# Genomic Data Processing and Analysis Solution Scientific Report

Elias Crum

The solution starts with the preprocessing function with a Sample ID as an input. The `preprocess_vcf()` method uses the Linux `wget()` and `gunzip()` methods to download and decompress the specified .vcf file. To avoid this step, the user can alternatively download and/or decompress the .vcf file manually. Additionally, the `fix_ontology()` method alters an included vcf_ontology.ttl file to the local machine's file path structures before parsing and conversion begins.

The .vcf file now enters the `parse_to_rdf()` function. The only parameter for this function is the name of the .vcf sample file. This method starts by iterating over the '##ParameterInfo' lines of the .vcf input file. Once the single nucleotide polymorphism (SNP) lines are reached, the columns with data stored in RDF format are columns 0 (chromosome location), 1 (nucleotide location), 2 (Unique SNP ID), 3 (Reference Allele), and 4 (Variant Allele) using the `Graph.add()` method within the RDFLib package. Importantly, the RDF Subjects are the SNP IDs because these IDs are unique to each SNP and therefore there will be no redundancy. Additionally, each SNP can be represented as a node with attributes Location, Variant, and Reference. Within this method, RDF stores are split into separate data sources (.ttl files) depending on the chromosome on which the SNP is located, essentially partitioning them into separate knowledge graphs for delocalization and additional semantic metadata. The choice for this data store separation strategy is arbitrary but also provides non-redundant SNP relationship information. Within this method, VCF data lines that did not have a documented SNP ID (represented by a "." in the .vcf file) were filtered out because this challenge revolves around the searching for variant alleles with a known SNP ID. The resulting 23 .ttl RDF data store files are written to a directory named "RDF_Data," and are then used by the following method to retrieve a specified SNP ID variant allele.

The `snp_query()` method takes a SNP ID input and searches the 23 .ttl RDF files for the SNP's variant nucleotide allele. The method iterates over the .ttl files within the "RDF_Data" directory while using them as local SPARQL endpoints to query for the RDF subject that matches the input SNP ID. When the SNP ID is found, the predicate (prefix: Variant) and object (literal: allele) are written to the output file "SNP_Information.txt" and the SPARQL query is terminated.

The local Python implementation allows for command line functionality and its only dependency is Python3 and the RDFLib Python package. To ensure all dependencies are installed properly the script can be run with the -t flag which uses a small test.vcf file that runs nearly instantaneously.

With the help of the above-described solution, it is revealed that the participant huF7A4DE exhibits an "A" allele at SNP rs762551, thus, since they do not carry the "C" allele, they are not a 'slow' metabolizer of caffeine based on their genetic data.

One drawback to this approach is that when running on a 4-core laptop, this script took ~33 minutes with 96% cpu load. Both the conversion process and SPARQL querying process took a significant amount of time, memory, and computational load. With more time, increased efficiency could be achieved through more deliberate file partitioning and the integration of a caching mechanism to make loading knowledge graphs from .ttl files quicker. Another reason for the long runtime is that this solution utilizes Python as the scripting language, which is notorious for long runtimes due to underlying language architecture quirks. Notably, numerous issues arose when running this script on a Windows OS due to RDFLib's inability to produce URI addresses for conversion to RDF with Windows file path structure. With more time I could have potentially found a solution, but currently, this script is only operable on Linux OS machines (or Windows machines with Linux subsystems).