# Team Google Chrome: Type I Diabetes Web Application - Documentation.

## Software Philosophy

Genome-wide association studies (GWAS) are useful to identify large numbers of genetic variants associated with diseases. However, this information can be overwhelming to parse through, and related information is difficult to source. This software was developed in order to make SNP (single nucleotide polymorphism) information related to Type 1 Diabetes Mellitus easier to access for clinicians. Thus when a clinician has a research question various genomic, functional, population, and clinical information can be accessed from one database. The software also provides analysis related to linkage disequilibrium and produces manhattan plots.

## Software Architecture

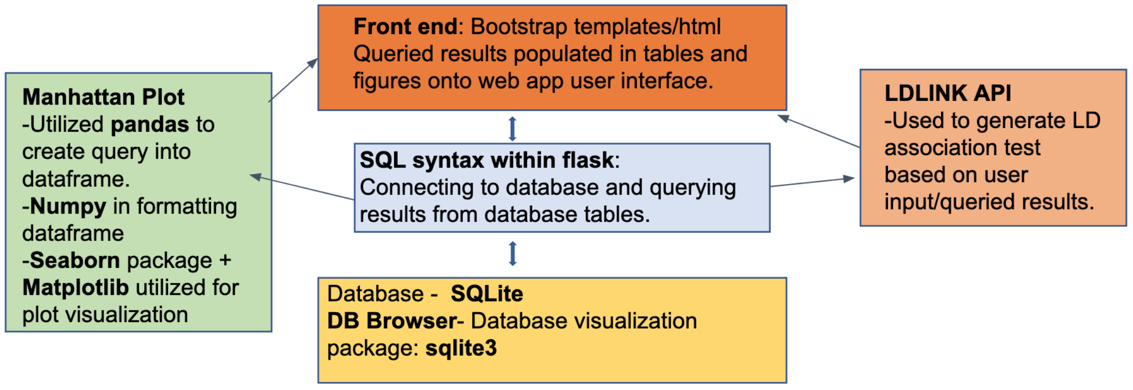

**Front end**: Bootstrap templates/html
Queried results populated in tables and figures onto web app user interface.
**Manhattan Plot**
-Utilized **pandas** to create query into dataframe.
-**Numpy** in formatting dataframe
-**Seaborn** package + **Matplotlib** utilized for plot visualization
**SQL syntax within flask**:
Connecting to database and querying results from database tables.
**LDLINK API**
-Used to generate LD association test based on user input/queried results.
Database - **SQLite**
**DB Browser**- Database visualization package: **sqlite3**

**Figure 1**: Figure showing the overview of the software architecture for the web application.

The architecture of the software is structured according to the framework above (figure 1). Our SQLite database stores all the data information that has been cleaned and structured into a schema according to the software requirements. In order to query and gather information, the web application framework is entirely built using Flask. To retrieve database information, Flask allows a connection to the SQLite database to query information from the tables. Once the queried information is retrieved, combined with HTML templates, the data is structured within tables on the user interface allowing easy access and interpretation for the clinician. Furthermore, the queried information is also used to generate plots such as a Manhattan plot of p-values and a linkage disequilibrium association heat map for rsID pairs of interest.

## Data Sources

The data for this project was sourced from four different websites, all funded by the government meaning the information is public access. The data for this project was restricted to information from chromosome 6 in order to build a functional database with a more manageable amount of data.

The GWAS information is sourced from the European Bioinformatics Institute (EBI) and organised into four columns; SNP rs ID, mapped gene, p-value, and coordinate location. In the GWAS information, duplicate rs IDs corresponded to the same SNP being verified by two different studies. For this software, the duplicate IDs were removed to streamline the database information and build a primary key. In a future version of the software, duplicate IDs could be kept in order to provide clinicians with the information that an SNP has been verified by multiple studies.

The gene ontology information is sourced from the BioMart tool on Ensembl using the mapped gene list from EBI. For each mapped gene there were potentially multiple gene ontology terms associated which were divided into three categories; biological process, cellular component, and molecular function.

The population data was sourced from the BCI SNPnexus website along with the CADD score information. For each SNP rs ID, data is available for different populations, ranging from the specific to the broad. It was determined that the broader population groups would be more beneficial to clinicians than specific subpopulations as the data can be used to look at general trends between populations. For this software, when entering a SNP rs ID, the alternative allele frequency for the five continent groups is returned. The software could be further developed to include all the subpopulations within the five main continent-based categories, with the ability to answer more specific research questions.

The CADD data included both the raw score and the adjusted PHRED score. A high score indicates a variant that is not stabilised by selection and is thus expected to be more often disease-causing than expected by random chance. The scores are returned for every type SNP for query made into the database. CADD scores were chosen as a way to represent SNP functional impact due to the variety of parameters utilised to create the score via a machine learning model. We believe this method would be most useful to clinicians using this application.

All the data types and their sources can be found in figure 2.

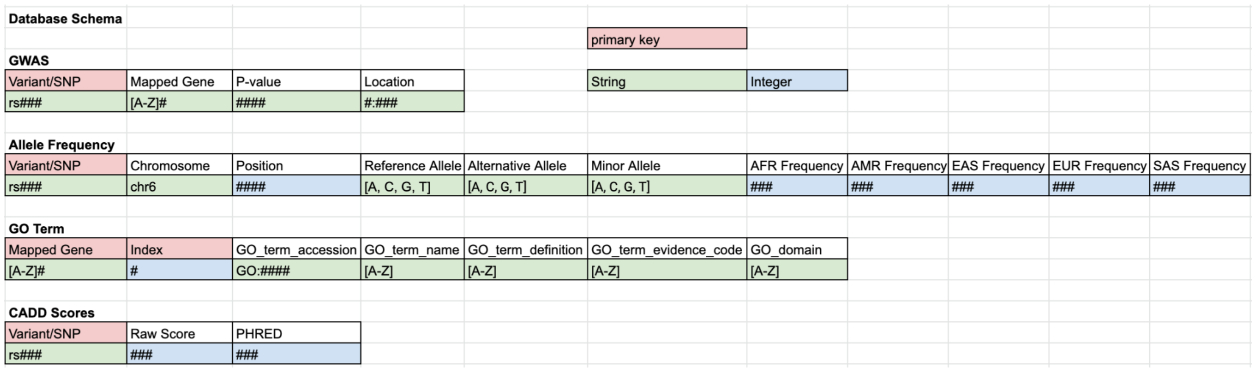
| Data type | Source | Link |
|---|---|---|
| rsID, mapped gene, p-value, chromosomal location | NHGRI-EBI GWAS catalogue | https://www.ebi.ac.uk/gwas/ |
| Gene Ontology accession codes, domains, names, definitions and evidence codes | Ensembl - Biomart Tool | https://www.ensembl.org/biomart/martview |
| CADD scores (raw and PHRED), population frequency data | SNPnexus | https://www.snp-nexus.org/v4 |
| LD values | LDlink - LDpop module | https://ldlink.nci.nih.gov/?tab=ldpop |

**Figure 2**: This table shows the sources and a link to access each type of data.

## Database Structure

The data was built using SQL because the language is easy to use, the data is all kept in one place, and multiple people can read and modify the data as opposed to building a data storage structure in a language like Python. The database was built in SQLite as opposed to other relational database management software because of its usability and considering the data size for this project. DB Browser was used to execute the SQL to build the database and remove any rows that contained null data. DB Browser was used as opposed to the command line in order to keep the code in one place and to easily visualise changes made to the database.

The data was split into four different tables because there are four data sources for the project and the rs IDs could be used to connect the tables as a primary key. The four tables contain the information for the GWAS data, allele frequency, GO terms, and the CADD scores. The schema structure is shown below (figure 3) where the primary keys for each column are highlighted in red and each data type is coloured according to whether the data was stored as a string, highlighted in green, or as an integer, highlighted in blue. The p-value column was stored as a string rather than an integer because the "x" present in the table did not easily convert to an integer using SQL; this problem was solved later on in order to produce the Manhattan plots. The location column was also stored as a string because of the ":" present in the data. In order to query the database correctly given chromosome coordinates this column was converted into a substring and cast as an integer.



**Database Schema**

| | |
|---|---|
| 🟥 | primary key |
| 🟩 | String |
| 🟦 | Integer |

**GWAS**

| Variant/SNP | Mapped Gene | P-value | Location |
|---|---|---|---|
| rs### | [A-Z]# | #### | #:### |

**Allele Frequency**

| Variant/SNP | Chromosome | Position | Reference Allele | Alternative Allele | Minor Allele | AFR Frequency | AMR Frequency | EAS Frequency | EUR Frequency | SAS Frequency |
|---|---|---|---|---|---|---|---|---|---|---|
| rs### | chr6 | #### | [A, C, G, T] | [A, C, G, T] | [A, C, G, T] | ### | ### | ### | ### | ### |

**GO Term**

| Mapped Gene | Index | GO_term_accession | GO_term_name | GO_term_definition | GO_term_evidence_code | GO_domain |
|---|---|---|---|---|---|---|
| [A-Z]# | # | GO:#### | [A-Z] | [A-Z] | [A-Z] | [A-Z] |

**CADD Scores**

| Variant/SNP | Raw Score | PHRED |
|---|---|---|
| rs### | ### | ### |

**Figure 3**: Schema for the SQLite database

The database can be queried using the rs IDs, chromosome number, gene name, and chromosome region coordinates. Each of the queries were tested in DB Browser to ensure that the queries would be successful when later used in the Flask application and that the correct information was returned.

The index column was introduced in the GO term table in order to build a primary key since there are many GO terms associated with each mapped gene. The index column is not displayed in the final output but was necessary to include in the building of the schema. In the GO term data for a portion of the mapped genes, there was only null data available, likely because there was no GO term information available for the gene. Returning tables full of null data for clinicians is useless, so these rows were removed from the original database.

## Software Structure

The software is written in Flask and utilises SQL queries in order to pull information correctly from the database depending on the user input. On the front page, the user has the option to submit four different types of input depending on what information they want to retrieve.

1. rs ID: If the user only wants information pertaining to the specific SNP they are studying then they have the option to query all the information needed using the rs ID. This will return a table containing the rs ID, mapped gene(s), p-value, chromosome number and position, the reference allele, the alternative allele, the minor allele, and the allele frequency for 5 significant geographic populations. The structure of this table is the base information returned for each query. Then for each mapped gene associated with the rs ID, a table will populate below the first containing the relevant GO term information.

2. Chromosome number: If the user wants to query the database for all information relevant to a particular chromosome they can enter “chr 6”. This will return all the rs IDs on that chromosome and the information mentioned above. As this is a prototype software the only information contained in it is from chromosome 6 but this could be expanded to every chromosome with further development.

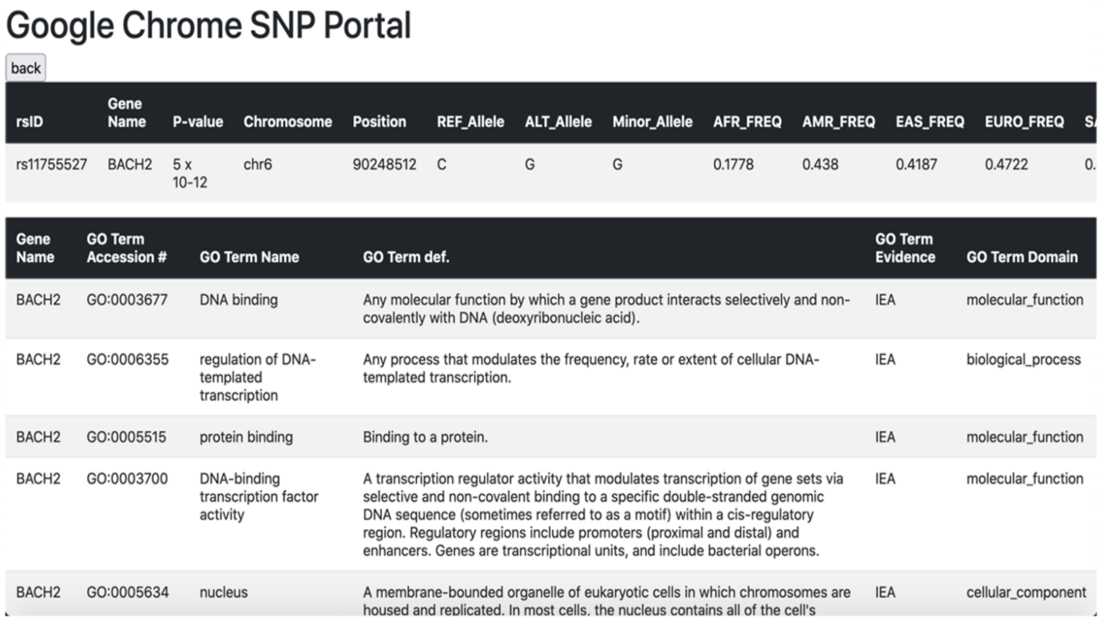
3. Chromosome position: If the user wants to query the database for a specific range on the chromosome they could type in “starting position,ending position” and this would return a table with every rs ID that is located between the two coordinates entered. As the database only has data from chromosome 6 this information is assumed and the user can omit the 6 identifier at the start of their query. For example, if the user wants to query between 6:32421478 and 6:170063801 they would type in “32421478,170063801” and a table would populate with all the information contained above.

4. Mapped gene: If the user wants to query the database for all the information pertaining to a mapped gene they need only type in the gene name and a table would populate with every SNP that maps to that particular gene and a table that contains all the information contained above.

## Flask

The web application was built using the software Flask. Flask was chosen as the micro-web framework because of its simplicity to use, as it has little or no dependencies on external libraries once installed. This is an optimal choice for an introductory-level web development application. Furthermore, through the utilisation of python libraries, Flask allows access to SQL databases for manipulation in data analysis and data wrangling. Lastly, Flask is a lightweight software ensuring that a vast amount of memory isn't required when building web applications from scratch.

Flask is used in the web application to route the users to specific web pages based on the user's input, query information from the SQLite database using SQL syntax, then render templates on the front end based on the results.



Google Chrome SNP Portal

| rsID | Gene Name | P-value | Chromosome | Position | REF_Allele | ALT_Allele | Minor_Allele | AFR_FREQ | AMR_FREQ | EAS_FREQ | EURO_FREQ | S |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| rs11755527 | BACH2 | 5 x 10-12 | chr6 | 90248512 | C | G | G | 0.1778 | 0.438 | 0.4187 | 0.4722 | 0. |

| Gene Name | GO Term Accession # | GO Term Name | GO Term def. | GO Term Evidence | GO Term Domain |
|---|---|---|---|---|---|
| BACH2 | GO:0003677 | DNA binding | Any molecular function by which a gene product interacts selectively and non-covalently with DNA (deoxyribonucleic acid). | IEA | molecular_function |
| BACH2 | GO:0006355 | regulation of DNA-templated transcription | Any process that modulates the frequency, rate or extent of cellular DNA-templated transcription. | IEA | biological_process |
| BACH2 | GO:0005515 | protein binding | Binding to a protein. | IEA | molecular_function |
| BACH2 | GO:0003700 | DNA-binding transcription factor activity | A transcription regulator activity that modulates transcription of gene sets via selective and non-covalent binding to a specific double-stranded genomic DNA sequence (sometimes referred to as a motif) within a cis-regulatory region. Regulatory regions include promoters (proximal and distal) and enhancers. Genes are transcriptional units, and include bacterial operons. | IEA | molecular_function |
| BACH2 | GO:0005634 | nucleus | A membrane-bounded organelle of eukaryotic cells in which chromosomes are housed and replicated. In most cells, the nucleus contains all of the cell's | IEA | cellular_component |

**Figure 4**: This table shows the sources and a link to access each type of data.

Furthermore, once the user input is redirected to the appropriate route, a connection is established and a cursor function is assigned to a variable in order to query information within our database. The SQL syntax 'SELECT' allows the program to retrieve information from the columns of all tables within our database and the 'INNER JOIN' syntax allows the combination of tables that contain the same values within the column. Based on the database structure, the primary keys and rsID allowed the 'INNER JOIN' function to work appropriately because of the common column names connecting each table. Further, to match the user input to the correct rsID, the 'WHERE' clause proved to be essential to filter out the database and correctly link the user's input with columns in the database. Figure 4 represents the search query output when a user inputs a rsID. The search result is the rsID and relevant clinical, population, and gene ontology information pertaining to the gene name associated with the queried rsID.

The limitations of this routing-based search criteria are that it only pertains to chromosome 6 information, as displayed in figure 4, rather than containing all information relating to every chromosome. The web application only includes GWAS information that specifically filtered out all chromosomal data except chromosome 6. Implementation of all chromosomal data should be included as well as a filtering feature at the top of the table in figure 4, to sort through which chromosome can be selected by the user. Therefore, to improve the web application in future development, the user input will need to query all SNP information for each chromosome. This would expand the user's potential when utilising the web application as a larger variety of data can be accessed.

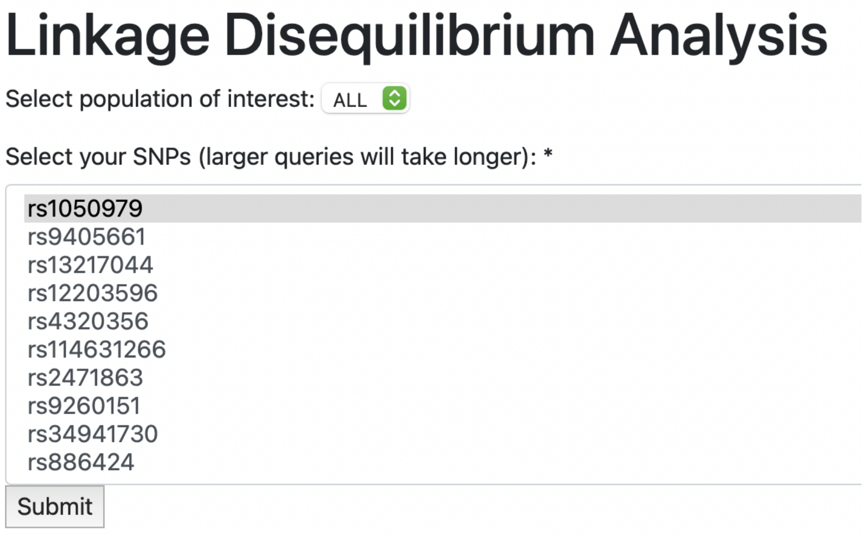
Another potential improvement to the software is to make the tables scrollable to make the website easier to read. At this point when a sizable amount of data is returned the user must scroll through a large table and manually look through all the information. If the tables were scrollable and filterable it would be easier for the user to access the information they are interested in using in their research.

## Linkage Disequilibrium Analysis

The web application has the capability to show linkage disequilibrium (LD) analysis for a selected list of SNPs. The user can select SNPs returned from their query into the database if there is more than one SNP returned (i.e. for chromosome, genomic position range, or mapped gene searches). Alongside the user's selection of SNPs, the user also has the ability to choose their population of interest for the LD analysis. The populations available match the population frequencies displayed in the table. There is also an 'ALL' option, which describes the LD value across all populations. Users just need to press the 'Submit' button to be redirected to the analysis page. The page displays a table containing the linkage disequilibrium values for each pair of SNPs, a heatmap describing these values, and a downloadable text file of a symmetric matrix of the values. The $r^2$ values, the correlation coefficient between pairs of loci, were chosen to represent the LD calculations due to their usefulness for population geneticists; allele frequencies are part of the returned database which may aid in their interpretation.

### Implementing LD analysis into Flask

The WTForms library was used to create the forms for users to input. An example of the form displayed to the user can be seen in figure 5. WTForms was chosen as it allowed SNP choices to be dynamic and responsive to the queries made by the user. The population choices were static due to them staying consistent across all queries.



Linkage Disequilibrium Analysis

Select population of interest: ALL

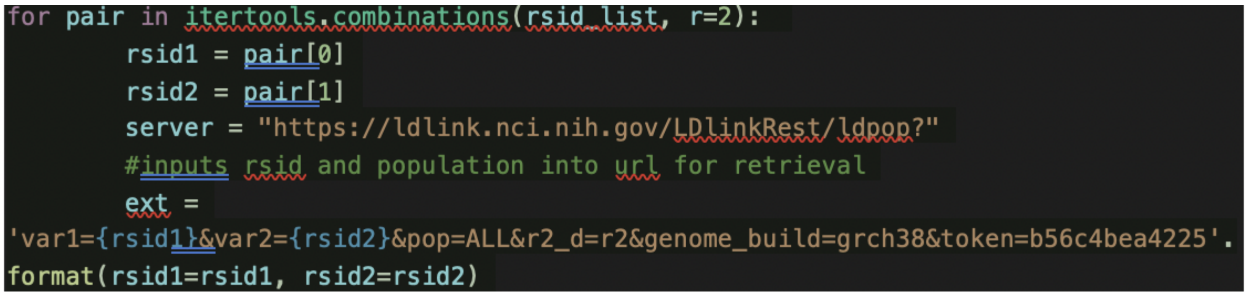
Select your SNPs (larger queries will take longer): *

rs1050979
rs9405661
rs13217044
rs12203596
rs4320356
rs114631266
rs2471863
rs9260151
rs34941730
rs886424

**Figure 5:** An example of the form that allows users to select their rsIDs and population of choice for subsequent LD analysis

## Retrieving LD values

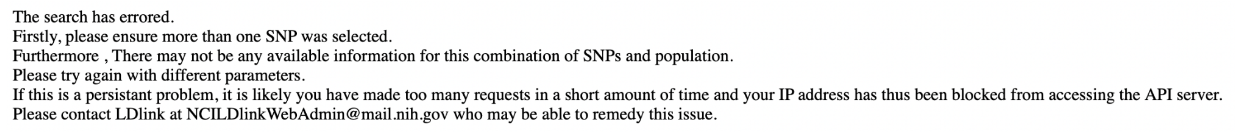
To produce the LD values, a python function was written which retrieved them via API access to the LDLink database provided by the National Cancer Institute, specifically the LDpop module offered by the website. This database was used as it utilises publicly available haplotypes from the 1000 Genome Project and contains the GRCh38 build, which is where other data in the web application is sourced from. The function takes a list of strings of rsIDs and the population of interest also as a string. For each list of rsIDs, every iteration of pairs is created using the itertools package method 'combinations'. This pair is then inputted into the API url and a request is made, This section of the script can be seen in figure 6. The response contents are then decoded from utf-8 format and transformed into a pandas dataframe. From there, the r$^2$ value for the specific population and pair of interest is isolated and inputted into a dictionary. This is then used to create the graph and downloadable file later on.



```
for pair in itertools.combinations(rsid_list, r=2):
        rsid1 = pair[0]
        rsid2 = pair[1]
        server = "https://ldlink.nci.nih.gov/LDlinkRest/ldpop?"
        #inputs rsid and population into url for retrieval
        ext =
'var1={rsid1}&var2={rsid2}&pop=ALL&r2_d=r2&genome_build=grch38&token=b56c4bea4225'.
format(rsid1=rsid1, rsid2=rsid2)
```

**Figure 6:** This code explains how the list of rsIDs returned by the query form was transformed and used to access the LDlink API.

A limitation of this method is that if multiple requests are made too often in a small time frame, the IP address of the user's computer is blocked by the server. This could result in the analysis section of the web app not being functional. If a search fails, the user is directed to a webpage encouraging them to try the search again with different parameters. There is also a link to the LDlink support email which should be contacted if the failed searches are a persistent problem, as it is likely their IP has been blocked. This error message can be seen in figure 7.



The search has errored.
Firstly, please ensure more than one SNP was selected.
Furthermore , There may not be any available information for this combination of SNPs and population.
Please try again with different parameters.
If this is a persistant problem, it is likely you have made too many requests in a short amount of time and your IP address has thus been blocked from accessing the API server.
Please contact LDlink at NCILDlinkWebAdmin@mail.nih.gov who may be able to remedy this issue.

**Figure 7:** The error message displayed to users if there is a query failure.

Future development should look at other ways of accessing LD values so this error would not occur. For example, making our own SQL database of $r^2$ values and extracting them from there. This would also improve another drawback of using API access, which is the time it takes to process long queries. Due to the script having to iterate through each pair of SNPs, with queries of more than ~10 SNPs this can become an issue. This is due to the iterations following a triangular number pattern (n(n-1)/2) meaning that a query of 20 SNPs means iterating through 190 pairs. Written into the script is a 5-second cooldown between each API query. This means for a query of 20 SNPs, even without incorporating the time it takes to actually run the scripts to access the API, produce the graph and make the downloadable file, the minimum wait time is ~16 minutes.

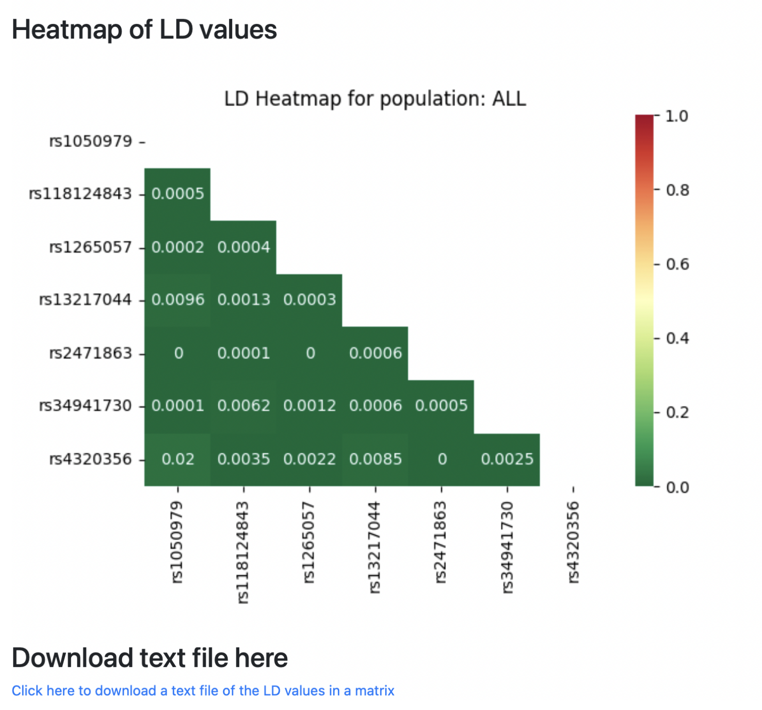
This is not ideal and could deter individuals from using our application. Creating our own database of LD values would mean a much faster retrieval process, and thus improve the efficiency of the web application. Displaying a loading bar or indication of progress would aid in informing users that their job is actually being completed, and not that the application has failed. However, a positive aspect of this method of accessing LD values is that it would be relatively easy to adjust the code to incorporate more than just the SNPs related to chromosome 6.

## Displaying and downloading the results

A triangular heatmap of SNPs identified via their rsID and their corresponding $r^2$ values is created upon a click of the submit button in the linkage disequilibrium section of the web application. A function takes the LD dictionary created during value retrieval and uses this as a parameter to create a heatmap using the seaborn library. To create the graph, the LD values, and the corresponding pair of SNP rsIDs are transformed into a symmetric matrix. Instances of pairs of the same rsID along the diagonal axis were given a value of 0. The cells are colour-coded, with a low $r^2$ value corresponding to green and high values to red. A colour key is included in the graph to help the user interpret the results of their query.

To display the graph via Flask, the graph is saved to a bytes object using the BytesIO() library and saved as a variable and encoded using base64 library. The graph is then decoded via utf-8 decoding when rendered on the html template. To better aid with visualisation, if the length of the dataframe used to make the graph is longer than 10 rsIDs, annotations are not included in the graph. To improve the graph, the SNPS are ordered via chromosomal position when not being queried from the region parameter.

The downloadable text file is in the form of a csv and is simply the pandas dataframe containing the symmetric matrix produced during the creation of the heatmap, outputted as a file. Flask implementation involved saving the file to a local folder, and then using a separate app route to send the csv file to the html page. All files are named LD_matrix.csv and replace each other upon creation. This is to save on storage space. An example of what is displayed on the analysis webpage can be seen in figure 9.



Heatmap of LD values

Download text file here

Click here to download a text file of the LD values in a matrix

**Figure** 9: Example of the Heatmap displayed as part of the LD analysis results (above), below is the link to the downloadable text file available for users to save to their local device.

## Manhattan Plot

The web application has the ability to display a Manhattan plot of p-values for SNPs located in a user-defined region on the chromosome. The plot will load on the web page alongside the

GWAS, population allele frequency, and GO terms for the SNPs that were returned within the stated range. The SNPs are positioned along the x-axis; ordered by the genomic position, chromosome number, and base pair position along the x-axis. The -$\log_{10}$ of the p-value is plotted along the y-axis. Each point on the graph is representative of an SNP in the inputted range. The further up an SNP is plotted on the graph, the greater that SNP association is with type 1 diabetes. A dashed horizontal line indicating the genome-wide significance threshold (p-value= 5 x $10^{-8}$) is plotted for easy analysis. Manhattan plots were chosen to display the statistical significance associated with the SNPs as many SNPs can be displayed in one figure and different genome regions can be compared to one another for example, one genome region, in particular, may contain lots of SNPs with high association with type 1 diabetes compared to another region. The entire code for the manhattan plot was directly implemented into the Flask framework.

## Function

When a chromosome region is entered by the user, The SQL database table ‘GWAS’ was queried for rs ID, p-value, and chromosomal location. A dictionary was created with the data retrieved before using pandas to turn it into a data frame. A pandas data frame was used as the SQL data had to be reformatted to generate the Manhattan plot. Furthermore, pandas allows for data handling that SQL does not such as mixed-type columns and hierarchical metadata.

The gwas data sourced from EBI was not formatted in a way suitable for generating a Manhattan plot. Firstly, the p-values were a string, written as 8 x$10^{-8}$ as an example. As this is a string, numeric operations to produce data like the -$\log_{10}$ of the p-value could not be conducted. Every p-value was rewritten into scientific notation by replacing ‘x10’ with ‘e’. The EBI GWAS opted to combine chromosome number and base pair location as one value under the ‘location’ column, for example, 6:410417. For ease of generating the plot, it is better to have these values as two separate columns, ‘chromosome’ for chromosome number and ‘location’ for the base pair location. Code utilising numpy was used to achieve this. Lastly, necessary columns were changed from objects to floats/integers for calculation and plotting purposes.

Matplotlib and seaborn were used for the generation of the plot. Seaborn was chosen for its relatively simple syntax as well as by default it prevents overlapping plots. The chromosomes are colour coded in a way that the colour of the SNPs located on one chromosome is different from that of the neighbouring chromosomes; clearly defining the start/end of each chromosome and aids in the legibility of the plot. Variables can be adjusted to match the table names of the data frame if they differ from what was used in the web application script. Aesthetic changes can be made such as adjusting the palette, i.e, ‘palette=bright’ to suit the individual. X-tick labels were set to the chromosome number.

As the database for the web application only contained SNPs located on chromosome 6, the generated plot will only show chromosome 6 SNPs. However, the Manhattan plot is dynamic therefore, if a database containing multiple chromosomes is used, the Manhattan plot will adjust to show these SNPs, with the Y-axis and X-axis automatically adjusting.

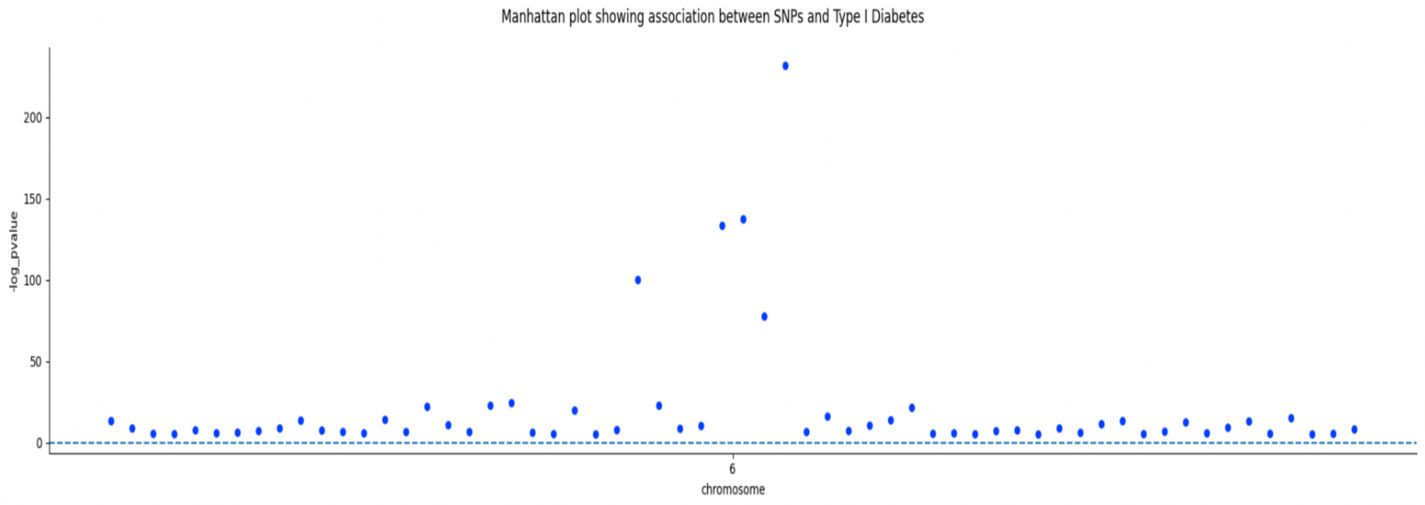


**Figure 10:** Manhattan Plot for all SNPs located within an inputted region on chromosome 6. Each plot represents an individual SNP. -Log$_{10}$ p-value along the y-axis and chromosome location along the x-axis. The horizontal dashed line is the GWAS significance threshold ($p = 5 \times 10^{-8}$). The axis adjusts depending on the SNP called.

If an SNP or region is searched that is not within the database, an error is returned: “No information available for %s." % snp_name”.

To connect to Flask, The plot is saved as a bytes object using the BytesIO() library and saved as a variable ‘buf ’in a ‘png’ format. Base64 library encodes the plot whilst utf-8 decodes it when returned on the HTML. The plot can be saved as a ‘png’ image when you right-click on it and select ‘save image as…’.

Future improvements include colouring certain SNPs by different criteria such as SNPs along the GWAS significance threshold or SNPs with the greatest -log$_{10}$ p-value. At a glance, these implications improve the readability and overall visuals of the plot. Annotating the greatest/top ten -log10 p-value makes the graph more informative for the user. Currently, the plot produced by the web application is a static visualisation. The next step would be to create a dynamic plot by using the mpld3 library, where the user can zoom in/out as well as crop to zoom on a particular section when the dynamic model is enabled. The fig_to_html function provided in the Mpld3 library takes a matplotlib figure and returns it in an HTML template.