. Documentation: https://pymysql.readthedocs.io/en/latest/.

Zarr

This package supports the extraction of data from compressed zarr files (annotated.zarr and clinvar.zarr). Installation: https://zarr.readthedocs.io/en/stable/installation.html, dependencies: NumPy.

Scikit-allel

This package provides utilities for exploratory analysis of large-scale genetic variation data, including the reading and manipulation of large VCF files. It contains a function named vcf_to_zarr which can convert large vcf files to compressed zarr files but still support quick data extraction. Documentation: https://scikit-allel.readthedocs.io/en/stable/

Numcodecs

A python package provides buffer compression and transformation codecs for other data storage applications, used in extract_to_zarr.py file along with scikit-allel package to convert VCF file to zarr file. https://github.com/zarr-developers/numcodecs

MySQL

MySQL is an open-source database management system designed to efficiently store, manage, and retrieve data. To install MySQL, visit https://dev.mysql.com/downloads/ and download the MySQL Community Server along with MySQL Workbench (version 8.0). Alternatively, you can opt to download MySQL Workbench directly from https://dev.mysql.com/downloads/workbench/, which includes additional MySQL software such as MySQL Shell version 8.0.36.

HTML/CSS and JavaScript

To create a functional website, HTML serves as the foundation, complemented by CSS and JavaScript to enhance its features and user experience. HTML is essential for structuring content, employing elements like headings, paragraphs, lists, and links. Meanwhile, CSS facilitates the visual presentation of HTML content; in this project, it's utilized to infuse a mint colour theme across all HTML files. Additionally, JavaScript plays a crucial role in implementing various functionalities such as dropdown menus and error handling, ensuring smooth interaction and correct inputs. These technologies work in tandem to deliver an engaging and seamless browsing experience.

Annotating the VCF file

After viewing the VCF file and realising how few SNPs had rsIDs, we attempted to convert the existing IDs to their corresponding rsIDs. We initially attempted to retrieve the SNP ids using online tools, such as the Variant Effect Predictor (VEP) from Ensembl and the Variant Annotation Integrator provided by UCSC Genome Browser (McLaren et al., 2016; Hinrichs et al., 2016). Unfortunately, the web versions of those tools were not able to support the size of our input file. This is why we eventually chose to run bcftools from the command line in order to annotate the chr1.vcf.gz file (Li, 2011). The SNP ids were found in the dbsnp_146.hg38.vcf.gz file available through the FTP site of Broad Institute (*Downloading "Dbsnp 146.Hg38.Vcf.gz" File for BQSR*, n.d.). UCSC Genome Browser provides more updated SNP databases as text files which were unable to convert to a vcf file the input format required by bcftools. The new annotated VCF file had many more rsIDs than the original VCF file but we were still unable to convert all the IDs to rsIDs.Therefore, we resorted to using the combination of the SNP's chromosome, position, reference, and alternative alleles for a unique ID for the SNPs that did

not have an rsID. While annotating did not provide the intended output we still used the annotated file for future data processing.

Data collection

Clinical relevance information:
When gathering clinical information on SNPs our first approach was to see if we could utilise APIs to gather up-to-date information on the queried SNPs straight from the source such as Clinvar or Ensemble. While this approach was very ambitious we soon realised that it was very inefficient as it required many hours to study and learn how to use APIs and it also relied on an external database that may not always be accessible. After spending a significant amount of time working on an API query function, we decided to retrieve the clinical information from a database, our chosen one was Clinvar, and then process and store this information on our database which was much more efficient (Landrum et al., 2017). The data was downloaded as clinvar.vcf.gz. By comparing the positional data, reference, and alternate bases of the SNPs from ClinVar with the SNPs from our VCF file in python we were able to map the SNPs to their clinical information, if it was available.

Gene names:

The names of genes and their positions (start base and end base) were gathered from Ensembl (Kinsella et al., 2011). The dataset was exported using the data-mining tool called BioMart provided by Ensembl. Users can replicate this mining process by first selecting the “Ensembl Genes 111” database and the “Human genes (GRCh38.p14)” dataset. Then, users should narrow down the region to chromosome 1 and choose attributes such as “Gene name,” “Gene start (bp),” and “Gene end (bp)” under the Features - GENE branch. The output can be downloaded by clicking “Results” as a tsv or csv file. Additionally, users should filter this dataset after exporting to include only genes with names. This dataset enabled the mapping of SNPs to their corresponding genes based on the positional data of SNPs and genes in python.

Database Populating

Procedure

To install a database copy, users should utilise the final_db.sql file to import both the data and the database structure into their own databases.

Alternatively, users have the option to generate the database by initially converting annotated.vcf.gz and clinvar.vcf.gz files into annotated.zarr and clinvar.zarr files. This could be done by executing the extract_to_zarr.py file. This script is designed to extract data from VCF files, storing it into a Zarr file saved on disk. This approach can be highly beneficial when dealing with large VCF files, as loading data into NumPy arrays often consumes significant main memory resources, whereas Zarr files offer efficient on-disk storage and retrieval of numerical arrays. The execution of this process requires the installation of two packages: scikit-allel and numcodecs.

After converting the VCF files to zarr format, users can choose to execute make_database.ipynb to generate the database. This notebook is designed to process the data stored in the Zarr files and populate the database accordingly. Minor changes are required according to your credentials within the make_database.ipynb, such as the database name, host, username and password. Additionally, ensure that the paths of other required files such as annotated.zarr, clinvar.zarr, allele_freq_df.csv, df_sorted_q.csv, only_gene_name.csv, pca_data.csv, and pop_superpop.tsv are correctly set to prevent FileNotFoundError from arising while running this Jupyter notebook. The required packages for running this notebook include pymysql, mysql connector/python, pandas, and zarr. Users should ensure that all necessary packages have been installed before proceeding with execution.

SQL database schema

Overview

The "final" database comprises six tables. The "allele_freq" table contains allele frequency data for each variant and links with the "allele_char" and "gene_names" tables. These tables contain variant IDs, position information, and corresponding gene information. This setup enables users to query and filter variants of interest based on ID, position, or gene name and retrieve their allele frequency data.

The "pop_superpop" table contains the ID of each sample along with their corresponding population and superpopulation. It links with the "pca_data" and "admixture_data" tables. This structure enables users to filter samples of interest based on their population and superpopulation and visualise their PCA and admixture results through charts.

SQL is not the tool for doing genetic analysis directly, although it can be useful for data administration, storage, and retrieval. Researchers frequently combine bioinformatics tools or computer languages for genetic analysis with SQL for effective data management. This integration enables cooperation, reproducibility, and an organised method to storing and managing genetic data.

Tables

Table: snp_char:

Description:

This table stores the relevant information from the annotated.vcf.gz and clinvar.vcf.gz files and links with table gene_names and allele_freq.

Columns:

snp_id: Identifier of variants from the annotated VCF file, shown as a unique identifier containing the position of the SNP on chromosome 1 followed by the reference and alternate alleles at that position or as the rsIDs when available.

Data type: varchar (255)

Constraints: primary key

position: The position of the SNP on chromosome 1 retrieved from the VCF file

Data type: integer

ref_base: Reference base, must be one of A, C, G, T, N.

Data type: char (1)

alt_base: Alternative base, must be one of A, C, G, T, N.

Data type: char (1)

disease_name: ClinVar's preferred disease name for the concept specified by disease identifiers in CLNDISDB. Type = String. ID = CLNDN. A short key in the additional information (INFO) fields separated by semicolon from the clinvar.vcf.gz file. NULL value is allowed if no information is found.

Data type: longtext

classification: Aggregate germline classification for this single variant. Type = String. ID = CLNSIG. A short key in the additional information (INFO) fields separated by semicolon from the clinvar.vcf.gz file. NULL value is allowed if no information is found.

Data type: longtext

Indexes:

Index name: position | PRIMARY

Type: Sinple Index | Primary Key Index

Indexed column: position | snp_id

Ordering: Ascending

Data Type: Integer | varchar

Purpose: Improve query performance when filtering or joining data based on the “position” column and “snp_id” column in the snp_char table.

Relationships:

In table snp_char, the column “position” links to the foreign key “position” in table gene_names. The primary key “snp_id” links to the foreign key “snp_id” in table allele_freq.

Table: gene_names:

Description:

This table links the SNPs in table snp_char with the genes they belong to in human chromosome 1 based on their position data.

Columns:

id: Unique identity for each row generated by attribute AUTO_INCREMENT.

Data type: integer

Constraints: primary key

gene_name: The name of genes in human chromosome 1 from only_gene_names.csv file retrieved from Ensembl.

Data type: varchar (255)

position: The position of reference base(s) from annotated chr1.vcf.gz file. Reference: column position in table snp_char.

Data type: integer

Constraints: foreign key

Indexes:

Index name: PRIMARY

Type: Primary Key Index

Indexed column: id

Ordering: Ascending

Data Type: Interger

Purpose: Improve query performance when filtering or joining data based on the “id” column in the gene_names table.

Relationships:

In table gene_names, the foreign key “position” links to the column “position” in table snp_char.

Table: allele_freq:

Description:

This table contains the minor allele frequency data of variants from the VCF file in 27 different human populations. The allele frequency data was calculated based on the genotype information on samples for each variant. Caution: When populating this table using make_database.ipynb and allele_freq_df.csv, please ensure that the SNP ids from allele_freq_df.csv match with the SNP ids from the snp_char table. Use the last script in make_database.ipynb to check for any mismatches in SNP ids and update them manually if necessary.

Columns:

snp_id: Identifier of variants from the annotated VCF file, shown as a unique identifier containing the position of the SNP on chromosome 1 followed by the reference and alternate alleles at that position or as the rsIDs when available. Reference: snp_id column in table snp_char.

Data type: varchar (255)

Constraints: primary key, foreign key

SIB: The minor allele frequencies of all SNPs in the Siberian population.

Data type: float

* The rest 26 columns in this table are similar to the SIB column where the column name corresponds to a population and the values represent the minor allele frequencies for each SNP in each of the populations.

Indexes:

Index name: PRIMARY

Type: Primary Key Index

Indexed column: snp_id

Ordering: Ascending

Data Type: varchar

Purpose: Improve query performance when filtering or joining data based on the “snp_id” column in the allele_freq table.

Relationships:

In table allele_freq, the foreign key “snp_id” links to the primary key “snp_id” in table snp_char.

Table: pop_superpop:

Description:

This table contains the basic information for each human sample: their sample ID and the population and superpopulation they belong to. This table links with table pca_data and table admixture_data. The superpopulation labels were retrieved from the 1000 Genomes Project website (The 1000 Genomes Project Consortium, 2015). For the purposes of the analyses performed in this application the Siberian samples were assigned to the European superpopulation.

Columns:

sample_id: The identifiers for human samples from population SIB (Siberia) and other samples from several human populations from the 1000 Genomes Project.

Data type: varchar (255)

Constraints: primary key

population: The name of human populations that each human sample belongs to.

Data type: varchar (255)

superpopulaton: The name of the human superpopulation each sample belongs to. This was done based on the geographic location of human populations at continental level.

Data type: varchar (255)

Indexes:

Index name: PRIMARY

Type: Primary Key Index

Indexed column: sample_id

Ordering: Ascending

Data Type: varchar

Purpose: Improve query performance when filtering or joining data based on the “sample_id” column in the pop_superpop table.

Relationships:

In table pop_superpop, the primary key “sample_id” links to the foreign key “sample_id” in table pca_data. The primary key “sample_id” links to the foreign key “sample_id” in table admixture_data.

Table: pca_data:

Description:

This table contains the results of principal component analysis (PCA) for each sample. Only the first two principal components were inserted for graphing.

Columns:

sample_id: The identifiers for human samples. Reference: column sample_id in table pop_superpop.

Data type: varchar (255)

Constraints: primary key, foreign key

pc1: Principal component 1

Data type: float

pc2: Principal component 2

Data type: float

Indexes:

Index name: PRIMARY

Type: Primary Key Index

Indexed column: sample_id

Ordering: Ascending

Data Type: varchar

Purpose: Improve query performance when filtering or joining data based on the “sample_id” column in the pca_data table.

Relationships:

In table pca_data, the foreign key “sample_id” links to the primary key “sample_id” in table pop_superpop.

Table: admixture_data:

Description:

This table contains the fractions of ancestry attributed to each of the 5 ancestral populations for each sample. The assignment column was created by assigning each sample to the ancestral population with the highest value.

Columns:

sample_id: The identifiers for human samples. Reference: column sample_id in table pop_superpop.

Data type: varchar (255)

Constraints: primary key, foreign key

pop1: The proportion of each sample that belongs to ancestral population 1.

Data type: float

pop2: The proportion of each sample that belongs to ancestral population 2.

Data type: float

pop3: The proportion of each sample that belongs to ancestral population 3.

Data type: float

pop4: The proportion of each sample that belongs to ancestral population 4.

Data type: float

pop5: The proportion of each sample that belongs to ancestral population 5.

Data type: float

assignment: The highest value for each sample in ancestral populations.

Data type: varchar (255)

Indexes:

Index name: PRIMARY

Type: Primary Key Index

Indexed column: sample_id

Ordering: Ascending

Data Type: varchar

Purpose: Improve query performance when filtering or joining data based on the “sample_id” column in the admixture_data table.

Relationships:

In table admixture_data, the foreign key “sample_id” links to the primary key “sample_id” in table pop_superpop.

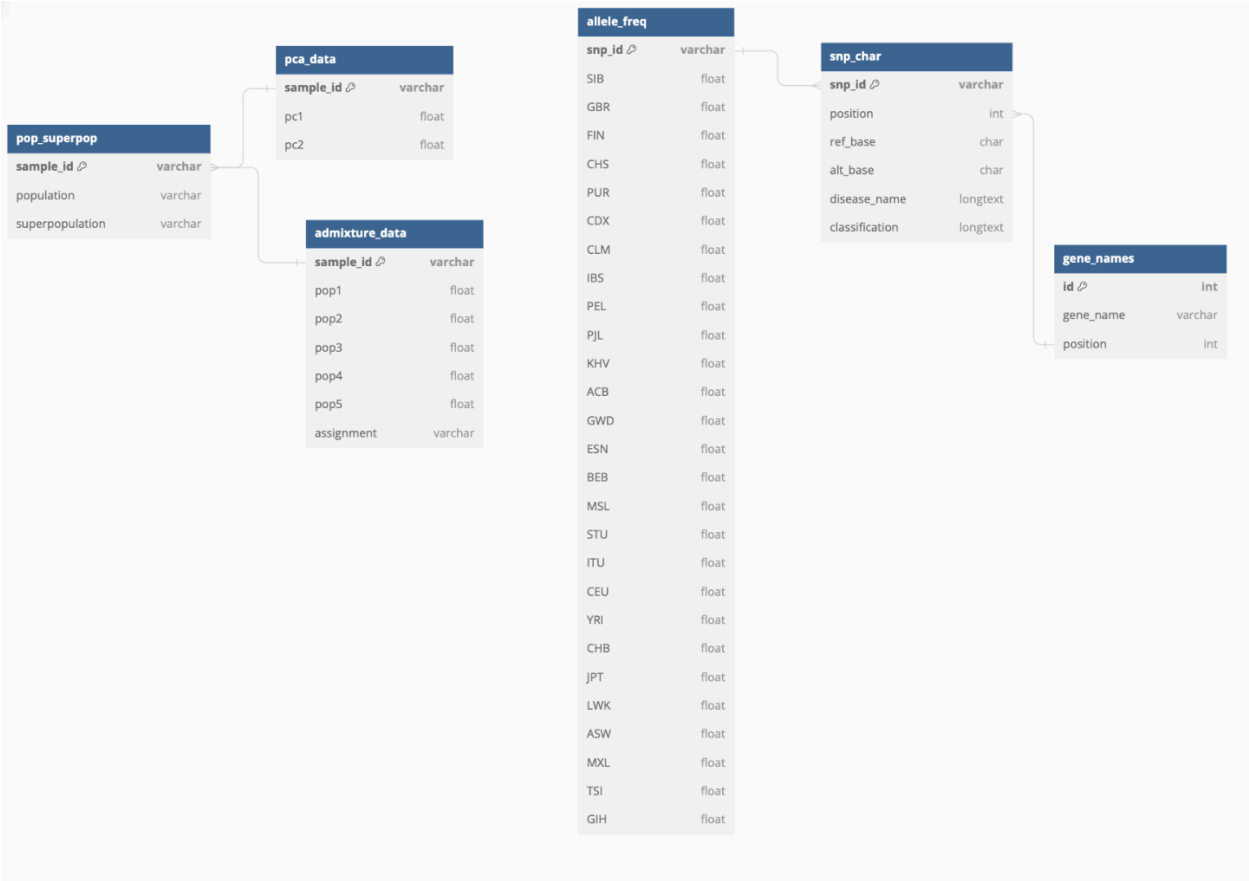

Figure 3: Schema of the database used in this application called ‘final’. The database has 6 tables; pop_superpop, pca_data, admixture_data, allele_freq, snp_char and gene_names. Pca_data and admixture_data tables are linked to the pop_superpop table through their sample_id columns. Allele_freq table is linked to the snp_char table through the snp_id column while gene_names table is linked to the snp_char table through the position column. This image was created using dbdiagram.io.

Limitations

A limitation of our database schema is the fact that not all tables are connected to each other.

App features

Clustering

For the clustering analysis Principal Component Analysis (PCA) was the chosen method. PCA is a commonly used type of clustering analysis usually as the first step in population genetic studies in order to cluster the samples based on their genetic ancestry and account for that in downstream analyses (McVean, 2009; Conomos, Miller & Thornton, 2015; Elhaik, 2022). It reduces the dimensionality of the data into a number of specified principal components (PCs), each explaining a proportion of the genetic variation in the data which can be visualised in a scatterplot. In the plot samples that are closer together would reveal a higher possibility of shared ancestry. PCA is so popular in population stratification studies because of its ability to be

used no matter the size of the dataset and the fact that it does not require us to make any assumptions about the distribution of the data (Elhaik, 2022).

It was first attempted to carry out PCA in python by writing a script that firstly computed the hamming distance between all samples for all SNPs that were provided, as it has been used before, but it was soon realised that this method was not computationally efficient for such a large dataset (Wang, Kao & Hsiao, 2015; Freyman et al., 2020). Therefore, the computationally efficient PLINK program was used. PLINK is a whole genome association analysis toolset that can carry out a variety of analyses on genotype data with minimal computational resources, including PCA (Purcell et al, 2007). Here, PLINK was used to firstly convert the VCF file to a BED format that is the accepted input file format of PLINK's PCA analysis. Then the PCA command was applied on the data which calculated the covariance matrix of the allele frequencies, the eigenvectors revealing where the data is the most diverse and the respective eigenvalues showing the degree of diversity. Since there is no consensus on the number of PCs to use in such investigations, the number was set to 20 which is PLINK's default (Elhaik, 2022; Purcell et al, 2007). The output file of this was processed in python to only keep the first and second PCs as they were the ones that captured the highest proportion of variability in the dataset and was then imported as a table to the SQL database (McVean, 2009). The samples were linked to their population and superpopulation labels and were thus ready for visualisation in a scatter plot using matplotlib where the axes represent the two PCs and the samples could be coloured based on either their population or superpopulation labels.

Limitations

As the number of samples in the dataset increases the proportion of variance explained by the first two PCs gets smaller and smaller and less accurate. Additionally, since only the first two components are visualised which as stated before only capture a small proportion of the variance in the dataset it is harder to identify individuals with ancestries from more than one population (Gaspar & Breen, 2019). Moreover, in the low dimensional space the distances between samples may be distorted (Elhaik, 2022). Alternatives to PCA that could be tried include the non-linear LAPSTRUCT algorithm which has been suggested to outperform PCA (Wu et al., 2011).

Admixture

For the admixture analysis the ADMIXTURE algorithm was used (Alexander, Novembre & Lange, 2009). This can estimate the fractions of ancestry from a predefined number of hypothetical ancestral populations (k) for each sample by maximising the biconcave log-likelihood model. According to the ADMIXTURE model the success probability in the binomial distribution depends on how much of an individual's ancestry can be assigned to each ancestral population and one allele's frequency in each ancestral population (Alexander & Lange, 2011). ADMIXTURE has demonstrated comparable performance to other established admixture algorithms, such as STRUCTURE, while being significantly faster than them (Alexander, Novembre & Lange, 2009).

The ADMIXTURE software was used to carry out the analysis from the BED file created in PLINK previously. Since ADMIXTURE allows the user to specify the number of ancestral populations to be inferred from the data, the k was set to 5 as it aligns with the number of superpopulations that the data come from in the dataset used. Furthermore, the choice of a higher k would require more iterations of the algorithm thus resulting in a delay in obtaining results. For example it was shown that the number of algorithm iterations increases by 130 when going from k = 2 to k = 3 (Alexander, Novembre & Lange, 2009).

Once ADMIXTURE was run the resulting Q file was imported to python where the sample ids were added and an assignment column was created by assigning each sample to the ancestral population with the highest value so it would be later used in the barplot to sort the samples within each subpopulation by this column. The final dataframe was imported as a table to the SQL database and the samples were linked to their population and superpopulation labels. The data were then visualised in a stacked bar plot (Koganebuchi et al., 2023; Sato et al., 2014). In this visualisation the matplotlib python library was used.

Limitations

Making the right choice on the number of ancestral populations to set in the admixture analysis is not an easy task; if more time was available different k values should've been tested to see which provides more accurate results (Alexander, Novembre & Lange, 2009). A limitation of ADMIXTURE is that it does not account for linkage disequilibrium (LD) (Alexander, Novembre & Lange, 2009). Since no pre-processing of the data was performed it means that the results are biased. In the future, before this analysis is run SNPs in known high LD regions can be filtered to reduce bias in the results. STRUCTURE is another highly used algorithm in admixture analysis that could also be tried to compare the performance of this dataset in the two algorithms (Hubisz et al., 2009). An assignment column was mentioned earlier to sort the samples within each subpopulation to improve the visualisation of the results which was not used so in future developments of this application it can be utilised.

Allele and Genotype Frequencies

PLINK was used to calculate the minor allele frequencies per SNP per population from the VCF file (Purcell et al, 2007). We used the annotated VCF file as input. Using the --freq flag tells PLINK to calculate the minor allele frequencies for each SNP in the file. The --loop-cats population was used to ensure that the allele frequencies would be grouped by population. The results were imported in python where they were merged into one table where each row represented a SNP and each column represented a population containing the minor allele frequency values. The final table was imported to the SQL database.

The search page allows the user to search SNPs and select populations for analysis. Three search forms were created; one per possible way a user can use to search SNPs.
The three query types are by SNP IDs/rsIDs, by gene names or by genomic region through a search box. For the position search form, in the HTML template, we prompted the user to select numbers within the range 12782-248936926, as these were all the SNPs positions available on

our database. There are two boxes for the start and end positions respectively, the search_snp(): function makes sure the end position number is larger than the start and that numbers in both fields are entered; it does not proceed if the numbers entered in the two search boxes do not meet all criteria. Furthermore, when the user enters a gene name or a SNP ID that is not in our database a message saying there is no information available for this SNP is shown in the results page.

The population selection form is clickable and allows for the selection of multiple populations. The function then constructs the necessary query for SQL, for each SNP searched to get the reference and alternative alleles as well as the clinical information from the snp_char table in the database. Another query is made to the allele_freq table to retrieve the minor allele frequency of the SNPs searched for the selected populations. Using the alternative allele frequency, the reference allele frequency and the genotype frequencies can be calculated using a python loop according to the Hardy Weinberg Equilibrium (HWE) (Andrews, 2010). Assuming q is the allele frequency of the minor allele and p is the allele frequency of the reference allele, p can be calculated using the following formula: $p + q = 1$. Furthermore, the genotype frequencies can be calculated using the following formula; $p^2 + 2pq + q^2 = 1$.

Limitations

HWE makes several assumptions about the population being studied such as that the population size is large; this is not always true in our dataset, for example the Siberian population is large with 726 samples but the African Ancestry in the Southwestern USA population group is only 74 samples. Other HWE assumptions that might be violated include no migration into or out of the population and no mutations occurring. Therefore, in future developments of this application more accurate ways to calculate the allele and genotype frequencies should be explored; especially ones that take into account all the processes happening in dynamic human populations.

Additionally, we had a vision for our search forms to include an autocomplete ability to aid the user in their search and reduce errors, after creating a functioning autocomplete, with jQuery, we realised that due to the sheer number of SNPs, the function took a significant amount of memory space and froze the page/computer. Due to this, we decided to side line this function as it was not feasible with our current computational power.

Another thing that should be noted is that in the allele_freq table, we replaced NaN values with 0 which may cause bias in certain analyses. In future updates we could replace the NaN values with a predicted value via machine learning methods (Wang et al., 2022).

Fst matrix

If the user selects multiple populations when exploring the allele and genotype frequencies of SNPs, a matrix of pairwise population genetic differentiation is created. The Fst matrix function is called to calculate the Fixation index (Fst), which is a measure of population genetic differentiation due to genetic structure. Fst can range from 0 to 1, where 0 is no difference in genetic material and 1 means completely different. The formula for the fixation index is the

following: Fst = N/D where N = q1(p2−p1)+q2(p1−p2) and D = q1p2+p1q2=N+q1p1+q2p2. In the equations above p1 and p2 are the reference allele frequencies of the SNP in population 1 and population 2 respectively, and qi = 1 – pi (Bhatia et al., 2013) corresponds to the alternative allele frequencies.
The numerator is the difference in allele frequencies between the two populations at a specific locus and the denominator is the expected heterozygosity in the combined population. The fixation index is then the ratio of N to D representing the proportion of genetic variation due to differences between populations relative to the total genetic variation.

The Fst value between those two populations is then added to the matrix in both cells corresponding to the two populations and the user is able to download this with the click of a button. The complete matrix is also visualised using matplotlib and is found at the bottom of the results page. The cells of the plot are annotated with the Fst values in the matrix to make it clearer for the user. The colour scheme of the plot allows the user to easily see which populations are the most different, as the larger the Fst value the lighter the colour of the box representing it. The colour bar on the side shows the user which colours mean what.

Limitations:
When calculating the mean allele frequency for a SNP across the selected populations there would often be just 0 values causing the function to return a division by 0 error to avoid this we skipped these and instead returned 0 to the matrix, which introduces some bias to the results.

Other features

Scroll to the bottom button

On the results page of the allele and genotype frequencies, we have implemented a JavaScript code in the HTML to scroll down to view the Fst matrix. This is because, depending on the parameters, a lot of information can be generated. Therefore, the scroll-down feature will allow for faster access to download the data and access the heatmap.

Download button

When looking at the Fst matrix graph there is a download link button where if the user clicks on it they will download a Fst matrix.txt file (there is an example one found in the github) which contains the results of the Fst matrix calculations.

Error 404

When the user attempts to access a URL that does not exist in our web application it leads to a page saying that the page does not exist and prompts the user to check their spelling.

No data found errors / UnboundlocalError

For the allele and genotype frequencies for the position start and end if the user picks a small range where there is no data within the selected range it will return a message “No data found within this range”.

In the website application there is a JavaScript code in the HTML to ensure that at least one population is selected where if no population is selected “Please select at least one population/superpopulation” message will return.

In allele and genotype frequency there is also code in place for incorrect names if not given gene name or SNP id found in the SQL database it returns message(s) saying that the name selected do not have data associated with them.

Further development

The website and its underlying code remain in the initial version, offering numerous opportunities for optimisation to decrease code size and enhance memory usage, particularly in graph plotting functionalities. Implementing additional features could greatly enhance user experience, including the option for users to customise the name of the downloaded 'fst.txt' file. Moreover, enriching the dataset with a wider range of data and including additional clinical information would significantly enhance the website's content and usefulness.

References:

Alexander, D. H., & Lange, K. (2011). Enhancements to the ADMIXTURE algorithm for individual ancestry estimation. *BMC Bioinformatics*, *12*(1). https://doi.org/10.1186/1471-2105-12-246

Alexander, D. H., Novembre, J., & Lange, K. (2009). Fast model-based estimation of ancestry in unrelated individuals. *Genome Research*, *19*(9), 1655–1664. https://doi.org/10.1101/gr.094052.109

Andrews, C. A. (2010). *The Hardy-Weinberg Principle*. Nature.com. https://www.nature.com/scitable/knowledge/library/the-hardy-weinberg-principle-13235724/

Bhatia, G. et al. (2013). Estimating and interpreting FST: The impact of rare variants. *Genome Research*, *23*(9), 1514–1521. https://doi.org/10.1101/gr.154831.113

Conomos, M. P., Miller, M. B., & Thornton, T. A. (2015). Robust Inference of Population Structure for Ancestry Prediction and Correction of Stratification in the Presence of Relatedness. *Genetic Epidemiology*, *39*(4), 276–293. https://doi.org/10.1002/gepi.21896

Downloading "dbsnp 146.hg38.vcf.gz" file for BQSR. (n.d.). Www.biostars.org. Retrieved February 28, 2024, from https://www.biostars.org/p/9554277/

Elhaik, E. (2022). Principal Component Analyses (PCA)-based findings in population genetic studies are highly biased and must be reevaluated. *Scientific Reports*, *12*(1), 14683. https://doi.org/10.1038/s41598-022-14395-4

Freyman, W. A., et al. (2020). Fast and robust identity-by-descent inference with the templated positional Burrows-Wheeler transform. *BioRxiv (Cold Spring Harbor Laboratory)*. https://doi.org/10.1101/2020.09.14.296939

Gaspar, H. A., & Breen, G. (2019). Probabilistic ancestry maps: a method to assess and visualize population substructures in genetics. *BMC Bioinformatics*, *20*(1). https://doi.org/10.1186/s12859-019-2680-1

Hinrichs, A. S. et al. (2016). UCSC Data Integrator and Variant Annotation Integrator. *Bioinformatics*, *32*(9), 1430–1432. https://doi.org/10.1093/bioinformatics/btv766

Hubisz, M. J. et al. (2009). Inferring weak population structure with the assistance of sample group information. *Molecular Ecology Resources*, *9*(5), 1322–1332. https://doi.org/10.1111/j.1755-0998.2009.02591.x

Kinsella, R. J. et al. (2011). Ensembl BioMarts: a hub for data retrieval across taxonomic space. *Database*, *2011*(0), bar030–bar030. https://doi.org/10.1093/database/bar030

Koganebuchi, K. et al. (2023). Demographic history of Ryukyu islanders at the southern part of the Japanese Archipelago inferred from whole-genome resequencing data. *Journal of Human Genetics*, *68*(11), 759–767. https://doi.org/10.1038/s10038-023-01180-y

Landrum, M. J. et al. (2017). ClinVar: improving access to variant interpretations and supporting evidence. *Nucleic Acids Research*, *46*(D1), D1062–D1067. https://doi.org/10.1093/nar/gkx1153

Li, H. (2011). A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. *Bioinformatics*, *27*(21), 2987–2993. https://doi.org/10.1093/bioinformatics/btr509

McLaren, W. et al. (2016). The Ensembl Variant Effect Predictor. *Genome Biology*, *17*(1), 122. https://doi.org/10.1186/s13059-016-0974-4

McVean, G. (2009). A Genealogical Interpretation of Principal Components Analysis. *PLoS Genetics*, *5*(10). https://doi.org/10.1371/journal.pgen.1000686

Purcell, S. et al. (2007). PLINK: A Tool Set for Whole-Genome Association and Population-Based Linkage Analyses. *The American Journal of Human Genetics*, *81*(3), 559–575. https://doi.org/10.1086/519795

Sato, T. et al. (2014). Genome-Wide SNP Analysis Reveals Population Structure and Demographic History of the Ryukyu Islanders in the Southern Part of the Japanese Archipelago. *Molecular Biology and Evolution*, *31*(11), 2929–2940. https://doi.org/10.1093/molbev/msu230

The 1000 Genomes Project Consortium. (2015). A Global Reference for Human Genetic Variation. *Nature*, *526*(7571), 68–74. https://doi.org/10.1038/nature15393

Wang, C., Kao, W.-H., & Hsiao, C. K. (2015). Using Hamming Distance as Information for SNP-Sets Clustering and Testing in Disease Association Studies. *PLOS ONE*, *10*(8), e0135918. https://doi.org/10.1371/journal.pone.0135918

Wang, H. et al. (2022). Application of machine learning missing data imputation techniques in clinical decision making: taking the discharge assessment of patients with spontaneous supratentorial intracerebral hemorrhage as an example. *BMC Medical Informatics and Decision Making*, *22*(1). https://doi.org/10.1186/s12911-022-01752-6

Wu, C. et al. (2011). A Comparison of Association Methods Correcting for Population Stratification in Case-Control Studies. *Annals of Human Genetics*, *75*(3), 418–427. https://doi.org/10.1111/j.1469-1809.2010.00639.x