
# OPENSUSE HUMAN POPULATION GENETICS WEB APPLICATION

**BMD727P – SOFTWARE DEVELOPMENT GROUP PROJECT**

**QUEEN MARY UNIVERSITY OF LONDON – MSc BIOINFORMATICS (FEBRUARY 2024)**

**Authors: Bruna Plateroti, Hanna Abby, Elma Handzic, Ryan Beadle.**

## Table of Contents

# INTRODUCTION / ABOUT US

Our web application offers a specialized approach to analysing human population genetics, designed to streamline the interpretation of genetic data from varied human populations. It facilitates both the analysis of population structures through clustering and admixture analyses and provides a detailed examination of specific Single Nucleotide Polymorphisms (SNPs), including allele and genotype frequencies and their clinical significance. Additionally, the tool analyses pairwise differentiation between populations.

The backend of the application is powered by SQLite and SQL Alchemy for database management, with Python and PLINK/ADMIXTURE handling the analytical computations. The front end is developed using Flask, enhanced with JavaScript and AJAX for efficient user interaction and request handling.

This software meets the critical need in molecular biology for fast and precise genetic analysis, enabling researchers to uncover genetic diversity and relatedness without the extensive computational effort typically required. Our application stands out in the field of population genetics for its clear visualizations and interpretable results, facilitating a seamless research process from data entry to analysis results. Our Flask implementation ensures a user-friendly experience, making our application an effective gateway for the exploration of genetic data by overcoming the challenges of accessibility and complexity in population genetic analysis.

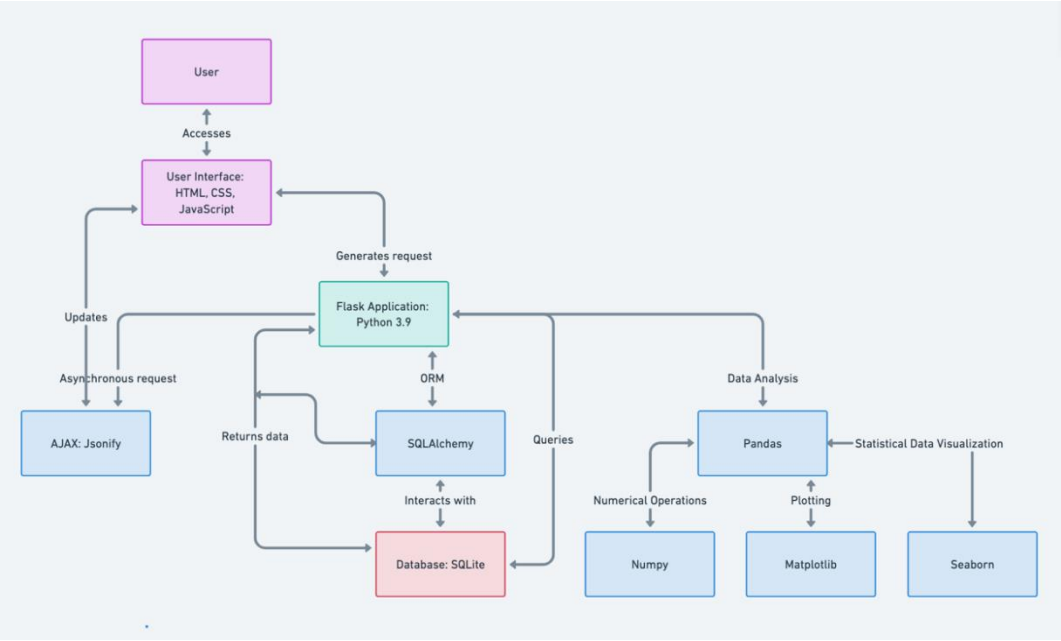


*Figure 1 - This diagram depicts the structure of the web application for data processing and visualization. A user interacts with the interface designed using HTML, CSS, and JavaScript, which sends requests to a Flask application built in Python 3.9. Flask communicates with an SQLite database via SQL Alchemy for data storage. Data analysis is performed using Pandas and NumPy, with results visualized through Matplotlib and Seaborn. AJAX with Jsonify is used for asynchronous web page updates. Pink signifies front end technologies, blue illustrates business logic, red shows backend technology and green visualises integration of technologies between all layers.*

# BACKEND – DATABASE

The original data was provided by Queen Mary University of London in VCF format containing genotyped markers for 726 human samples from an unspecified location in Siberia. They were integrated with the whole-genome sequencing data of samples from different populations from the 1000 Genomes Project database. In total the file contains 3929 individuals, all with data only from chromosome 1. Additionally, an annotation file linking the samples to their respective population codes was provided.

The initial stages in constructing the database involved data filtering using bcftools' *sort* command to organise the information. Subsequently, entries with missing data were removed through the application of the bcftools command *view –e*, ensuring data integrity. Following this, only pertinent columns including sample ID, position, reference allele, alternate allele, AN, AC and GT were retained using the bcftools command *query -f*. The resulting filtered VCF was then exported as a tab-separated values (TSV) file. To further structure the data, the Pandas and NumPy libraries were used for data manipulation, facilitating the creation of distinct TSV files each corresponding to the different database tables.

The database management system (DBMS) we used for this project is SQLite version 3 due to its simplicity and compatibility with the data structure. The design of the relational database schema was tailored to meet specific user requests and software requirements. The schema comprises five tables, each serving specific purposes outlined as followed:

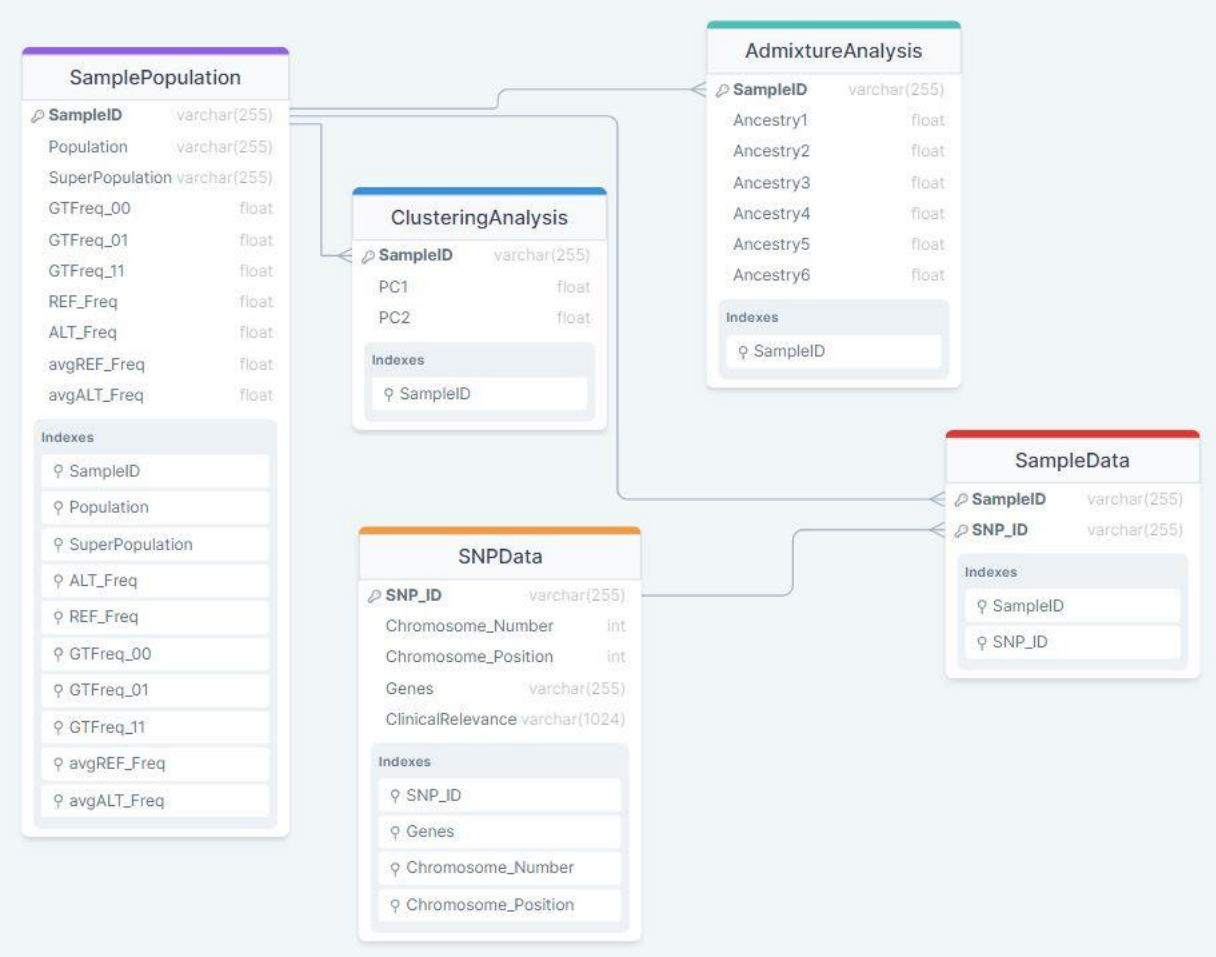


*Figure 2 - Shows the relational database schema which consists of five tables, each containing specific fields corresponding to the data it holds. Additionally, the data type for each field is displayed. Indexes, crucial for optimizing query performance, are denoted at the bottom of each table, which ensure efficient data retrieval for queries.*

All tables within the database were indexed on specific columns to optimise queries, facilitate faster execution, and enhance data quality. Indexing improves query performance by allowing the database to locate data more efficiently without scanning the entire table. The decision to index the table was influenced by software requirements involving the processing of large data and calculations (e.g. pairwise matrix and SNP retrieval). The database query construction process was facilitated by SQL Alchemy for SQL database interaction. For population-related queries, the *SamplePopulation.query* method was employed to efficiently retrieve distinct population and superpopulation names. Regarding data analysis, queries were constructed using *db.session.query* along with *.join* operations to integrate multiple tables, enabling complex analysis and data retrieval tasks.

| **'AdmixtureAnalysis' table** | |
|---|---|
| *SampleID* | A unique identifier for each sample |
| *Ancestry1 to Ancestry6* | These fields represent the ancestry proportions or genetic contributions of each sample to different ancestral populations. Each field corresponds to a specific ancestral population. |
| *Relation* | SampleID is the primary key and foreign key that references the SamplePopulation table. |

| **'ClusteringAnalysis' table** | |
|---|---|
| *SampleID* | A unique identifier for each sample. |
| *PC1 (Principal Component 1)* | Represents the first principal component derived from PCA. Principal components are linear combinations of the original variables. PC1 captures the direction of the greatest variance in the dataset. |
| *PC2 (Principal Component 2)* | Represents the second principal component derived from PCA. PC2 captures a linear combination of the original variable but orthogonal to PC1 and represents the direction of the second greatest variance of the dataset. |
| *Relation* | SampleID is the primary key and foreign key that references the SamplePopulation table. |

| **'SNPData' table** | |
|---|---|
| *SNP_ID* | A unique identifier for each Single Nucleotide Polymorphism (SNP) included in the dataset. |
| *Chromosome_number* | It provides a numerical identifier or reference for the gene associated with the SNP. |
| *Chromosome_position* | Represents the physical position or location of the SNP within the genome |
| *Genes* | Contains the name or the symbol of the gene associated with the SNP |
| *ClinicalRelevance* | Indicates the clinical significance or relevance of the SNP, particularly in relation to disease susceptibility and other medical implications. |
| *Relation* | SNP_ID is the primary key. |

| **'SamplePopulation' table** | |
|---|---|
| *SampleID* | A unique identifier for each sample. |
| *SuperPopulation* | Represents the broader geographical or ancestral group to which each sample belongs. |

| | |
|---|---|
| *GTFreq_00* | Represents the frequency of the reference allele homozygous genotype observed in the sample population. |
| *GTFreq_01* | Represents the frequency of the heterozygous genotype observed in the sample population. |
| *GTFreq_11* | Represents the frequency of the alternate allele homozygous genotype observed in the sample population. |
| *REF_freq* | The overall frequencies of the reference allele observed in the sample population |
| *ALT_freq* | The overall frequencies of the alternate allele observed in the sample population |
| *Relation* | SampleID is the primary key. |

| **'SampleData' table** | |
|---|---|
| *SampleID* | A unique identifier for each sample. |
| *SNP_ID* | A unique identifier for each Single Nucleotide Polymorphism (SNP) included in the dataset. |
| *Relation* | SampleID and_SNP_ID are the compound primary keys and foreign keys that reference both the SamplePopulation and SNPData tables. |

# PROCESSING OF CLINICAL RELEVANCE AND GENES RELATED TO SNPS

Our application includes a robust feature for fetching and storing gene and clinical relevance information related to Single Nucleotide Polymorphisms (SNPs) in the dataset. This process is crucial for researchers who need to understand the potential clinical implications of SNPs.

At the heart of this are four key functions, each tailored to address specific aspects of the data processing workflow. The **fetch_studies_for_snp** function initiates the process, reaching out to the EBI GWAS Catalog to gather studies pertaining to a given SNP ID. Following this, the **extract_study_info** function takes the raw JSON data returned from the API and extracts critical details about the studies, focusing on disease traits associated with each SNP. This structured extraction process ensures that the data is not only accessible but also ready for analysis.

Parallel to this, the **fetch_gene_for_snp** function queries the Ensembl database to identify genes based on the SNP's chromosomal location. This step is pivotal in linking SNPs to their genetic foundations, offering insights into potential biological implications. The culmination of this workflow is the **process_snp_data** function, which orchestrates the reading of SNP IDs and chromosomal locations from the user, leverages the previously described functions to gather and process data, and finally outputs the results into a structured, tab-separated file. This file serves as a bridge, connecting the raw data with potential applications in research and clinical diagnostics.

This element is designed with an emphasis on user experience, featuring a straightforward workflow that guides users from input preparation through to the generation of processed data. It ensures that even those with minimal technical expertise can effectively leverage the tool to enhance their research. Error handling mechanisms and placeholders are woven throughout, providing clear feedback in instances of API failure or when data cannot be retrieved, thus ensuring reliability and user confidence.

The data curated through this method, encompassing disease traits and gene information for each unique SNP, is subsequently stored in the application's “analysis.db” database in the SNPData table. This structured storage ensures efficient retrieval for future queries, allowing researchers to access and analyse genetic associations quickly.

# CLUSTERING – PRINCIPAL COMPONENT ANALYSIS (PCA)

Principal component analysis (PCA) is an unsupervised statistical technique used in population genetics to uncover and visualise the genetic structure and the patterns of populations. It works by reducing the dimensionality of genetic data, transforming it into a new set of variables known as principal components. In the context of this project, PCA is instrumental in analysing human genetic populations, as through the distilling of complex genetic information into principal components, PCA simplifies the data, making it easier to identify the genetic similarities and differences amongst and between populations.

The selection of Principal Component Analysis (PCA) over other clustering methodologies like Multidimensional Scaling (MDS) and Uniform Manifold Approximation and Projection (UMAP) for this project was driven by PCA's exceptional capability to manage high-dimensional genetic data, characterised by the analysis of numerous Single Nucleotide Polymorphisms (SNPs). Unlike MDS and UMAP, which also operate unsupervised but focus more on preserving data's geometric or topological properties, PCA simplifies the complexity of genetic data by transforming it into principal components that capture the most significant variance. This simplification allows for a more intuitive visualisation and understanding of genetic diversity and structure. PCA's computational efficiency, essential for the responsive performance of web-based applications, combined with its direct interpretability, aligns perfectly with the project's aim to make complex genetic analyses accessible and understandable. Therefore, PCA's alignment with our project's goals, its efficiency, and its ability to provide clear, interpretable results amidst the high-dimensional nature of genetic data, made it our top choice over MDS and UMAP.

Before initiating Principal Component Analysis (PCA), we had to prepare and process the genetic data to make it ready for analysis. This preparation involved a thorough cleaning phase where unnecessary columns, such as reference (REF) and alternative (ALT) alleles, allele counts, and the SNP positions, were removed. This process, coupled with the use of the Pandas library, narrows down the data exclusively to the genotype information essential for PCA. Given that genotype data in Variant Calling Files (VCF) are typically stored in string format, a critical transformation step involves converting these genotypes into numerical values. This conversion is necessary for PCA, which requires numerical inputs to perform its calculations. Following this, a matrix is constructed using the **pivot_table** function in pandas, organising the data with samples as rows and SNPs as columns, populated with the newly numerical genotype values. This matrix forms the foundation for the PCA. To ensure a fair and balanced analysis, the data undergoes standardisation, adjusting each SNP to have a mean of zero and a unit variance. This standardisation is crucial, equalising the contribution of each SNP to the PCA and avoiding any bias towards SNPs with larger variance. The decision to focus on the first two principal components (PC1 and PC2) stems from their ability to capture the most significant variance within the data, providing a more simplified yet insightful visualisation of genetic diversity and structure. PC1 normally captures the largest amount of variance, which corresponds to the

variations between populations with distinct ancestorial origins, whilst PC2 might capture more subtle differences such as adaptations to diverse environments or migrations. To visually interpret the PCA results as shown in figure 3, we used the Matplotlib package, assigning distinct colours to each population.

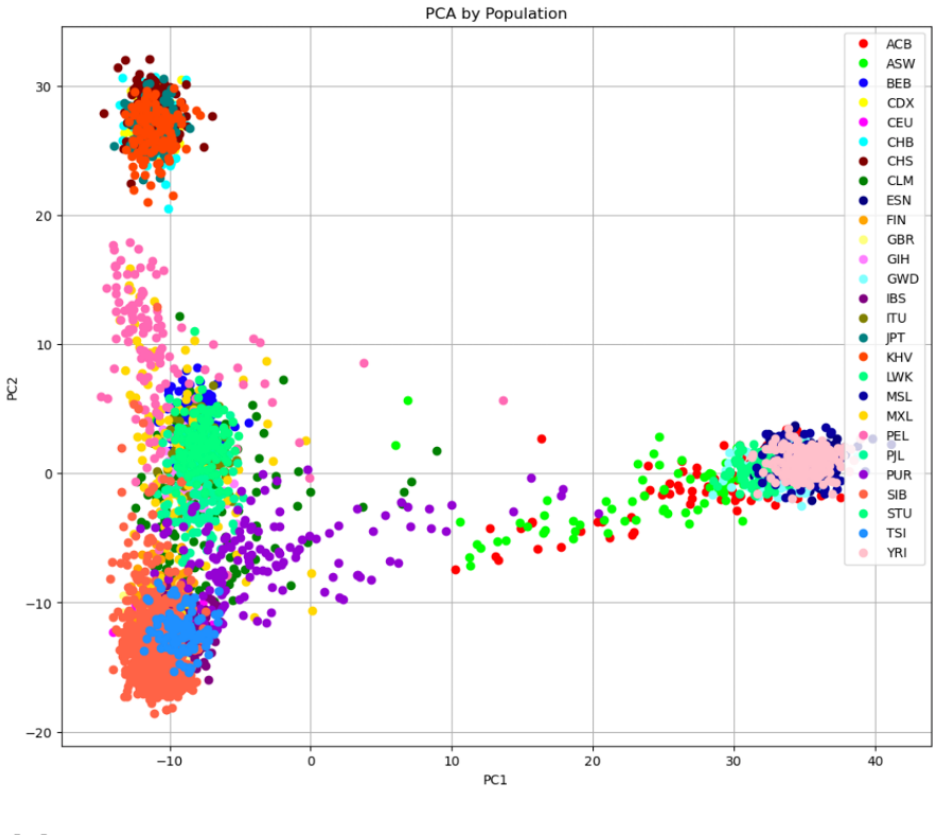


*Figure 3 - Principal Component Analysis (PCA) of genetic data by population. This scatter plot visualises across all 27 populations including Siberia, with each dot representing an individual's genetic makeup projected onto the first two principal components (PC1 and PC2). Different colours correspond to distinct populations, highlighting the genetic diversity within and between these groups, the clustering patterns suggest degrees of genetic similarity and differentiation amongst the populations, which could be reflective of historical migrations, genetic drift or selection.*

The PCA results are then read into the database as TSV files, streamlining both storage and access. This pre-storage mechanism offers a performance boost, enabling quick access to the PCA data for any of the 27 populations within our database. When a user initiates a clustering analysis request, the **perform_pca** function dynamically retrieves the relevant principal component data by executing an SQL query that joins the SamplePopulation and ClusteringAnalysis tables. It then generates a data frame, visualises the first two principal components with Matplotlib, and dispatches the plot in a JSON format for front-end rendering. This system design minimises the computational load and ensures a swift and responsive user interface, eliminating the need for real-time PCA computation and creating a faster and more efficient user experience.

# ADMIXTURE ANALYSIS

Admixture analysis was used to study the ancestral components of each of our populations, by inferring and quantifying the contributions of different potential ancestries to the genetic makeup of each individual within the populations. For our own Admixture analysis, we decided to use the software ADMIXTURE (Alexander et al. Genome Research 2009) [1] due to its ease of use, as well as its useful cross-validation feature to estimate the optimal value of K (number of genetic clusters) and its speed for the large dataset we are working with.

To use ADMIXTURE, we first had to use PLINK (Whole-genome Association Analysis Toolset), a flexible tool designed for the management and analysis of genetic variation data. In our case, we used its data format conversion features to turn our VCF file into PLINK's binary format (.bed, .bim and .fam) to allow for compatibility with ADMIXTURE, as a VCF file cannot directly be used as input.

ADMIXTURE's algorithm was then applied using the PLINK binary files as input to infer population structure by estimating the ancestry proportions for individuals in the dataset. It does this by using a 'block relaxation' approach to alternately update allele frequency and ancestry fraction parameters. Each block update is handled by solving many independent convex optimisation problems, which are tackled using a fast sequential quadratic programming algorithm. Convergence of the algorithm is then accelerated using a quasi-Newton acceleration method. [2] Tests have shown that this algorithm can outperform Expectation-Maximisation (EM) algorithms and Markov Chain Monte Carlo (MCMC) sampling methods by a wide margin.

Additionally, cross-validation errors were computed for different values of K, representing the potential number of genetic clusters within the data. The determination of the optimal K value was done by analysing the cross-validation error graph generated during the ADMIXTURE analysis. In our case, the graph indicated that the cross-validation errors stabilised and leveled off at K=6, signifying a point of diminishing returns regarding predictive accuracy. We considered using K=5 due to the possibility of focusing on the 5 superpopulations, which could be considered for future analyses, but decided to go with our data-driven results instead, as they indicate that there could be substructures or additional patterns in the data that are being captured which might not be present at lower values of K.

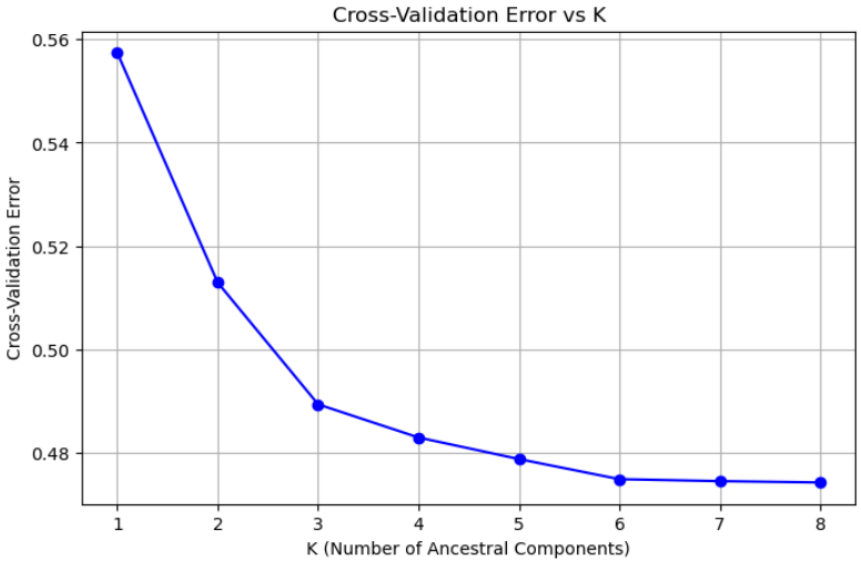


*Figure 4 – Optimal K-value calculated through ADMIXTURE's cross-validation error graph function, which shows levelling-off of CV error at K=6, which we then used for our estimated number of ancestries within our dataset.*

To visualise and test the data, thorough testing was conducted in a Jupyter notebook environment, to make sure that the results seemed sensible for our expectations. The results obtained from the ADMIXTURE analysis, specifically the .Q file for K=6, were then read into a TSV file, which was integrated into the SQL database. The integration process is detailed in the *AdmixtureAnalysis.py* file, which contains the full process from software installation to reading the results into a TSV file to be added to our database.

We considered using alternative software such as STRUCTURE or Frappe, but ADMIXTURE's easy-to-use approach to estimating population structure through its own cross-validation error function sets it apart, as we were not certain of the ideal K value, whereas STRUCTURE is more difficult in that aspect, usually requiring external tools. STRUCTURE uses the same statistical model as ADMIXTURE, but ADMIXTURE is faster due to its numerical optimisation algorithm, allowing it to be a good choice for our large dataset as well. According to tests done by its creators, ADMIXTURE's algorithm also runs faster and with greater accuracy than Frappe's Expectation-Maximisation (EM) algorithm, making it a superior option.

In our Flask web application, the '*/fetch_admixture_data*' route communicates with the front end using jsonify, a Flask utility for converting Python objects to JSON format. The route processes user-selected populations and superpopulations, fetching corresponding data from the backend database via SQL Alchemy queries, specifically from the SamplePopulation and AdmixtureAnalysis tables, with a join on the SampleID columns. The retrieved data is then sent as a JSON response to the HTML page. On the front end, Chart.js, a JavaScript library, renders an interactive chart based on the received JSON data, to show a visualisation of the admixture analysis results:


*Figure 5 – An example of the output from our HTML page for Admixture analysis results, in this case for all samples over every population, providing an interactive display using the Chart.js library, showing a stacked bar representation of the 6 ancestry proportions for each selected population. The webpage features dropdown menus for selecting populations and superpopulations, allowing users to tailor their query to any number of specific populations or superpopulations.*

# ALLELE AND GENOTYPE FREQUENCY CALCULATIONS

The analysis of Single Nucleotide Polymorphisms (SNP) data is fundamental to understanding genetic variations within human populations. Allele and genotype frequencies offer a window into this genetic diversity and are critical for various genetic analyses.

## Genotype Frequency Calculation

Our software includes a function named calculate_genotype_frequencies, which meticulously processes SNP data to compute the frequencies of genotypes. These genotypes are classified into three categories: homozygous reference (0/0 or 0|0), heterozygous (0/1, 0|1 or 1/0, 1|0), and homozygous alternate (1/1 or 1|1). By parsing through a TSV file containing multiple samples' SNP information, the function ensures accurate frequency calculations, accommodating variations in genotype format (using '/' or '|'). The resulting data, mapping each sample to its genotype frequencies, is then compiled into an output TSV file through the write_genotype_frequencies function. This step is essential for subsequent analyses and provides a detailed snapshot of the genetic makeup of the samples.

## Allele Frequency Calculation

The calculate_allele_frequencies function is equally crucial in the SNP data analysis pipeline. It focuses on determining the frequencies of the reference and alternate alleles. Using the genotype frequency data, this function calculates the proportion of alleles across the population sample. The allele frequency data, revealing the prevalence of reference versus alternate alleles for each SNP, is systematically organised and outputted. The write_allele_frequencies function, analogous to the write_genotype_frequencies function, compiles this information into a structured TSV file format, crucial for downstream genetic analyses and offering insights into allelic variations and their distribution patterns.

# FST MATRIX AND VISUALISATION

In the pursuit of understanding genetic differentiation among human populations, our web application employs a suite of computational functions that analyse Single Nucleotide Polymorphisms (SNPs) data. Central to this analysis is the calculate_fst function, which quantifies the genetic variance between two populations. It computes the Fixation Index (FST) by leveraging precalculated average allele frequencies. The weighted average approach accounts for differences in allele frequencies and provides a robust statistical measure that is less prone to sampling variance. This makes it suitable for datasets with varying sample sizes across populations. First, calculate the average allele frequencies for the reference allele and the alternate allele across selected populations. Then, it calculates the variance of allele frequencies among the populations for both alleles. Thereafter it, calculates the Fst for the reference allele and the alternate allele using their respective variances and average frequencies. Finally, it averages the values of the reference and alternate alleles to get a single fst value per population. This weighted average method is well-established in the field of population genetics, allowing our results to be easily compared with a wide range of studies and facilitating collaboration and data sharing within the scientific community. This method can be readily adapted to include additional statistical considerations or to accommodate future enhancements based on emerging genetic research.

$$p_{avg\_ref} = \frac{p1_{avg\_ref} + p2_{avg\_ref}}{2}$$

$$p_{avg\_alt} = \frac{p1_{avg\_alt} + p2_{avg\_alt}}{2}$$

$$\sigma^2_{ref} = \frac{1}{2}\left[(p1_{avg\_ref} - p_{avg\_ref})^2 + (p2_{avg\_ref} - p_{avg\_ref})^2\right]$$

$$\sigma^2_{alt} = \frac{1}{2}\left[(p1_{avg\_alt} - p_{avg\_alt})^2 + (p2_{avg\_alt} - p_{avg\_alt})^2\right]$$

$$F_{ST\_ref} = \frac{\sigma^2_{ref}}{p_{avg\_ref}(1 - p_{avg\_ref})}$$

$$F_{ST\_alt} = \frac{\sigma^2_{alt}}{p_{avg\_alt}(1 - p_{avg\_alt})}$$

$$F_{ST} = \frac{F_{ST\_ref} + F_{ST\_alt}}{2}$$

*Figure 6 - This figure displays the formulas for calculating average allele frequencies (p_avg_ref, p_avg_alt), variances (σ^2_ref, σ^2_alt), and fixation indices (F_ST_ref, F_ST_alt) in two example populations. Average allele frequencies are the means of two observed frequencies. Variances are based on the differences between observed and average frequencies. Fixation indices, indicative of genetic differentiation, are the variances normalised by average allele frequencies. The overall F_ST is the mean of the fixation indices from both populations. Note: this formula is extended based on number of populations selected by the user*

Complementing the FST calculation, the get_population_avg_frequencies function retrieves average allele frequencies from a database. Utilising SQLAlchemy for database interaction, it constructs a dictionary mapping population name to their respective allele frequencies, facilitating quick access for the FST calculations. The interpretation of FST values is made accessible through the save_matrix_to_csv function, which uses the pandas library to transform the FST matrix into a structured CSV file. This file is saved in a static directory, ready for researchers to download and explore further. Visual analytics are provided by the generate_heatmap function, which takes the FST matrix and creates a heatmap using seaborn and matplotlib. This visual representation is saved as an image file, which researchers can use to swiftly discern patterns of genetic differentiation across populations. The integration of these computational components is orchestrated by the compute_fst Flask route. This endpoint handles POST requests from the web interface, invoking the functions to calculate FST, save the results, and generate a heatmap. Upon completion, it returns the URLs for the downloadable CSV file and the heatmap image, thus linking sophisticated backend calculations with a user-friendly frontend experience.

## Integration and Workflow

The integration of functions and user interactions within our software allows for a comprehensive analysis and visualisation of complex genetic differentiation data. From the moment a user submits a request through the Data Retrieval interface, the process involves fetching allele frequencies, computing pairwise FST values, generating visual and downloadable results, and providing access to these results for further analysis or reporting. Our platform performs the FST calculation in real-time rather than retrieving precomputed values from the database. This design decision was based on several considerations, Genomic datasets are often extensive, and the number of pairwise comparisons between populations for FST calculations can be substantial. Precomputing and storing all possible combinations would lead to an exponential increase in database size, making it impractical and inefficient. Researchers may wish to apply specific parameters or models when computing FST that are unique to their research needs. On-demand calculations allow for such customisation, which would not be possible with a static precomputed approach.

## Data Retrieval Functionality

Our web application's business logic includes a sophisticated search_snps function that allows users to query SNP data based on specific criteria like SNP IDs, genomic coordinates, or gene names, coupled with population filters. This functionality is integrated with the frontend, by serialising the results and returning a JSON response which is received by the front end, enabling users to retrieve and interact with the data seamlessly, reflecting the search requests through dynamic data presentation.

# FRONT END STRUCTURE AND FUNCTIONALITY

The front-end of our web application is designed to provide a user-friendly interface, allowing for intuitive interaction with the genetic analysis tools. The structure is rooted within the web_app directory, under the templates and static subdirectories.

## Templates

The templates directory contains HTML files that serve as the templates for the web application. Each template is crafted to present data and receive user input efficiently.

*index.html* - Serves as the entry point to the web application. It provides users with an overview of the functionalities available and guides them through starting their analysis.

*dataretrieval.html* - Facilitates the retrieval of SNP data based on user-defined criteria. It is the interface for submitting queries and displaying results.

*admixtures.html* - Allows users to explore genetic admixture analyses. This template is designed to present complex genetic information in a digestible format.

*clustering.html* - Used for visualising genetic clustering results. It provides interactive elements to manipulate and understand the clustering data.

## Integration of CSS and JavaScript in HTML Templates

Our application's user interface is crafted using HTML templates, which are enhanced by CSS and JavaScript to deliver a visually appealing and interactive experience. CSS is responsible for the look and feel of the web pages. It defines the colour schemes, layouts, fonts, and responsive design features that make our interface both functional and aesthetically pleasing. CSS helped us to create an interface that is intuitive and easy to navigate, ensuring that users can find what they're looking for without confusion. JavaScript adds interactivity and dynamic behaviour to web pages. JavaScript runs in the user's browser and handles tasks such as form validation, dynamic content updates, and user interaction without the need for a full page reload. As mentioned earlier, we use AJAX, which is based on JavaScript, to perform asynchronous server communications. This means that data can be sent and retrieved from the server in the background, allowing the user to continue using the application uninterrupted. JavaScript allows us to create rich, interactive elements such as drop-down menus, spinner elements, and custom controls that enhance the user experience. CSS and JavaScript work together within our HTML templates to produce a cohesive user experience. This combination is crucial for creating a modern web application that meets the expectations of today's users who are accustomed to sophisticated and responsive interfaces. By integrating CSS for styling and JavaScript for functionality into our HTML templates, we ensure that our application is not only informative and useful but also engaging and easy to use.

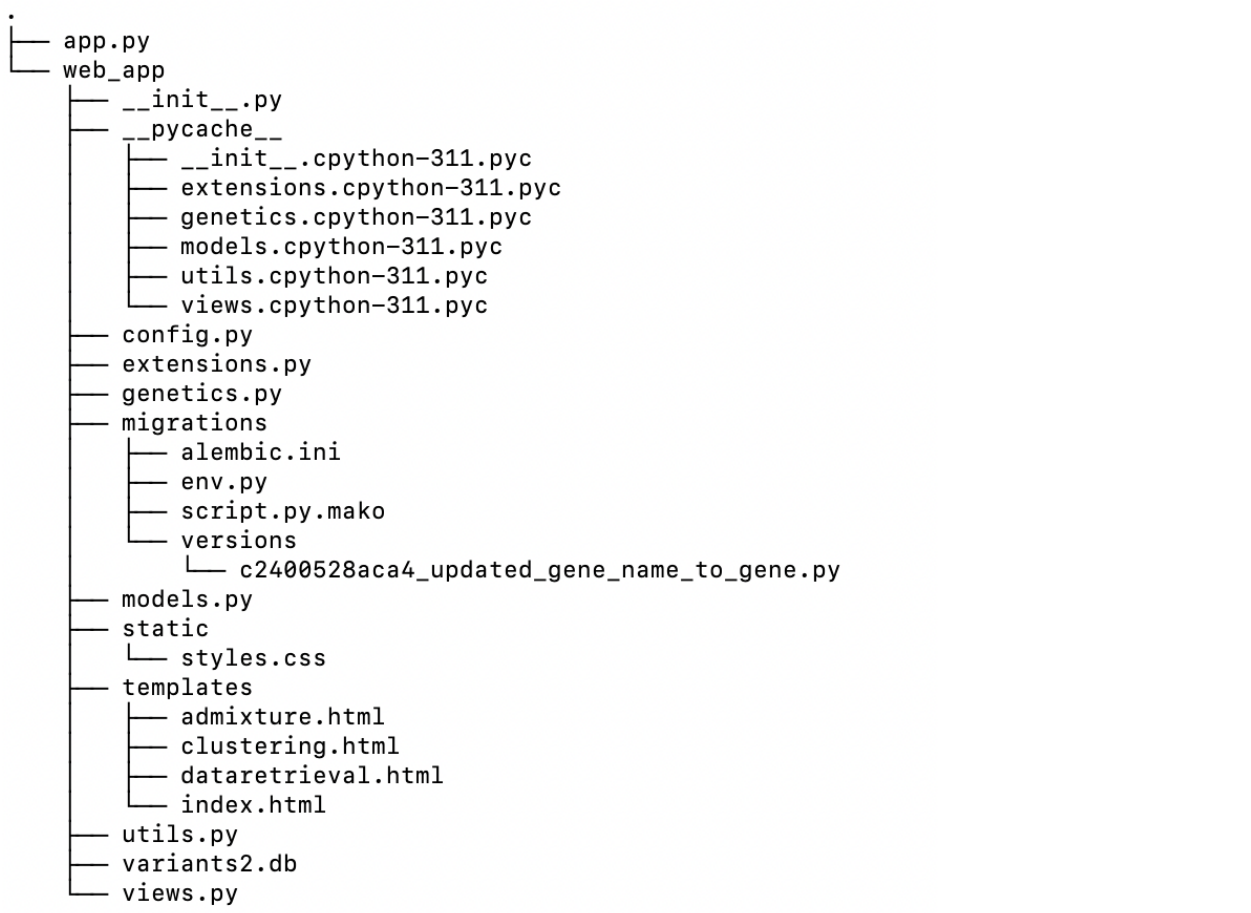


*Figure 7 – This diagram displays the file structure of a Flask web application, including Python scripts, compiled bytecode, configuration files, HTML templates, CSS stylesheets, and an SQLite database, organised within specific directories for application logic, database models, migrations, static content, and web templates.*

# CONFIGURATION AND ARCHITECTURE

The web application is structured following the Flask framework's best practices, with distinct responsibilities assigned to different components for clarity and maintainability. Here's how the main files and directories work together:

*app.py* - This is the entry point of the application. It's responsible for initialising the Flask app and bringing together all the components. It typically contains the setup needed to run the server, such as configuration loading, initialising extensions, and potentially setting up logging.

*__init__.py* - is crucial for Python package initialisation. In the context of a Flask application, this file contains the app creation and configuration logic. It sets up the application context and registers various components like Blueprints, database connections, and additional configurations that are be defined in config.py.

*views.py* – This file contains the blueprinted route definitions – the views that handle the user requests and serve the appropriate HTML templates or JSON responses. Each view function associated with a route gets data from the request, interacts with the model to retrieve, update, or insert data, and then renders a template or responds with data.

*genetics.py* - This is where the logic related to genetic data processing is defined, such as SNP analyses, FST calculations, and other genetics-specific computations. This file contains functions that are imported and used within views.py to handle requests related to genetic data operations.

*utils.py* - The utils.py file is used to store utility functions that might be needed across the application. These include helper functions for data formatting, error handling, and other common task that doesn't fit into a specific category like genetics or views.

*models.py* - This file defines the data models for the application, using SQLAlchemy ORM (Object-Relational Mapping). It represents the structure of the ‘analysis.db’ database with classes, defining tables, and their relationships, which the application uses to interact with the database.

*extensions.py* - This is used to instantiate and configure extensions that can be imported into other parts of the application. This helps to avoid circular imports and keeps the initialisation and configuration of extensions in a single location, which is a clean and modular approach.

When the application runs, *app.py* is executed, which initialises the Flask app and its configurations. The app then imports routes from views.py via Blueprints or direct imports. When a user makes a request, the Flask server determines which view function to execute based on the URL. The view functions may call upon genetic data processing functions defined in genetics.py to handle computations and data manipulation specific to genetic analyses. These functions might, in turn, use utility functions from utils.py for common tasks. Data models defined in *models.py* are used by *genetics.py* and *views.py* to interact with the database –

retrieving, records as needed to fulfil the user requests. This modular setup ensures a separation of concerns, making the application easier to maintain and scale. The clear distinction between each file's responsibilities allows developers to quickly locate and manage specific aspects of the application's functionality.

# BUSINESS LOGIC IMPLEMENTATION

Our web application's business logic is the layer where core operations, decision-making, and data processing occur. This includes handling user requests, interacting with the database, and executing genetic analysis algorithms. To ensure seamless operation and maintainability, we've implemented AJAX for front-end to back-end communication.

AJAX (Asynchronous JavaScript and XML) allows the web app to update asynchronously by exchanging data with the server in the background. This means users can continue to interact with the webpage without interruption while data is being fetched or submitted. By using AJAX, we can provide a smoother, more responsive user experience. Forms can be submitted, and data can be loaded without the need to refresh the page, which can be crucial for complex tasks such as genetic data analysis. AJAX is particularly useful for loading new content or updating existing content on a page without reloading it entirely. This is especially important in our application for tasks like retrieving SNP data, where the results must be displayed promptly without navigating away from the page. AJAX requests only send and retrieve the necessary data, reducing the amount of data transferred between client and server. This leads to faster interactions and reduced load on the server.

## Use of Blueprints

In our application, we utilise Blueprints to structure the application into distinct sections. Blueprints are a feature of Flask that allow us to create modular components within the web application. They can be thought of as mini applications that can be registered on the main application, making them an ideal choice for structuring and scaling applications. The Blueprint system facilitates a modular codebase, enabling us to compartmentalise features and services. By organising code into Blueprints, we keep our project tidy and manageable, ensuring that related views and operations are grouped logically. New features can be developed as separate Blueprints and integrated into the main application without significant restructuring. This modularity allowed for easier code reuse and sharing across different parts of the application. Each Blueprint acts as a standalone entity, with its own routes, templates, and static assets. This made the codebase easier to navigate and maintain.

## Integration of SQLAlchemy

SQL Alchemy is used to link the SQLite database to the Python interface. SQL Alchemy provides object-relational mapping (ORM), simplifying database access by abstracting the complexity of raw SQL queries and providing Pythonic interfaces for database operations.

The connection is unable by defining ORM models and using SQL Alchemy syntax to represent database tables as Python classes. Each class corresponds to a database table and attributes within the class represent columns in the table. In addition, the application is built by using the Flask web framework structured using Flask Blueprints to promote modularity and maintainability.

To integrate SQL Alchemy with Flask Blueprints, a separate module *modules.py* is created to define database models and manage the database session. This module is then imported and utilised within the blueprint's routes and views.

# LIMITATIONS AND FUTURE PROSPECTS

Our web application is dedicated to the analysis of human population genetics. However, our current approach to managing genetic data involves solely focussing on chromosome 1, as all SNPs provided to us in the VCF file are located there. This means that, while simplifying the analysis process, we potentially miss out on critical genetic variations present across other chromosomes, which could be essential for a more thorough understanding of the population genetics of our samples. A study on the coverage of Illumina arrays across various chromosomes emphasises the importance of expanding genetic data coverage to include additional chromosomes. By broadening our dataset beyond chromosome 1, we can achieve a more comprehensive analysis, uncovering richer insights into genetic diversity and structure. [3]

For our clustering analysis, if we had more time, it would be informative to try more types of analysis such as MDS and UMAP for clustering which could build on or provide alternative interpretations to our current dataset. The use of additional admixture software such as STRUCTURE to predict alternative ancestries would also be beneficial, as there is no perfect algorithm for admixture due to each software employing its own method of ancestry prediction.

Regarding our back-end database architecture, we currently employ SQLite and SQLAlchemy, a combination known for its efficiency in handling smaller datasets. However, this configuration may not maintain its efficacy if the dataset were to grow in the future, potentially limiting scalability and performance. Transitioning to a more robust database system, such as PostgreSQL, or implementing distributed database systems, could significantly enhance our application's ability to manage larger datasets effectively. Furthermore, the reliance on pre-calculated results for admixture and PCA analyses may not accurately reflect the most current data changes or allow users to customise their data visualisations extensively. Incorporating interactive visualisation tools like ChromoMap could enhance real-time analysis capabilities, enabling more dynamic and personalised exploration of genetic data. ChromoMap's innovative approach to visualising multi-omics data across chromosomes could provide an exemplary framework for improving our application's visualisation and analysis features, making it more responsive and versatile [4].

The software currently does not fully leverage the capabilities of comprehensive bioinformatics software for network analysis such as Cystoscape which could shed light on the relationships between different genes between individuals. By mapping these networks, researchers could identify key genetic markers and their interactions that contribute to populations specific traits.

We want to deploy our software on Amazon Elastic Beanstalk in the future in order to have an online version of our application, and we believe that it would be the best choice as it is a quick process and has a straightforward web console for configuration, as well as a pay-as-you-go pricing model, and could allow us to integrate other amazon web services easily, such as amazon S3 (Simple Storage Service).

In conclusion, our web application represents a significant step forward in the analysis of human population genetics, leveraging modern technologies to simplify complex genetic data interpretation. Despite its innovative approach, the application encounters limitations, notably in data scope and database scalability. Future development paths include expanding genetic data coverage beyond chromosome 1, integrating advanced bioinformatics tools like Cytoscape for enriched network analysis, trying new types of analyses and upgrading to more scalable database solutions. These enhancements aim to elevate the application's analytical depth, user customisation, and performance, ensuring it remains a valuable resource in the evolving field of genetic research.

# REFERENCES

[1] D.H. Alexander, J. Novembre, and K. Lange. Fast model-based estimation of ancestry in unrelated individuals. Genome Research, 19:1655–1664, 2009.

[2] H. Zhou, D. H. Alexander, and K. Lange. A quasi-Newton method for accelerating the convergence of iterative optimization algorithms. Statistics and Computing, 2009

[3] Mägi R, Morris AP. GWAMA: software for genome-wide association meta-analysis. BMC Bioinformatics. 2010;11:288.

[4] Lakshay A, Sreenivasulu N. ChromoMap: an R package for interactive visualization of multi-omics data and annotation of chromosomes. BMC Bioinformatics. 2021;22(1):23