1. **About SNPEuro**

SNPEuro has been created as a web application (webapp) designed to give users information regarding population genomics. This field, which is the application of genomic technologies to understanding populations of organisms and how genes contribute to health, well-being, and conservation. With recent advances in sequencing technologies, the need for niche platforms into their investigation is underscored.

SNPEuro has been created as a simple user-friendly platform, to allow users to retrieve single nucleotide polymorphism (SNP) information for a genomic region of interest from a single chromosome from all available European population data. Search features of SNPEuro are linked to a curated database with information on the SNPs (add here). The user can then see genotype and allele frequencies relative to selected SNPs. They are also able to visualise calculated statistics relative to their chosen SNP, which include a measure of genetic/nucleotide diversity, haplotype diversity and a measure of neutrality (see statistical tests section). When multiple populations are selected, population genetic variation for each pair value is reported. The user is also able to visualise analysis of the data with the graphical outputs, which can be downloaded for future use/ interpretation.

This webapp was developed as part of the MSc Bioinformatics programme at Queen Mary University of London (QMUL), under the supervision of Professor Conrad Bessant and Dr Matteo Fumagalli. Developers of this webapp were V. Bui, U. Gill, L. Johnson, and N. Kharnair. The source code is available on GitHub (<https://github.com/MSc-Bioinformatics-Group-Project/SNPEuro>)

1. **Design philosophy**

SNPEuro Portal was designed to aid biologists with a simple tool of accessing data on population genomics. Inferences from population genomic data are widely used in a number of disciplines, including conservation genetics, evolutionary biology, and precision medicine amongst others. Next-generation sequencing (NGS) technologies have revolutionised the genomics field, providing biologists and researchers alike to attain large amounts of genomic data for many samples from different populations. SNP genotyping is the measurement of genetic variations of SNPs between numbers of a species and the most common type of genetic variation. The analysis of SNPs is important as it is able to link sequence variations to phenotypic changes. Researchers, for example, have found that SNPs may help predict individuals’ responses to drugs, susceptibility to environmental factors and the risk of developing certain diseases (ref). Information from SNPs can link sequence variations to phenotypic changes, which can advance and elucidate the molecular basis of disease.

This webapp is a small application to help biologists with limited data retrieval and information regarding population genomics, specifically SNPs, relative to European populations. Of key consideration is that this webapp does not rival or replace larger more robust webapps such as Ensemble and UniProt. This webapp is however useful for aid future similar web-based developments in population genomics. We utilised GitHub throughout development to aid version control. (specificities)

1. **Software details**
   1. ***architecture explanation/structure***

The SNPEuro portal was designed using the Flask framework. The data used in the webapp was generated using local raw data from the 1000 Genome Project, which are stored as VCF files as a database (see database generation section). Statistical tests were undertaken using RStudio and then linked to Flask via rpy2. The main application was run via Flask. Webapp users can access this data via the browser (HTML) via webpages, displaying data as txt/csv files or opt for different statistical analysis relevant to the European populations in which they are interested. (This needs additions/modifications) – need to update below, once all is finished.

Diagram

Description automatically generated

*Figure 1: Schematic of software architecture.*

The manner in which the software operates, where the user enters an input on the webapp, via the browser, this operates to the back-end and feeds the relevant URL. At the back-end this will ‘access’ the curated database, which has been generated. The data will then be manipulated/accumulated relevant to the input from the user. The required information, as per request will then be delivered as an output to the user, via the browser and user interface. (At the back-end, we also have access to specific hyperlinks to other websites, germane to the information the user has retrieved). The user is able to access the webapp, with the specific features created for the front-end, with the landing page visualisations and generation of results.

* 1. ***Software & website layout***

(If able to successfully deploy, then can provide a link here/via Github with how to run this)

Need to update website architecture

Diagram

Description automatically generated with medium confidence

*Figure 2: Website functionality schema*

*Can also provide a table for above e.g About\_html – provides information about the webapp*

Diagram for website architecture – to add here (need to include the download button option here).

The schematic of the website functionality/process is shown above in Figure 2. We included a home/landing page which provides generic details of the webapp with graphics (need to confirm this). Users are then able to access an ‘About’ page detailing the utility of SNPEuro. Further single tabs including a ‘Help’ and ‘Documentation’ page with an external link to GitHub are also included. A ‘Resources’ tab is offered which links to the specifics of SNPEuro. This includes search windows for SNP, gene search and genomic co-ordinates (positions). The gene search tab can take an input as gene ID and also any gene aliases which may be relevant. Genomic co-ordinates can also be supplied with a start and end coordinate (capped at a limit of X, also see limitations section), which then also deliver the results. The webapp returns, for each SNP, genomic position, genotype frequencies and allele frequency, where frequencies are provided for each population separately. When multiple SNPs are returned the user is able to select the European populations of choice, which are British, CEPH (Centre d´Etude du Polymorphisme Humain), Finnish, Iberian and Toscani. Statistical analyses are provided if multiple SNPs are shown. These can be shown dependent on the different European populations selected and the desired statistical tests from the user (see Statistical tests section). The selection of both functions for populations and statistical tests asks the user to specify by typing, their desired selection. We opted for this over a drop-down menu for the ease of selecting multiple populations and/or statistical tests simultaneously. Graphical representations are presented in accordance with the chosen statistical test by the user, either as separate graphs or sliding windows according to the range selected by the user. Summary statistics can also be downloaded as a csv/txt file. (modifying)

(Need to explain the above branches a bit more and to justify why we have done as above)

1. **Database details**
   1. ***Data collection & curation***

Primary data collection for the webapp was obtained via use of the 1000 Genome Project website/tool (<https://www.internationalgenome.org>). Data pertaining to the full chromosome 21 was undertaken with all data from the 30X study, as VCF genotype/haplotype information for each sequenced sample at each SNP from all populations. Full chromosome data, GTFV file (rsID numbers, genes, locations) were extracted from the NCBIwebsite with all permutations. Via bash command, BCFtools were used to filter the VCF for the required populations relevant to SNPEuro. This was annotated with RSIDs with BCFtools using complete DBSNP VCF provided by NCBI (00-All.vcf.gz on this link  <https://ftp.ncbi.nih.gov/snp/organisms/human_9606_b151_GRCh38p7/VCF/>) Extraction of RSID positions for the reference allele and alternative allele from the now annotated 1000 Genome Project and then this was stored as a .txt file. Plink2 was the used on this annotated VCF to calculate allele frequencies and genotype counts for each position in the 1000 Genome Project VCF. Returned results show the position and RSID (if available), which are stored in a .txt file. Using the RSID, positions, reference, and alternative allele txt file from above, a dictionary was created in Python with RSID as the key and the rest as the values. Respective genotype and allele frequencies were then added to this dictionary (NB those without RSIDs are not stored). The output here is a .JSON file for storage, which is read each time as required. For example, if the RSID was 123, then the following:

rs123":[100,'a','g',0.00001,'0/0=3,0/1=4,1/1=100']

Position, Reference, Alternative, Allele Frequencies, Genotype counts.

For gene name, this is used as the key, with the positions of the RSIDs associated as the values, then taking the maximum and minimum for those position for the requested input for reading the readVCF function in R:

i.e "Gene1":[1,3,5,17,66,153] -> readVCF(1,153)

The data obtained from the VCF files was sorted to obtain the relevant information, which included the following: (put below into table format).

- GT; Genotype

- AB; Allelic balance for each heterozygote genotype

- AD; Allelic depths for the reference and alternative alleles (in the order listed)

- DP; Approximate read depths (reads with MQ=225 or with bad mates are filtered)

- GQ; Genotype Quality

- PL; Normalised Phred-scale likelihoods for genotypes as defined in SCF specifications.

Once the data was collected into the VCF file format, downstream reading was undertaken for statistical analysis using an R package; PopGenome (version X). The VCF file was read using the readVCF() function, which requires a vcf.gz, compressed file, and its indexed file in the vcf.gz.tbi format, with the get a “GENOME.class” object. Subsequent data curation for statistical analysis was undertaken in RStudio using the PopGenome package (see statistical analysis section). Consequently, to integrate the R-based scripts for statistical analysis into Flask, using Python, we used the rpy-2 package, developed specifically as a Python-R bridge. This therefore allows the user inout to change relevant request information within the Flask framework. For the purposes of graphical outputs being merged onto the Flask framework, we saved these outputs to the hard drive as a .jpg file and then used that .jpg file to load into Python.

(Need to get this checked by Liam and add/modify here)

* 1. ***Database manipulation & schema***

For database manipulation and generation we used pandas and easy.entrez and json for storage of data files within the database, which was a simplified method of handling the database. Add in Gene names JSON

Although there are a number of other platforms which could have been used for database handling and manipulation, such as Structured Query Language (SQL), we decided that these were not essential for this webapp. We required access, manipulation and communication with the database, which we were able to undertake this utilising the pandas package with Python and the Flask framework. Whilst we could have undertaken this using SQL which has the advantages of faster query processing and the limited need of coding skills with multiple data views it still has a complex interface and for the purposes of our simple webapp we felt its use was not required.

(Need to get this checked by Liam and add/modify here)

(Need to speak to Liam – get verbal summary of data assembly.ipynb doc)

1. **Statistical Analysis**

***5.1 Statistical Test development (add sub-headings)***

Within the SNPEuro Portal webapp, following retrieval of information on the relevant SNPs, when multiple hits are returned, specific to populations selected summary statistics can be calculated. Statistical analysis which is delivered by SNPEuro includes nucleotide diversity as a measure of genetic diversity (need to mention pi), haplotype diversity and Tajima’s D as a measure of neutrality. In addition, when multiple populations are selected then population genetic variation (FST value) for each pair of populations is reported. Using the PopGenome package in RStudio all statistical analyses could be undertaken within a single platform.

Here we describe the selection of statistical tests that are utilised within the webapp and then the relevant functions from the PopGenome package which were used to implement and employ these for data generation

For statistical tests we selected diversity statistics measurements which included nucleotide diversity and haplotype diversity. Nucleotide diversity is a concept in molecular genetics measuring the degree of polymorphism within a population. Nei and Li in 1979 first introduced the measure of nucleotide diversity, which is simply defined as the average number of nucleotide differences per site between two DNA sequences in all possible pairs within the same population, (this is denoted as Pi). Nucleotide diversity if a measure of genetic diversity and commonly used in combination with other statistical analyses of population genomics and diversity and is similar to expected heterozygosity, a measure of haplotype diversity. Within populations highly diverse or well-balanced libraries have approximately equal populations of all four nucleotides in each cycle throughout the sequencing run. Conversely, low diversity libraries have a high proportion of certain nucleotides in a cycle. Within our SNPEuro portal we are able to provide nucleotide diversity between SNPs comparing different populations (need refs)

Also, a measure a genetic diversity, our webapp also provides a calculation for haplotype diversity. A haplotype is a set of DNA polymorphisms that tend to be inherited together and can be a combination of alleles or set of SNPs found on the same chromosome. Haplotype diversity represents the probability that two randomly sampled alleles are different and can be implicated by a number of different processes such as mutation, recombination, marker ascertainment and demography. It is thus the measure of the uniqueness of a particular haplotype in a given population.

In line with genetic diversity, key in population genomics is also a measure of neutrality. Neutrality tests compare two estimators of the population mutation parameter that characterises the mutation-drift equilibrium. The Tajima’s D test is one measure of neutrality and is computed as the difference between two measures of genetic diversity. It is the mean number of pairwise differences and the number of segregating sites, each scaled such that they are expected to be the same in neutrally evolving population of constant size. Tajima’s D thus distinguishes between a DNA sequence evolving randomly (or neutrally) and one evolving under a non-random process. A negative result signifies an excess of low frequency polymorphisms relative to expectation (population size expansion), whereas a positive result signifies low levels of both low and high frequency polymorphisms, indicating a contraction in population size and/or balancing selection (refs).

Within the webapp is the user can select different European populations as noted above and then population genetic variation is delivered. This measure, also called the fixation index (FST). FST is denoted as the proportion of the total genetic variance contained in a subpopulation relative to the total genetic variance and thus is a measure of population differentiation due to genetic structure, thus a measure of genetic differentiation and when species diverge in the presence of gene flow. It is commonly estimated from genetic polymorphism data (SNPs) as it is within this webapp. Population selection strongly influences FST values, such that closely related groups are indistinguishable, displaying panmixis, where 2 populations are interbreeding freely. Thus, within this SNPEuro webapp, expectedly there may be low FST values between the selected populations. (refs).

(? Need to show the formulas)

The SNPEuro app delivers the statistical outcomes, relative to the tests indicated above along with graphical visualisations. The readouts utilised here include a sliding window and plot for Tajima’s D. The sliding window analysis is a method studying the properties of molecular sequences, where data are plotted as moving averages of the criterion in question (e.g., nucleotide diversity), for a window of a certain length slid along a sequence or sequence alignment. A plot for Tajima’s D is also included.

***5.2 PopGenome package***

To undertake the statistical analysis within the SNPEuro app the PopGenome package within RStudio was utilised. This provides sufficient tools for the appropriate tools for population genomics data analysis. We noted this package included a wide range of polymorphism tests, neutrality tests and FST estimates with suitable execution from sequence data from the 1000 Genome project similar to that for SNPEuro. The package allows for the full range of methods to be applied to whole alignments, sub-sets of sequences and sliding windows as we have utilised, based on nucleotide positions or on SNP counts.

***5.3 Statistical analysis code***

Following installation of the PopGenome package, statistical analysis was carried subsequent to loading the relevant libraries.

Library (tidyverse)

Library (PopGenome)

Library (ggplot2)

Population data was set and then position data was obtained, with the GENOME.class object, which provides some information about the main methods provided by PopGenome. This object/function is also used to read the VCF file with the information from the selected populations for SNPEuro. Initially SNP counts are obtained. The sliding window is then generated, which is a function contained in PopGenome. The function sliding.window.transform() transforms an object of class GENOME into another object of class GENOME, where new regions correspond to individual windows. This mechanism enables the user to apply all methods that exist in the PopGenome framework. PopGenome concatenates the data if the parameter whole.data=TRUE. This mechanism enables the user to work with very large datasets, which can be split into smaller chunks that are stored in the input folder. PopGenome is able to concatenate these chunks for analysis. The functions used; readVCF and readSNP undertake this automatically. If whole.data=FALSE, the regions are scanned separately, which can be done as type=1: Define windows based on SNP counts or type=2: Define windows based on nucleotide counts.

The diversity.stats-methods were used as a generic function to calculate nucleotide and haplotype diversities, which require division by the slot GENOME.class @ n.sites to obtain diversity per site. Neutrality (Tajima’s D) is then calculated with the GENOME.class function. Subsequently FST statistics and dxy (diversity stats) can be obtained, where the latter rely on the former, in relation to the populations. Number of synonymous and non-synonymous changes were also calculated using the GENOME.class @ () object, these outputs were then used for measuring neutrality and Tajima’s D.

(Need to check code fully with Liam and check nothing missing)

***5.4 Pros and cons of PopGenome***

We decided to use the PopGenome package, which can efficiently process genome-scale data as well as large datasets via the R platform. A number of computer programs exist for performing population genetics calculations, they can be limited in the analyses they provide within a single platform, which was an advantage of PopGenome. The package can read associated annotation files in GFF format, which enables the easy definition and classification of SNPs based on their annotation, with analyses applied to sliding windows as depicted in SNPEuro. The diverse statistical analysis offered by this package was an attractive attribute for its use. The integration using R facilities with downstream analysis allowed the production of high-quality graphics, which were limited with other packages that we had explored. The limitation we noted with using PopGenome and thus RStudio was the additional step of integrating these analyses from R to Python for the Flask framework. We utilised rpy2 for this purpose, but this was an additional step, which would not have been required had a Python related package been used, however, we felt the graphical readouts using PopGenome in RStudio were superior to those in other packages explored. In RStudio, we did also note a technical issue where we were not able to select a genomic coordinate with a range of >140000 and thus have included this as a stipulation in our webapp. This, of course, is not ideal and a limitation of using RStudio here as any value with a range larger than this led to abortion of the package and thus unable to execute the code any further. We experimented by changing from linux to Mac OS operating machines but were unable to solve this issue. Currently we have stipulated this as the maximum range for genomic coordinate insertion and would improve this feature should we have had more time and in future development.

1. **Technologies**
   1. ***Flask architecture***

The Flask framework is based on templates related Werkzeug and Jinja 2 templates and has an easy link integrated with Python. Flask runs with x version of Python and is based on Jinja 2 sharing a similar language to Python, with easy integration to run code on the webapp as well as HTML. Within the framework we have a main homepage, which includes…. about, along with other tabs, which the user can click on. A documentation tab also links to the GitHub repository. The homepage allows the user to enter different SNP detailed information – allocated to different tabs (e.g., genomic co-ordinates etc). And updates following the structure of the flask architecture. Need to also add the version within software schematics. More needed here and need to add the different packages we have used with their respective position in the software architecture.

Add why Flask was used

* 1. ***Flask Framework (how Flask was used)***

**(includes integration and table for site layout)**

The information provided here is relevant to the developer of the webapp. We utilised the Flask framework for creation of the webapp. Additional technologies that were required for integration with Flask included our curated database with Pandas along with RStudio and rpy2.

For the installation of Flask a virtual environment is created within a folder (using a python writer, we opted for Visual Studio Code) from where the developer wishes to run their framework. The virtual environment is then activated prior to initiating the framework, with the following code:

pip install virtualenv

Virtualenv env

Source venv/bin/activate

The virtual environments allow the developers to manage the dependencies for the project, both in development and production. These virtual environments are independent groups of Python libraries and thus one project does not affect other projects or operating system packages. We used documented instructions when creating virtual environments in visual studio code (? requirements.txt).

Within the activated environment, Flask is installed via the command line:

pip3 install flask

An app.py folder is then created, which is the main constituent containing the Flask application. The relevant code for this set-up is:

- Import Flask: (from flask import Flask)

- Set-up application: (app=Flask(\_name\_)

- Set-up route of homepage: @app.route(‘/”) – [within the brackets included is the URL setting].

- Define index for the route: - Def index():

- Set-up the route for the ‘About’, ‘Help’, ‘gene search’, ‘SNP search’, and ‘Genomic co-ordinate’.

Additional imports required for the inclusion of statistical analysis for our webapp, within Flask framework included ‘flash’, ‘statistics’, ‘E’, ‘name’, ‘pandas’.

Templates are then generated within the Flask framework with the utilisation of HTML and bootstrap (Cascading Style Sheets; CSS) HTML is thus coupled to the CSS languages for website delivery, which included the navigation bar with drop down options for the resource’s tabs including gene ID, coordinates, and SNP searches (achieved through bootstrap). A footer was also created which also included the links on the navigation bar.

The Flask web forms were created with various libraries. Flask-WTF was installed for working with forms within the framework:

pip install Flask-WTF

and the following code used for library imports:

from flask\_wtf import FlaskForm

from flask\_wtf import Form

from wtforms import StringField, SubmitField

from wtforms.validators import Required

The created webforms, noted above, were then linked for the statistical analysis. R packages were imported, with rpy2.robjecrs as robjects. A class was created to define a form for SNP (SNPForm) along with a search for the SNP. SNP data was then loaded from a TSV file into pandas dataframe with SNP name as index. Similarly, forms were created for gene name/ID and genomic coordinates. Here we included the relevant return information should specific SNPs, genes, co-ordinates not be available, with the generation of code using exceptions.

To test that the variables entered in each form were functional, we used truncated data (VCF files) to determine if the correct outputs were delivered and adjusted the code accordingly in the event of errors arising. Having imported the rpy2 package as the bridge between Python-R, we were able to integrate the code from RStudio for the statistical analysis to Flask. Graphical outputs of the statistical analyses, which included sliding windows and Tajima’s D plot, were also integrated within the Flask framework.

(Need to check with Vi and Liam – need to modify here)

***6.3 Front-end framework***

Development of the front-end of the SNPEuro webapp was largely performed using HTML and CSS – with bootstrap which allowed for the improvement in functionality of the webapp. This included the creation of various tabs, a footer page and the inclusion of data entry points for the user. The different views from tab or selection otherwise are then returned to the user as a HTML template, which contains the information regarding the manner in which the information is to be displayed to the user via the browser. We note that in hindsight and with additional time we would also provide additional focus tom the front-end with further bootstrap use to improve the SNPEuro webapp in terms of aesthetics, but the priority here was to complete a functioning prototype with the software requirements as requested.

The table below, in conjunction with Figure 2, includes the relevant tabs that are included with the webapp with a brief description of their function.

Table of the tabs (html)

1. **Flask Pros and Cons**

There are number of technology platforms that can be used for webapp software development ­such as Flask, Django and FastAPI, which are operated with Python. We opted to use the Flask framework, with it being flexible, as most parts of Flask have the possibility of changing, which is not necessarily the case for other platforms. It has compatibility with the latest technologies and was simple enough to integrate with R via the rpy2 package. Flask is also beginner friendly due to its overall simplicity and enables app development with ease. In addition, it is easy to experiment with Flask with the overall architecture and libraries, especially as we were not initially certain which libraries or frameworks would work best for our webapp. The SNPEuro webapp is a small platform, and we noted that when developing the app further Flask may not be the best platform, though for prototype purposes it seemed ideal as it is known to be well utilised for simple cases and web applications. Flask also is easy for documentation purposes, with tips arranged in a structured manner.

For the purposes of this small webapp, the disadvantages of Flask were limited, but potentially include risks for security, with slower development and complicated maintenance for larger implications, factors which would need to be considered. Furthermore, for future use, Flask lacks a large toolbox, thus developers need to manually add extensions such as libraries. The addition of a number of extensions may lead to the app becoming slow. The modular nature of the Flask framework means that many developers working on software together need to familiarise themselves with each constituent part of the framework

(Need to add a bit here)

1. **Further development of webapp**

SNPEuro can be further developed….

1. **Deployment**

For deployment options….

1. **Limitations and areas for further improvement**

Issues with RStudio and the range of genomic coordinates….

1. **References**

<https://stackoverflow.com/questions/43309343/working-with-user-uploaded-image-in-flask>

<https://www-users.york.ac.uk/~dj757/popgenomics/workshop5.html>

<https://evolutionarygenetics.github.io/Chapter8.html>