Module code: BIO727P

Bioinformatics Software Development Group Project 2025

Bionic Beavers
Student SID numbers: 241078765, 240657752,
200119492, 200371300, 231113715

MSc Bioinformatics

Table of Contents

1 INTRODUCTION

Type 2 Diabetes (T2D) is a cardiovascular genetic disease affecting roughly half a billion people worldwide. According to the NHS, in 2021-2022 almost 250,000 adults in England were diagnosed with T2D. Genetic susceptibility to T2D varies across ancestries with some populations experiencing a higher incidence rate. For instance, south Asians develop T2D early in life despite having a normal body mass index (BMI).

This application has been developed by the Bionic Beavers, a group of five MSc Bioinformatics students at Queen Mary University of London. The purpose of the project is to produce a functioning protype of a web-based tool to examine T2D molecular data in south Asian populations.

The software application is located within GitHub at the following location:

https://github.com/japh140/Bioinformatics_Group_Project

To install the application

1) Check minimum software versions as per section 5.2 of the documentation file "Project Documentation.pdf".
2) Browse to the GitHub download page at https://download-directory.github.io/
3) Paste the software location URL into the box and press enter.
4) Unzip the package into an installation folder of your choosing.
5) Open a terminal window, change directory to your installation folder, locate the file run.py and type "python run.py"
6) Open a browser and type in the following URL: http://127.0.0.1:5000

A full description of the application with more complete installation instructions including prerequisite software requirements can be found in the word document "Project documentation.pdf".

2 ACKNOWLEDGEMENTS

The Bionic Beavers would like to thank Professor Conrad Bessant and Dr Matteo Fumagalli of Queen Mary University of London for their guidance in producing this application.

3 ABBREVIATIONS

API   Application Programming Interface

AJAX   Asynchronous JavaScript and XM

BMI   Body Mass Index

CSS   Cascading Style Sheets

FST   Fixation Index

HTML   Hypertext Mark-up Language

nSL   Number of Segregating sites by Length

PNG   Portable Network Graphic

SNP   Single Nucleotide Polymorphism

T2D   Type 2 Diabetes

TSV   Tab Separated Values

TXT   Text File Document

4 TABLES AND FIGURES

5 INSTALLATION

5.1 Software Location

The application package is located in GitHub at the following location.

https://github.com/japh140/Bioinformatics_Group_Project

5.2 Minimum System Requirements

Software packages support should be at the following minimum versions.

Table 1 – Minimum System Requirements

Package	Version
blinker	1.9.0
click	8.1.8
colorama	0.4.6
Flask	3.1.0
Flask-SQLAlchemy	3.1.1
greenlet	3.1.1
Flask_wtf	3.2.1
itsdangerous	2.2.0
Jinja2	3.1.5
MarkupSafe	3.0.2
pandas	2.2.3
Python	3.10.11
setuptools	75.1.0
typing_extensions	4.12.2
Werkzeug	3.1.3
wheel	0.44.0
wtforms	3.2.1
wtforms-validators	1.0.0

It is recommended that operating systems and browsers should be at the manufacturers current versions.

5.3 Installation

5.3.1 Windows Installation

1) Create a new directory in your local file system where you wish to install the package.

2) Browse to https://download-directory.github.io/

3) Paste the following URL
https://github.com/japh140/Bioinformatics_Group_Project into the text box. A zip file of the package will be downloaded.

4) Unpack the zip file into your new directory.

5) Open a windows command prompt and change directory to the new local directory. Change directory again into the directory that was downloaded.

6) In the command prompt type "python run.py"

7) Open a browser and type in the following URL: http://127.0.0.1:5000

5.3.2 Mac Installation

1) Create a new folder in your local file system where you wish to install the package.

2) Browse to https://download-directory.github.io/

3) Paste the following URL
https://github.com/japh140/Bioinformatics_Group_Project into the text box. A zip file of the package will be downloaded.

4) Unpack the zip file into your new directory.

5) Open a terminal window and change folder to the new local folder. Then change folder again into the folder that was downloaded.

6) In the terminal window type "python run.py"

7) Open a browser and type in the following URL: http://127.0.0.1:5000

5.3.3 Linux Installation

1) Create a new directory in your local file system where you wish to install the package.

2) Browse to https://download-directory.github.io/

3) Paste the following URL
   https://github.com/japh140/Bioinformatics_Group_Project into the text box. A
   zip file of the package will be downloaded.

4) Unpack the zip file into your new directory.

5) Change directory to the new local directory. Change directory again into the
   directory that was downloaded.

6) Type "python run.py"

7) Open a browser and type in the following URL: http://127.0.0.1:5000

6 SOFTWARE ARCHITECTURE

The software architecture design philosophy follows the well-established front end/back end methodology, with the backend comprising three layers and the front end a single layer (Figure 1). The frontend is implemented using Flask Hypertext Markup Language (HTML) templates, cascading style sheets (CSS) and Jinja. The Backend logic and web serving is implemented using Flask and the database Application Programming Interface (API) uses SQL queries to query the database. All of these are implemented in Python and use is made of the blueprints feature to keep components separate and more manageable. The database is implemented using SQLite with DBBrowser as the management tool. As it is just 700KB in size, it has been placed within GitHUb for convenience.
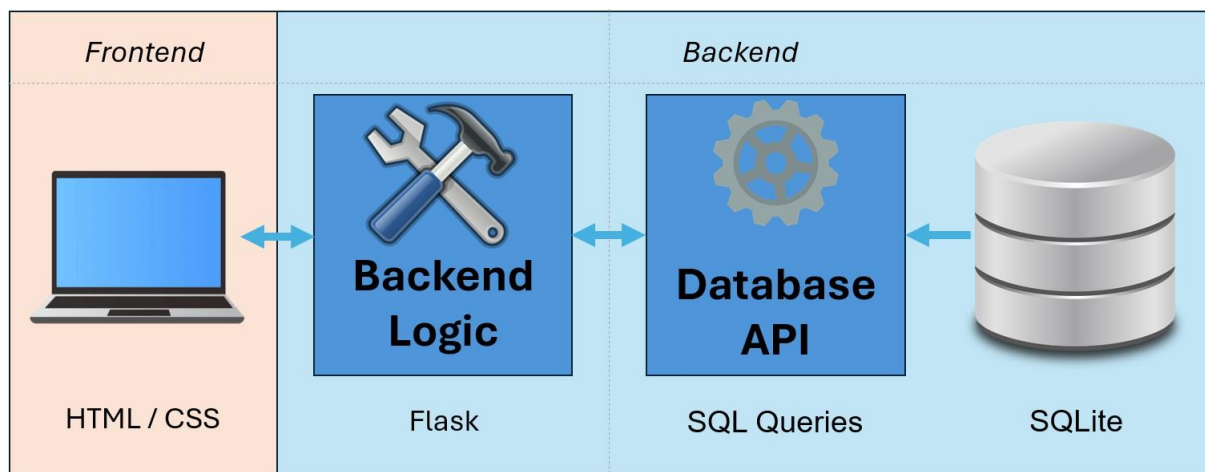
Figure 1 – Application Architecture

6.1 Frontend

The front end is implemented using Flask HTML templates and some Jinja, with styles dictated by CSS. All pages feature a "home page" feature whereby clicking on the banner returns to the home page. This feature has been implemented for ease of navigation, as the home page is the most important page to navigate to in this application due to it being where the search function is found.

6.1.1 Home Page

index.html displays the home page (Figure 2).
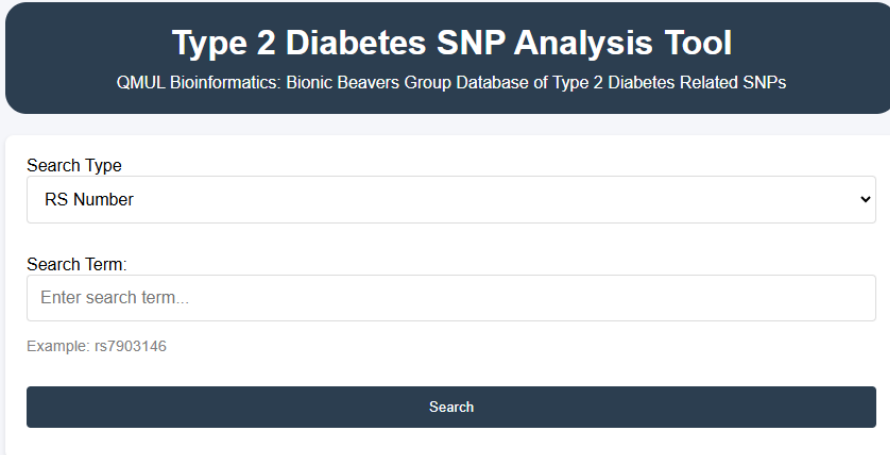
Figure 2 – Home Page

The home page features:
- Picklist search type selection (RS number, genome coordinates, gene name) for ease of use
- Dynamic example search format (implemented using JavaScript) to provide examples for demonstration
- Input validation for each search type ensures data quality and provides user guidance directly on screen (Figure 3).
    - RS numbers must follow format "rsNNNNN"
    - Coordinates accept both single position (chrN:position) and range (chrN:start-end)
    - Gene names must begin with a letter and can contain letters, numbers, underscores, or hyphens

Figure 3 – Home Page Input Validation

6.1.2 No Results Page

If no results are found *search_error.html* displays an error page (Figure 4) with features such as:

- Clear error message
- Specific feedback for what kind of error and the input that caused it
- Option to return to previous page

Across all error templates, a consistent error handling system is implemented to ensure clear feedback to users guiding them to correct their input format, maintain ease of navigation, and to present error information in a user-friendly manner.

Figure 4 – No Results Found Page

6.1.3 Results Page

If there are results, they are displayed by *results.html* (Figure 5), grouped by SNP with:

- Search type and search term
- SNP details (ID, chromosome, position)
- p-values associated with SNP data
- Mapped genes with links to gene information

The results are grouped this way to maintain clear data organisation, show multiple p--values per SNP as per the structure of the database, allow easy comparison between populations and to support statistics visualisation.
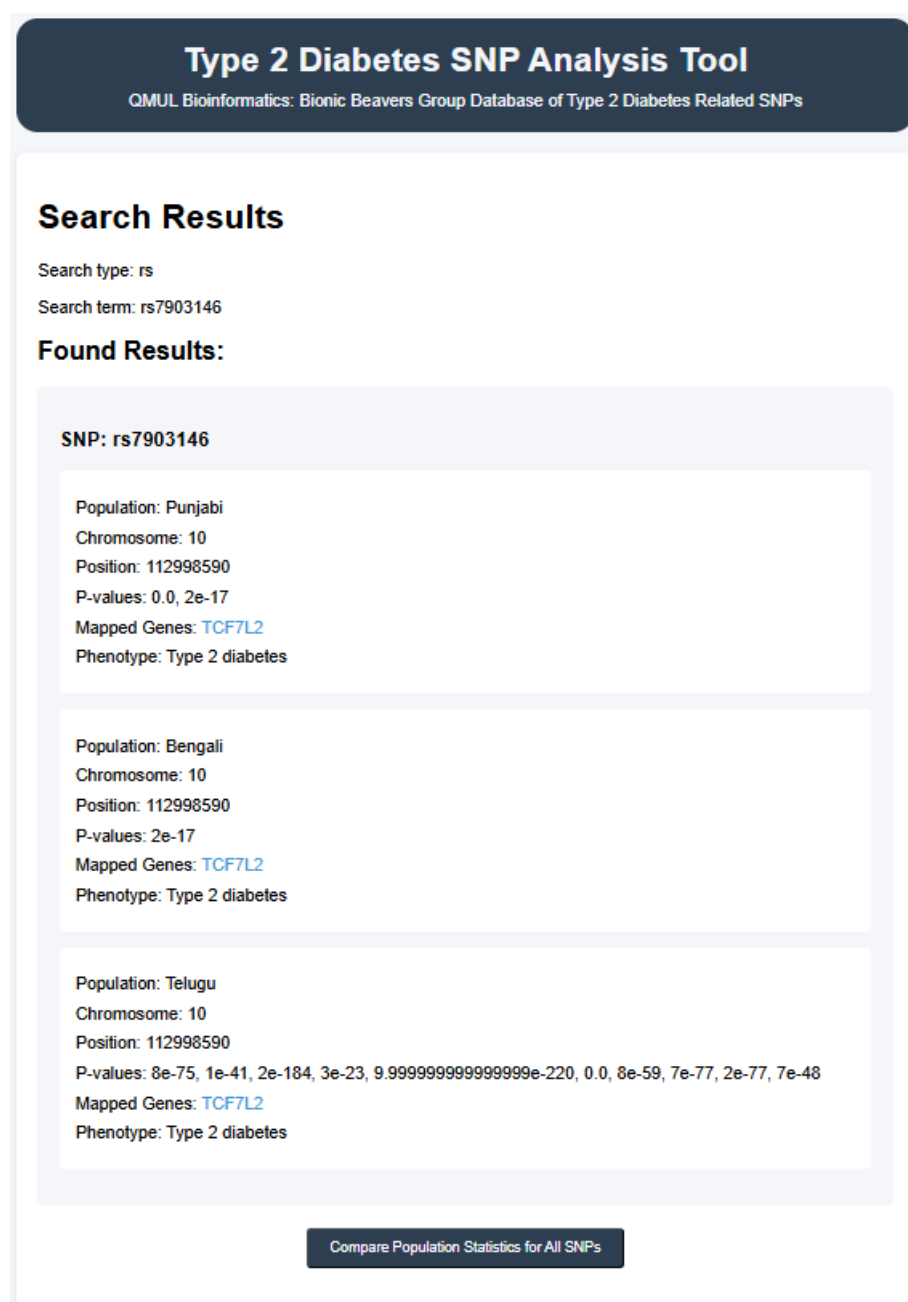
Figure 5 – Search Results Page

6.1.4 Mapped Gene Page

Clicking on the Mapped Gene link on the results page invokes *gene.html* (Figure 6).

This shows in a clear and concise manner:

- The gene details and annotations
- Associated pathways
- Ontology terms

Figure 6 – Mapped Genes Page

6.1.5 No Mapped Gene Page

If there are no mapped gene results, *gene_error.html* displays an error (Figure 7). It has:

- Consistent styling with the rest of the application
- Clear error message and feedback on the error
- Navigation options
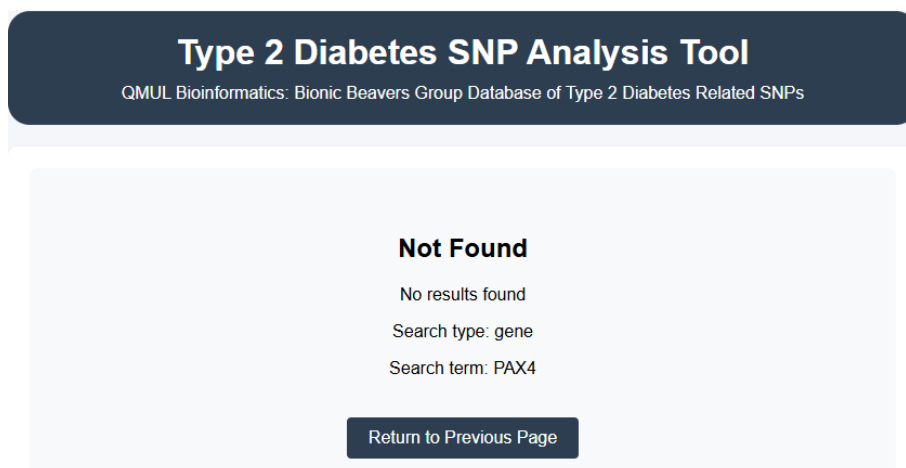
Figure 7 – No Mapped Genes Error Page

6.1.6 Population Statistics Page

Clicking on the "Compare Population Statistics for All SNPs" button on the result page invokes the page *population_comparison.html.* This page consists of the four sections highlighted in colour in Figure 8 (Populations Selected, Positive Selection Statistics, FST/nSL Interpretation and Visualisation) and described thereafter.
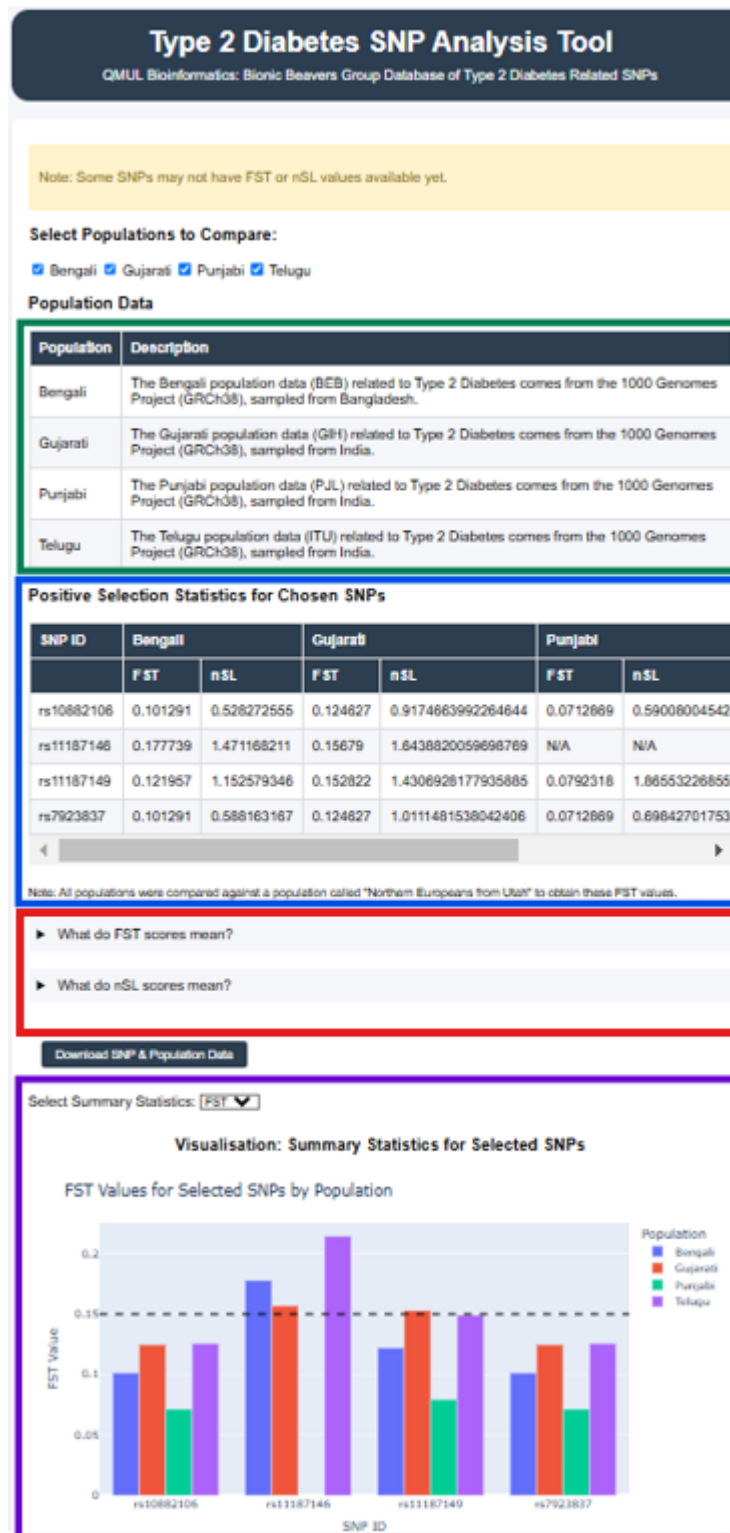
Figure 8 – Population Comparison Page

The user can select from South Asian populations—Bengali, Gujarati, Punjabi, and Telugu—at the top of the page. By default, all populations are selected when the user first accesses the page, however, they can uncheck any selection, which will dynamically update the Statistics Table, Description Table, and plots.

Tables displayed on the Population Statistics page are:
- The description table, which provides details on the source of the data for each population and the corresponding population code (Figure 9).
- A table displaying fixation index (FST) and single-population haplotype-based statistic (nSL) values for all SNPs within the selected genomic region. The table updates dynamically based on the populations selected. If an FST or nSL value is unavailable, the table will display 'N/A' (Figure 10). In addition, if no value is available, there will be a warning sign at the top of the page that informs the user that some FST or nSL values may be currently unavailable so the user can understand that the N/A values are caused by them being missing from the database and not for any other reasons.
- A collapsable table includes interpretations of FST and nSL values (Figure 11 and Figure 12).

Select Populations to Compare:

☑ Bengali ☑ Gujarati ☑ Punjabi ☑ Telugu

Population Data

Population	Description
Bengali	The Bengali population data (BEB) related to Type 2 Diabetes comes from the 1000 Genomes Project (GRCh38), sampled from Bangladesh.
Gujarati	The Gujarati population data (GIH) related to Type 2 Diabetes comes from the 1000 Genomes Project (GRCh38), sampled from India.
Punjabi	The Punjabi population data (PJL) related to Type 2 Diabetes comes from the 1000 Genomes Project (GRCh38), sampled from India.
Telugu	The Telugu population data (ITU) related to Type 2 Diabetes comes from the 1000 Genomes Project (GRCh38), sampled from India.

Figure 9 - Descriptions of Populations Selected

SNP ID	Bengali		Gujarati	
	FST	nSL	FST	nSL
rs10882106	0.101291	0.528272555	0.124627	0.9174663992264644
rs11187146	0.177739	1.471168211	0.15679	1.6438820059698769
rs11187149	0.121957	1.152579346	0.152822	1.4306928177935885
rs7923837	0.101291	0.588163167	0.124627	1.0111481538042406

Note: All populations were compared against a population called "Northern Europeans from Utah" to obtain these FST values.

Figure 10 – Positive Selection Statistics

▼ What do FST scores mean?

FST (Fixation Index) is a measure of population differentiation due to genetic structure. It shows how different populations are from each other genetically. The table below is how we decided to filter the data frame with FST statistics.

Any FST statistic with a negative score is attributed to sample noise and low sample sizes. We have converted those to 0 for the purposes of this tool.

FST Value Ranges:

FST Range	Interpretation
0.00 - 0.05	Little genetic differentiation, no strong selection
0.05 - 0.15	Moderate genetic differentiation, possible local adaptation, weak selection
0.15 - 0.25	Great genetic differentiation, likely positive selection in one population
>0.25	Very great genetic differentiation, strong evidence of local adaptation or selection pressure

Figure 11 – FST Score Interpretation Summary

▼ What do nSL scores mean?

nSL (number of segregating sites by length) is a haplotype-based selection statistic used to detect recent natural selection in a population. It looks at how long stretches of DNA are shared between individuals, assuming that beneficial mutations spread quickly, reducing genetic variation around them.

For the purposes of this tool, nSL scores have been normalised. Since raw nSL values can vary between genomic regions, normalisation helps compare values across different loci.

nSL Value Ranges:

nSL Range	Interpretation
> 2	Strong evidence of positive selection (recent beneficial mutation spreading).
1 to 2	Moderate evidence of positive selection
-1 to 1	Neutral - no strong selection signal
-2 to -1	Moderate evidence of balancing selection or recombination.
<-2	Strong evidence of balancing selection or high recombination rates.

Figure 12 – nSL Value Interpretation Summary

A drop-down menu allows the user to select which statistics to visualise. The grouped bar charts display all SNPs in the chosen genomic region and update dynamically based on the selected populations. Each bar represents the value (FST or nSL) for a specific SNP in a given population. The bars are grouped by SNP ID, and different populations are represented by different colours. Users can hover over the graph to view accurate values for the bars. On the top-right corner of the graph, a menu allows the user to download the plot in .png format. Users can also zoom in or out of the graph, and once zoomed, they can select 'Reset Axes' to return to the original graph size (Figure 13 and Figure 14). If no valid data is available, an error message is returned, and no plot is displayed (Figure 15).

Figure 13 – Visualisation of FST Positive Selection Statistics

Figure 14 - Visualisation of nSL Positive Selection Statistics

Select Summary Statistics: [FST ⌄]

Visualisation: Summary Statistics for Selected SNPs

Error loading plot. Please try again.

Figure 15 - Error Message When no Data is Available

The download button generates a .txt file containing the SNPs within the searched genomic region, along with the chromosome number, chromosomal position, association p-value(s), mapped gene, population, superpopulation name, FST and nSL values. Additionally, the file includes the average and standard deviation of the summary statistics (Figure 16).

```
SNP ID     Chromosome Position    P-value    Mapped Genes    Phenotype    Population FST
Value nSL Value
rs7903146  10    112998590  0.0    TCF7L2     Type 2 diabetes  Bengali     0.0    -
1.081003328
rs7903146  10    112998590  0.0    TCF7L2     Type 2 diabetes  Gujarati    N/A    N/A
rs7903146  10    112998590  0.0    TCF7L2     Type 2 diabetes  Punjabi     N/A    N/A
rs7903146  10    112998590  0.0    TCF7L2     Type 2 diabetes  Telugu      N/A    N/A


                                   FST      nSL
Mean                               0.0000   -1.0810
Standard Deviation                 0.0000   0.0000
```

Figure 16 - Example Download File

6.2 Backend

6.2.1 Database

The database consists of three tables (Figure 17).
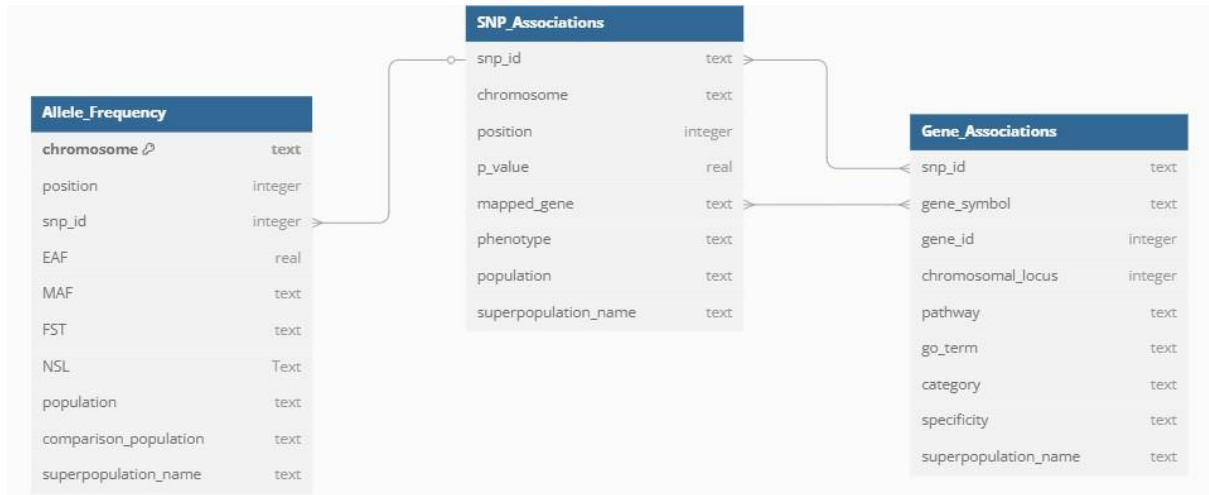
Figure 17 – Database Schema

Each table is partitioned with a field called *superpopulation_name*. In all tables the super population is set to *South Asian Ancestry*. When retrieving data, the database API will only retrieve data where the super population in the table is equal to the global variable *SUPER_POPULATION*, which is initialised to *South Asian Ancestry* in the primary configuration file *config.py*. This partitioning enables the application to be relatively easily expanded to other super populations by simply changing the value of the global variable and adding the relevant data to the database tables.

There are no primary keys defined as the *snp_id* and *population* fields do not contain unique values. A description of the data contained withing the tables can be found in section 7.

6.2.2 Database API

The database API services requests from the backend logic module and interrogates the database. Results are returned as a data frame, which is empty if no results are found. The functionality of the database API is governed by three global variables contained in the *config.py* file.

Table 2 – Database API Global Variables

Global Variable	Function	Default Value
DATABASE_PATH	The name and location of the database file.	app/project.db
SUPER_POPULATION	The super population partition to use for all queries. See section 6.2.1	South Asian Ancestry
QUERY_LIMIT	The maximum number of results returned by the API for any query. This prevents the frontend form becoming overwhelmed.	25

Functions implemented by the API are:

Table 3 – Database API Functions

Function Name	Purpose
get_snp_by_id(<snp_id>)	Returns a data frame of results (snp_id, chromosome, position, p_value, mapped_gene, phenotype, population) when querying table SNP_Associations by <snp_id>
get_snp_by_gene(<mapped_gene>)	Returns a data frame of results (snp_id, chromosome, position, p_value, mapped_gene, phenotype, population) when querying table SNP_Associations by <mapped_gene>)
get_snp_by_coordinates (<chromosome>, <start>, <end>)	Returns a data frame of results (snp_id, chromosome, position, p_value, mapped_gene, phenotype, population) when querying table SNP_Associations for chromosome <chromosome> between and including positions <start> and <end>

Function Name	Purpose
get_gene_annotations_by_gene_symbol (<gene_symbol>)	Returns a data frame of results (gene_symbol, gene_id, chromosomal_locus, snp_id, pathway, go_term, category, specificity) when querying table Gene_Annotations by <gene_symbol>
get_gene_annotations_by_gene_id (<gene_id>)	Returns a data frame of results (gene_symbol, gene_id, chromosomal_locus, snp_id, pathway, go_term, category, specificity) when querying table Gene_Annotations by <gene_id>
get_gene_annotations_by_snp (<snp_id>)	Returns a data frame of results (gene_symbol, gene_id, chromosomal_locus, snp_id, pathway, go_term, category, specificity) when querying table Gene_Annotations by <snp_id>
get_snp_and_gene_by_snp (<snp_id>)	Returns a data frame of results (snp_associations_snp_id, chromosome, position, p_value, mapped_gene, phenotype, population, gene_symbol, gene_id, chromosomal_locus, gene_annotations_snp_id, pathway, go_term, category, specificity) when querying table SNP_Associations by <snp_id> left joined to table Gene_Annotations
get_allele_frequency_by_snp (<snp_id>)	Returns a data frame of results (snp_id, FST, population, EAF, MAF) when querying table Allele_Frequency by <snp_id>
get_fst_by_population (<population_name>)	Returns a data frame of results (snp_id, FST, population) when querying table Allele_Frequency by <population_name>
get_fst_value_by_snp_for_empty_population (<snp_id>)	Returns a data frame of results (snp_id, FST) when querying table Allele_Frequency for an empty or null population
get_stats_by_snp_and_population (<snp_id>, <population>, <comparisonpopulation>)	Returns a data frame of results (snp_id, FST, population, NSL) when querying table Allele_Frequency by <snp_id>, <population> and <comparisonpopulation>

Not all API functions are necessarily used by the application. Some were created in anticipation of their use and have been retained for testing purposes. All are tested by the database API test suite (section 6.2.3).

There is no need to open and close the database. Each time that any of the above functions are called, the local context variable *dbconnection* within the Flask request context *g* is checked to see if it is defined. If it is not defined a connection to the database is made using the *sqlite3.connect* function and the connection context is stored in *g*. If *dbconnection* is defined, the existing database connection is used. Queries to the database are then made using pandas *pd.read_sql_query().* This uses direct SQL queries, not SQLAlchemy as this provides greater flexibility.

As the database queries are being initiated by a human it is good practice to close the database after each query in order to release resources as there could be several minutes or hours between requests. The database is connection is closed automatically by use of the *app.teardown_request* feature. When registered, this causes the function *close_app_connection()* to be called after each app request, and this closes the database connection.

6.2.3 Database API Testing

Included within the package is a test suite program *dba_api_test_suite.py*. Running this test suite checks the interface between Flask and the SQLite database, calling each of the functions listed in Table 3. The test suite checks the number of rows and columns returned by each function and also the column names and their order. It also confirms that each function returns an empty frame if there are no results. Note that the test suite is data dependent in that it checks for the number of results returned in the data frame, so if additional data is added the test suite may need revision. The primary purpose of the test suite is for regression testing during development. Before and after any alteration to the database API software, backend software or alterations to the database, the test suite was run to ensure upwards compatibility.

An example extract from a successful test suite run is show in Figure 18.

```
TESTING: get_snp_by_id()
      snp_id chromosome   position      p_value mapped_gene         phenotype population
0  rs12334481          8   71514886  2.000000e-10        EYA1  Type 2 diabetes      Telugu
1  rs12345069          9   94208893  6.000000e-10    LINC02603  Type 2 diabetes      Telugu
      2  rs12380322          9  19074540 1.000000e-09        HAUS6 Type 2 diabetes       Telugu
Empty DataFrame
Columns: [snp_id, chromosome, position, p_value, mapped_gene, phenotype, population]
Index: []


TESTING: get_snp_by_gene()
      .   .   .   .
EVERYTHING IS WORKING
```

Figure 18 – Example Output from Database Testing Program

```
TESTING: get_snp_by_id()
```

```
Columns: [snp_id, chromosome, position, p_value, mapped_gene, phenotype, population]
```

6.3 Backend Logic

Database logic is divided over seven code blocks contained within three python files, *views.py*, *genes.py* and *plot.py*. These implement the frontend functionality and interface to the database through the database API.

6.3.1 Home Page Logic

views.py implements the home page search functionality provided by *index.html* (section 6.1.1). User input is validated, and on-screen error messages are shown if the user input is invalid or if no results are returned by the database API. Error messages are displayed by the frontend as error pages or simple error messages. User input is checked against a regex to see if it is valid input for the type of search selected. The three different regexes used are *'^rs\d+$'* (for RS Number), *'^chr([1-9]|1[0-9]|2[0-2]|[XYM]):\d+-\d+$'* or *'^chr([1-9]|1[0-9]|2[0-2]|[XYM]):\d+$'* (for genomic coordinates) and *'^[A-Za-z][A-Za-z0-9_-]*$'* for (gene name)

The SNPSearchForm() class defines a Flask form that prompts the user for a search method (RS Number, Gene Name, or Genomic Coordinates) and a search term (Figure 2). The template uses the snp_ids_added variable to keep track of the SNP IDs that have already been added to the hidden input fields. As SNP results are displayed, each unique SNP ID is stored in a hidden input field with the name snp_ids, which is dynamically added to the form for submission. These hidden inputs are submitted when the form is sent to the backend.

The function *validate_search_term()* validates the frontend supplied search method and search term by use of regular expressions. If an error is detected, a validation error message is displayed in red on the home page (Figure 3).

The route *snp_bp.route('/')* handles the logic behind the home page *index.html*. It causes the home page search form to be displayed and validates the input. If correct input has been supplied then control is redirected to the results page *results.html* (section 6.1.3), otherwise control remains with the home page *index.html*.

6.3.2 Results Page Logic

The route *snp_bp.route('/search/<search_type>/<search_term>')* section of *views.py* implements the logic behind the results page *results.html* (section 6.1.3) and the no results page. *search_error.html* (section 6.1.2).

Using the search method and search term previously captured, SNP results will be retrieved from the database using either the database API function *get_snp_by_id()*, *get_snp_by_gene()* or *get_snp_by_coordinates()* as appropriate. If no results are found, or any other error is detected, then control is redirected to the no results page *search_error.html* with appropriate messages. If results are found, then they are built into a data frame and displayed by the results page *results.html*.

The results page includes two main links: one for mapped genes, which redirects to *gene.html*, and another for population comparisons, which redirects to *population_comparison.html* (section 6.3.4). When the user submits the population comparison form, the SNP IDs (one for each unique SNP displayed in the results) are transferred to the backend via a POST request to the */population-comparison* route. The form contains hidden input fields for each SNP ID, which are dynamically added based on the results shown on the page. This enables the backend to receive the list of SNP IDs that the user has viewed and wants to compare across populations.

6.3.3 Mapped Genes Page Logic

genes.py implements the logic behind the mapped gene page *gene.html* (section 6.1.4) and the no mapped gene page *gene_error.html* (section 6.1.5).

Route *gene_bp.route('/gene/<gene_symbol>')* uses the database API function *get_gene_annotations_by_gene_symbol()* to retrieve gene results from the database table *gene_associations*. If there are no results user is redirected to the "No Mapped Gene" page *gene_error.html* and an error screen is shown (Figure 7). If there are results, any duplicates are dropped and the mapped gene page *gene.html* shows the Gene Symbol, Gene ID, Pathway, Gene Ontology term (GO term) and category which depicts the phenotype (Figure 6).

6.3.4   Population Statistics Page Logic

The route *snp_bp.route('/population-comparison')* section of *views.py* implements the logic behind the population statistics page *population_comparison.html* (section 6.1.6).

6.3.4.1   Population Description Table

An HTML form in the *population_comparison.html* allows the user to select the populations they wish to compare. The checkboxes for each population have the checked attribute set by default, meaning these populations are selected when the form is initially rendered. This ensures that both the Population Description and Statistics tables are populated with data upon initial load, preventing empty tables if the user does not select any populations.

The Population Description table is updated dynamically based on the populations selected in the form. The descriptions for each population are stored in a JavaScript object called *populationDescriptions*. An event listener is attached to the checkboxes, which listens for changes such as when a user checks or unchecks a population. When a change occurs, the event listener triggers several functions, one of which is *updatePopulationTable().* This function checks which populations are currently selected, clears the previous content of the table, and dynamically generates and adds rows for the selected populations and their corresponding descriptions.

The selected values are retrieved using the *querySelectorAll* method, which identifies all checked checkboxes. These values are then stored in a JavaScript variable called *selectedPopulations*, allowing the table to be updated in real-time based on the user's selection. Finally, the list of selected populations is stored in a form variable called *selected_population*, which is sent to the backend route *snp_bp.route('/population-comparison')* for use in retrieving relevant data for the Statistics table.

6.3.4.2   Statistics Table

The Statistics Table is built by first gathering the data from the selected SNP IDs and populations in *snp_bp.route('/population-comparison')* and then presenting it in a structured format for the user.

An HTML form in *results.html* sends the list of SNP IDs from the results page to the *snp_bp.route('/population-comparison')* route when submitted (section 6.3.2). The snp_ids represent the SNPs collected on the results page, and the selected_populations from *population_comparison.html* (section 6.3.4) contains the populations the user has chosen for comparison. Input validation ensures that both SNP IDs and selected populations are provided. If either is missing, an appropriate error (such as a *ValueError*) is raised. For each selected population, the program fetches the FST and nSL values for each SNP ID using the *db.get_stats_by_snp_and_population* function from Database API (section 6.2.2). If these values are available, they are added to the *combined_data* dictionary. If not, 'N/A' is assigned to the SNP and population. The fetched statistics (FST and NSL data) are stored in the session which allows the data to be accessible by *plot.py* (section 06.3.5) and *download.py* (section 6.3.6). Once the data is gathered, the *population_comparison.html* template is rendered. The template is provided with the fetched FST and NSL values, as well as the SNP IDs and selected populations for display.

The table in the *statistics-table* section of *population_comparison.html* displays SNPs and their corresponding statistic values for the selected populations. The table is populated dynamically based on the user's selections. An event listener is attached to the population checkboxes, which triggers the *updateTableHeaders* and *updateFSTTable* functions whenever the population selection changes. Initially, the table headers (SNP ID and column headers for each population) are added dynamically based on the selected populations. This is done in the *updateTableHeaders* function. It creates a header for each population, and under each population header, two sub-headers (FST and nSL) are added. The *updateFSTTable* function dynamically populates the table with FST and nSL values for each SNP in the selected populations. It first retrieves the FST and nSL data from the backend via Jinja and stores it in the JavaScript variable *fstData*. It clears any existing rows in the table and then iterates through the SNP IDs available for the first selected population, populating the table with FST and nSL values for each SNP across all selected populations.

6.3.5 Plot Logic

plot.py implements the logic to generate a visualisation of the FST and nSL data while working with the frontend.

The *population_comparison.html* template has a dropdown menu for summary statistics selection and checkboxes for populations (section 6.3.4.1). JavaScript listens for changes in the population checkboxes and the statistics dropdown. When the user selects or deselects populations or changes the selected statistic, the frontend updates the plot. When the user interacts with these two forms (checkboxes or dropdown), JavaScript collects the selected populations and the statistic and sends an AJAX request to the backend to fetch the plot data. The request is either sent to */plot-fst* or */plot-nsl* based on the selected statistic. Once the data is returned from the backend, the plot is inserted into the plot-container div.

There are two main routes for the plots – *plot_bp.route('/plot-fst')* which creates a bar chart to visualise the FST values and */plot-nsl* which visualises nSL values. The backend receives a POST request with the selected populations from the *population_comparison.html* template. Using JavaScript for population selection instead of sessions allows the selected populations to be dynamically updated, providing a smoother experience. Storing populations in a session would keep the initial list intact, even if the user deselects some, leading to inconsistency. The plot route fetches SNP IDs from *session(snp_ids)* and the necessary data from the *session(combined_data)* from *snp_bp.route('/population-comparison')* (section 6.3.4.2) and processes it to extract the relevant FST or nSL values for the selected populations and SNPs. The backend ensures that valid data exists for the selected populations and SNPs. If the necessary data for the selected populations is unavailable, an error message is shown, and the plot is not generated. The backend prepares a list of dictionaries, where each dictionary contains an SNP ID, population, and the corresponding value of either FST or nSL. This data is converted into a Pandas data frame, which is then used to create an interactive grouped bar plot using *Plotly Express*. *Plotly* offers built-in interactivity by default. Users can hover over data points, zoom in/out, pan, and interact with the plot in many ways without needing to write extra code. The figure is returned to the frontend and embedded in the plot-container div using JavaScript.

6.3.6 File Downloads

The views.py file includes a route called */download_snp_data*, which is triggered when a user clicks the "Download SNP Data" button on the Population Comparison page.

This route retrieves data stored in the session from a previous request, including FST, nSL, and SNP (as described in section 6.3.4.2). Similar to the plot logic (section 6.3.5), it also receives a POST request with the selected populations from the *population_comparison.html* template. If any required data is missing, the function raises an error. *StringIO* is used as an in-memory file to temporarily store the table data. This enables the program to construct the content in memory before sending it to the user as a downloadable file. SNP and population data are retrieved using the *get_snp_by_id()* function from the database API (section 6.2.2). For each population, the corresponding FST and nSL values are retrieved. If any data is missing, "N/A" is written for the respective fields. The download process is then managed by creating a Blob (Binary Large Object) URL and simulating a click on a hidden <a> element to trigger the file download.

The average and standard deviation of the FST and nSL values are calculated across all SNPs and populations using Numpy. Once the content is constructed in *StringIO*, the cursor is reset to the beginning, and the data is converted to bytes. The *send_file* function from Flask is then used to send the file as a response, enabling the user to download the SNP data.

6.4 Tools Used

Apart from Python, the primary tools to implement the application were Flask and SQLite. Flask provides a small framework that easily meets the requirements of the project. One of the project requirements was that the application use a database rather than a flat files such as tab separated files (TSV). To this end SQLite was selected as it is a simple file based system that can easily be published in file systems such as google drive, or even within GitHub if it is small enough.

Tools used to develop the application were

- GitHub – For code management.
- DBBrowser – To build and manage the database.
- Visual Studio Code – To develop and test the code for the database API.
- PyCharm – For code writing in developing the front end and some of the back end.
- Apocrita high performance computing facility – For the calculation of FST values.

6.5 Limitations

Limitations have to be put in the context of the application being a prototype study and not a public tool. The primary limitation of the application is the volume of data it holds as good data on South Asian populations, split out into Bengali, Gujarati, Telugu, Tamil and Punjabi is difficult to obtain. However, if the volume of data were to increase significantly, computer systems more power than laptops would be needed.

From a system's point of view, the application has no security implemented and would be open for anyone to use, or for them to take the data.

From a data point of view, limitations identified were:

- SNP Density Variation: Some genomic regions may have sparse variant coverage, affecting nSL accuracy.
- Demographic Effects: Population structure (e.g., admixture, bottlenecks) can mimic selection signatures.
- Filtering Trade-offs: Removing multi-allelic sites ensures consistency but may exclude complex selection signals.

6.6 Future Expansion

The application could be relatively easily expanded to other populations by adding additional data to the database and by using the database partition feature described in section 6.2.1. Additional expansion could include greater exploitation of data visualisation.

From a data point of view, future improvement could include:

- Additional Selection Metrics – Integrating iHS, XP-EHH, and FST for a broader selection analysis.
- Expended T2D associated SNPs dataset- Incorporate additional GWAS studies from diverse global populations to improve variant representation.
- Broader Population Coverage – Extending analysis to other ancestries.
- Optimized Functional Annotation – Setting up a local Ensembl mirror for faster annotation.
- Interactive Visualization – Developing a web-based interface to explore selection signals.

7 Database Collection and Processing

7.1 Selection of T2D Phenotypes

Gouda et al. (2021) performed a review of recent findings of phenotypes associated with type 2 diabetes (T2D) and categorised them into five groups. The major phenotypes identified were severe insulin resistance, severe insulin deficiency, mild obesity, high BMI and mild age related diabetes.

In a study by Shoily et al. (2021) the phenotypes of diabetes, cardiovascular disease, inflammation, hypertension and kidney diseases were examined, and four SNPs were found to be associated: rs5186, rs1800795, rs1799983 and rs1800629.

Another study by Willer et al. (2007) examined 134 SNPs from a literature review of 120 published studies looking at SNPs associated with type 2 diabetes. They reported significant associations between type 2 diabetes and the following phenotypes: rs5210, rs5219, rs5400, rs1044498, rs1800610, rs1800795, rs1801262, rs1801282, rs2071023, rs2701175, rs3856806.

7.2 Table "SNP Associations"

The SNP associations table data was built using data from the GWAS catalogue. The All Associations v1.0 file (gwas_catalog_v1.0-associations_e113_r2025-01-08.tsv) was downloaded and inserted raw into a SQL database. The table was then filtered on the column DISEASE_TRAIT for the phenotypes identified by Pishoy et al. (2021) and also for the specific SNPs identified by Shoily et al. (2021) and Willer et al. (2007). Further filtering on the column INITIAL_SAMPLE_SIZE for Pakistani (Punjabi), Bangladesh (Bangali) or South Asian (Telugu). Extracted just the data for the phenotypes of interest associated with type 2 diabetes and for three South Asian populations, a total of 1192 rows of data.

7.2.1 SNP Associations Structure

"snp_id"	TEXT,
"chromosome"	TEXT,
"position"	INTEGER,
"p_value"	REAL,
"mapped_gene"	TEXT,
"phenotype"	TEXT,
"population"	TEXT,
"superpopulation_name"	TEXT

7.3 Table "Gene Associations"

Studies by Gupta et al. (2011) and Nair et al. (2010) were cross referenced to compile a list of gene associations. Particular attention was given to ensuring relevance for South Asian populations while also incorporating globally validated T2D-associated loci. A total of 40 rows of data were added to the table.

7.3.1 Gene Associations Structure

"gene_symbol"	TEXT,
"gene_id"	INTEGER,
"chromosomal_locus"	INTEGER,
"snp_id"	TEXT,
"pathway"	TEXT,
"go_term"	TEXT,
"category"	TEXT,
"specificity"	TEXT,
"superpopulation_name"	TEXT

7.4 Table "Allele Frequency"

7.4.1 Population Genomic Data

We sourced whole-genome sequencing data from the 1000 Genomes Project (Phase 3, GRCh37/hg19). This dataset was chosen for its broad population coverage, high variant resolution, and well-structured metadata. SNP data was retrieved in VCF format, covering autosomal chromosomes (1–22).

Tool Used: *wget*

Purpose: Download large genomic datasets directly from FTP repositories.

Rationale: The 1000 Genomes Project provides pre-phased genomic VCF files, which streamline downstream analyses, such as variant filtering, population structure assessment, and selection scans. Using wget allows direct access to these high-quality datasets, ensuring integrity and reproducibility in data acquisition.

7.4.2 Population-Specific Filtering

To extract T2D-relevant populations, sample metadata was retrieved from the 1000 Genomes sample panel. We then filtered the dataset for specific ancestry groups, such as South Asians (e.g., Bengali from Bangladesh - BEB).

Tool Used: *bcftools*

Purpose: Extract specific populations based on sample metadata, ensuring we analyse the correct cohort.

Rationale: Filtering for South Asian populations enables targeted selection scans, allowing us to identify genetic variants under positive selection that may be linked to T2D susceptibility. Using bcftools ensures an efficient and accurate extraction process, preserving the integrity of the dataset while maintaining consistency with population metadata.

T2D-Associated SNP Integration

T2D-linked SNPs were gathered from multiple GWAS sources, including:

- GWAS Catalog
- DIAMANTE 2022
- Genes & Health 2022
- T2DGGI 2024

These datasets provided SNP IDs, genomic positions, and association p-values, which were merged to create a comprehensive T2D variant list.

Tools Used: Datasets were retrieved locally via download from the T2D Knowledge Portal and *Pandas* was used for data integration, deduplication, and merging.

Purpose: Ensured the dataset contained only validated, T2D-associated SNPs.

Rationale: Combining multiple GWAS sources enhances data reliability and ensures that only statistically significant T2D-linked variants are retained. The inclusion of large sample size datasets increases the statistical power of the analysis. We specifically focused on South Asian populations to identify T2D-associated genetic variants under positive selection, ensuring findings are relevant to the genetic and evolutionary factors influencing T2D risk in this ancestry group. Pandas streamlines integration and deduplication, providing a clean, high-confidence dataset for further analysis.

7.4.3 Reference Genome Alignment (LiftOver)

The 1000 Genomes data was aligned to GRCh37, while some GWAS datasets used GRCh38.

Tool Used: *liftOver*

Purpose: To standardise SNP positions, genomic coordinates.by Converting SNP coordinates from GRCh38 to GRCh37, preventing mismatches that could lead to incorrect analyses.

Rationale: Standardising genomic coordinates ensures accurate SNP mapping onto our VCF files, maintaining compatibility with the reference genome. This prevents positional discrepancies that could distort association signals, ensuring reliable downstream analyses of selection and disease associations.

7.4.4 Filtering: T2D-Associated SNPs

The genomic datasets were further refined by extracting T2D-linked variants for autosomal chromosomes. This was done to each autosomal chromosome prior to merging for each population.

Tools Used: *vcftools*

Purpose: to filter by chromosome positions found in T2D-linked SNPs datasets.

Rationale: Filtering ensures high-confidence T2D-associated SNPs are retained, preserving data integrity and enabling accurate genotype analysis across populations for selection and association studies.

7.4.5 VCF Merging & Indexing

To streamline the dataset, autosomal chromosomes were merged into a single VCF file, reducing redundancy and ensuring efficient processing.

Tools Used: *bcftools concat*, – Merges multiple chromosome-level VCFs.

            tabix - Indexes VCF files to allow fast querying.

Purpose: Enables quick retrieval and processing of genomic data for further filtering. bcftools concat was used to merges multiple chromosome-level VCFs and tabix was used to index VCF files to allow fast querying.

Rationale: Easy accessibility of genomic regions to extract genomic data.

7.4.6 Final Filtering: T2D-Associated SNPs

From the merged autosomal dataset, we extracted only T2D-linked variants, ensuring SNPs with no insertions or deletions and Biallelic sites with no multi-allelic variants.

Tools Used: *bcftools view* – Filters by SNP type (excludes indels).

                   awk – Identifies multi-allelic sites for removal.

Purpose: Guarantees statistical compatibility for nSL computations.

Rationale: Filtering ensures biallelic SNPs, aligning with selection tests that assume only ancestral and derived alleles, enhancing nSL accuracy.

7.4.7   Detecting positive selection by FST

FST statistics measures genetic differentiation between populations based on allele frequency found in genetic polymorphism data. To assess potential selection pressure in South Asian populations, FSTs were calculated against European populations.

Tools Used: *vcftools*

Purpose: To calculates FST scores

Rationale: Weir and Cockerham' (1984) is widely cited and adopted by vcftools. It measures the genetic differentiation between populations.

7.4.8   Detecting Positive Selection Using nSL

nSL is a haplotype based statistics that measures how long haplotypes remain intact before recombining, with higher values suggesting recent positive selection. Since selection pressures vary by ancestry, population-specific nSL scores were calculated.

Tools Used: *Selscan* – Calculates nSL scores from haplotype data.

                  Selink (Python) – Calculates nSL scores from haplotype data.

Purpose: Identifies selection signatures in T2D-linked SNPs.

Rationale: The nSL statistic does not require genomic location information (e.g., a genetic map), unlike other haplotype-based statistics such as iHS. Selscan is a robust and optimised tool for detecting haplotype-based selection signals, making it well-suited for efficient and accurate nSL calculations.

7.4.9 Normalisation and Statistical Significance (nSL scores)

Raw nSL scores can vary significantly across chromosomes, populations, and genomic regions due to differences in recombination rates, mutation hotspots, and demographic history. To ensure comparability, normalization was applied.

Z-score transformation adjusts scores by centering them around the mean and scaling them by the standard deviation. This allows for direct comparisons across genomic regions and populations, making it easier to identify outliers that may indicate positive selection.

Tools Used: *Pandas* – Data transformation.

SciPy – Z-score normalization and p-value computations.

Purpose: Ensures statistical comparability across populations.

Rationale: Easier interpretation of positive selection signal per population for cross population comparisons

7.4.10 Gene Annotation and Functional Insights

To determine biological significance, SNPs were mapped to nearest genes using the Ensembl REST API. This helped infer potential effects on T2D-related pathways, such as insulin secretion and glucose metabolism.

Tools Used: curl (Ensembl API queries) – Retrieves gene annotations.

Purpose: Links selection to the closet SNPs

Rationale: Helps to investigate gene association to SNPs

7.4.11 Tools and Versions

Tools and version used in the data collection and processing were as follows.

Table 4 – Tools used in the Data Collection and Processing

Package	Version
wget	1.21.3
curl	7.88.1
bcftools	1.7
VCFtools	0.1.16
tabix	1.7
LiftOver	1.3.0
Pandas	2.2.3
Selscan	2.0.2
NumPy	2.2.2
SciPy	1.15

7.4.12 Allele Frequency Table Structure

"chromosome"   TEXT,

"position"   INTEGER,

"snp_id"   TEXT,

"EAF"   REAL,

"MAF"   REAL,

"FST"   REAL,

"population"   TEXT,

"comparison_population"   TEXT,

"superpopulation_name"   TEXT

8 References

1000 Genomes Project (n.d.) International Genome Sample Resource. Available at: https://www.internationalgenome.org/ [Accessed 23 Feb. 2025].

Gene Ontology Consortium (n.d.) Gene Ontology Resource. Available at: https://geneontology.org/ [Accessed 23 Feb. 2025].

Gouda, P., Zheng, S., Peters, T., Fudim, M., Randhawa, V.K., Ezekowitz, J., Mavrakanas, T.A., Giannetti, N., Tsoukas, M., Lopes, R. and Sharma, A., 2021. Clinical phenotypes in patients with type 2 diabetes mellitus: characteristics, cardiovascular outcomes and treatment strategies. Current Heart Failure Reports, 18(5), pp.253-263.

GWAS Catalog (n.d.) NHGRI-EBI GWAS Catalog. Available at: http://www.ebi.ac.uk/gwas [Accessed 23 Feb. 2025].

National Health Service (NHS), 2021. Overview - Crohn's disease, https://www.nhs.uk/conditions/crohns-disease/ [Accessed 10 December 2024]

Shoily, S.S., Ahsan, T., Fatema, K. and Sajib, A.A., 2021. Common genetic variants and pathways in diabetes and associated complications and vulnerability of populations with different ethnic origins. Scientific Reports, 11(1), p.7504.

T2D Knowledge Portal (n.d.) Type 2 Diabetes Knowledge Portal. Available at: https://t2d.hugeamp.org/ [Accessed 23 Feb. 2025].

Weir, B.S. and Cockerham, C.C., 1984. Estimating F-statistics for the analysis of population structure. evolution, pp.1358-1370.

Willer, C.J., Bonnycastle, L.L., Conneely, K.N., Duren, W.L., Jackson, A.U., Scott,
  L.J., Narisu, N., Chines, P.S., Skol, A., Stringham, H.M. and Petrie, J., 2007.
  Screening of 134 single nucleotide polymorphisms (SNPs) previously
  associated with type 2 diabetes replicates association with 12 SNPs in nine
  genes. Diabetes, 56(1), pp.256-264.