

Bioinformatics & Computational Genomics of SARS-CoV-2 and other  
Coronaviridae: Nucleotide Composition, Metagenomics Cluster Analysis, and  
Mathematical Optimisation of Pairwise Sequence Alignment Algorithms



A dissertation submitted in partial fulfilment

of the requirements for the degree of

B.Sc. (Hons) in Computer Science

at

The Queen's University of Belfast

by

Daniel Bell

23/04/2021

**SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER  
SCIENCE**

**CSC3002 – COMPUTER SCIENCE PROJECT**

**Dissertation Cover Sheet**

A signed and completed cover sheet must accompany the submission of the Computer Science dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

Student Name: **Daniel Bell**

Student Number: **40198194**

Project Title: **Bioinformatics & Computational Genomics of SARS-CoV-2**

Supervisor: **Dr. Anna Jurek-Loughrey**

**Declaration of Academic Integrity**

Before submitting your dissertation please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.

**I declare that I have completed the tutorial on plagiarism at**

**<http://www.qub.ac.uk/cite2write/introduction5.html> and are aware that it is an**

**academic offence to plagiarise. You declare that the submission is my own original work. No part of it has been submitted for any other assignment and I have acknowledged all written and electronic sources used.**

*Student's signature*

*Daniel A. Bell*

*Date of submission*

*23/04/2021*

## Acknowledgements

I thank Dr. Anna Jurek-Loughrey from Queen's University Belfast for her willingness to take on the role as project supervisor and displaying patience throughout towards my learning of both biology and industry standards as prerequisites.

## Abstract

This dissertation details the prototype development process undertaken; design, implementation, and testing, as well as comprehension of applied algorithmic theory and analyses of results from trialling out the solution as the intended user, a Bioinformatician, accompanied with validation.

The project aim was to propose a one-stop software system of well-rounded features to assist Bioinformaticians in the ongoing research into SARS-CoV-2; later committing to the mission of being extensible for emerging variants that have appeared during this time for preliminary research, i.e. practical for immediate data analysis of novel virus samples.

The solution meets all project requirements promised and to effect.

## Contents

Acknowledgements.....	iii
Abstract.....	iii
Contents.....	iii
1 Introduction and Problem Area.....	1
1.1 Problem .....	1
1.1.1 Bioinformaticians .....	1
1.1.2 Industry.....	1
1.2 Solution .....	2
1.3 Benefits.....	2
1.3.1 Bioinformaticians .....	2
1.3.2 Preclinical Research.....	2
1.4 Goals .....	3
2 System Requirements and Specification.....	3

2.1	User Interactions .....	3
2.2	Assumptions .....	3
2.3	Constraints.....	3
2.4	Requirements.....	3
2.4.1	System Characteristics .....	4
2.4.2	Features.....	4
2.5	Requirement Amendments.....	5
2.6	Development Technologies.....	5
3	Design .....	6
3.1	Application Overview .....	6
3.2	Protein Synthesis.....	7
3.3	Normalised Frequencies .....	7
3.4	Bio Type.....	8
3.5	Multiple Sequence Alignment.....	8
4	Implementation .....	9
4.1	Development Tools .....	9
4.1.1	Python.....	9
4.1.2	Integrated Development Environments .....	10
4.1.4	Source Control.....	10
4.1.3	Docker Server.....	10
4.1.4	SimLoRD.....	10
4.2	Code Approach .....	11
4.2.1	High Performance.....	11
4.2.2	Procedural Programming.....	12
4.2.3	Exception Handling .....	12
4.4.3.1	Trinucleotide Protein Synthesis .....	12
4.2.4	Storing Outputs.....	12
4.3	Essential Features & Algorithms.....	12
4.3.1	Normalised Frequencies.....	13
4.3.2	Cluster Analyses.....	13
4.3.3	Protein Synthesis .....	14
4.3.4	Pairwise Sequence Alignment .....	18
5	Testing.....	21
5.1	Unit Testing .....	21
5.2	Usability Testing.....	21

5.3	Integration Testing .....	21
5.4	Performance Testing.....	22
6	System Evaluation and Experimental Results .....	22
6.1	Success Summary .....	22
6.2	Nucleotide Composition.....	23
6.2.1	Nitrogenous Base Pairs.....	23
6.2.2	Normalised Frequencies.....	23
6.3	Metagenomic Cluster Analyses.....	24
6.3.1	Principal Component Analysis Results .....	25
6.3.2	t-Stochastic Neighbour Embedding Results.....	25
6.3.3	Interpretations.....	26
6.3.4	k-Means .....	27
6.4	Protein Synthesis.....	30
6.5	Pairwise Sequence Alignment.....	31
6.5.1	Algorithmic Optimisation.....	31
6.5.2	Genomic Relationships .....	32
6.5.3	Comparison with EMBL-EBI.....	34
6.6	Multiple Sequence Alignment.....	34
6.6.1	Alignment Files .....	34
6.6.2	Visualisation Tool.....	34
6.7	Macromolecular Docking.....	35
6.8	Significance of Results .....	36
6.9	Developed Improvements.....	36
6.10	User Feedback.....	36
6.11	Implications .....	37
6.11.1	Societal .....	37
6.11.2	Commercial.....	37
6.11.3	Economic .....	38
7	Conclusion.....	39
7.1	Project Management Evaluation.....	39
7.1.1	Benefits.....	39
7.1.2	Drawbacks.....	39
7.2	Solution Evaluation .....	39
7.2.1	Strengths .....	39
7.2.2	Weaknesses.....	40

7.2.3 Learning by Failure.....	40
7.2.4 Implementation.....	40
7.2.5 Development Technologies .....	40
7.3 Future .....	41
7.3.1 Possible Features .....	41
7.3.2 Prospects .....	42
7.4 Summary .....	42
8 Appendices .....	42
A – Functions .....	42
B – Genome Datasets .....	52
C – Data Model.....	52
D – Code .....	53
D.1 – Code Style .....	53
D.2 – List Comprehensions.....	53
E – Experimental Findings .....	54
E.1 – Normalised Frequencies .....	54
E.2 – Cluster Analysis.....	56
E.5 – Macromolecular Docking .....	58
F – Conclusion.....	59
F.1 – Gantt Chart.....	59
F.2 – Learning by Failure .....	60
G – Virology.....	61
G.1 – Coronavirus Taxonomy.....	61
G.2 – Epidemiology .....	62
G.3 – Biochemistry .....	63
H – Other Terminology .....	67
H.1 – Sequence Analysis.....	67
H.2 – Metagenomics.....	67
9 References .....	67

# 1 Introduction and Problem Area

## 1.1 Problem

### 1.1.1 Bioinformaticians

Biologists with expertise in computer science have increasingly been demanded in academia and industry. Despite employment demands, there's a large skill gap. Many Biologists have no knowledge of Data Science; that is software engineering, data analytics, and mathematics.

Authors John C. Wooley and Herbert S. Lin in their book “Catalyzing Inquiry at the Interface of Computing and Biology” list an assortment of challenges that Bioinformaticians face.

Many of which remain to be completely resolved.

A few that stand out that this solution aims to address:

- “Bridge the gap between computationally feasible and functionally relevant time scales” (p. 429) [18],
- “Bring experimental and computational groups in molecular biomedicine closer together” (p. 429) [18].

### 1.1.2 Industry

Pharmaceutical companies have the task of obtaining a clear understanding of novel viruses and developing its vaccine promptly. The sooner targets are met, the less likely a widespread pandemic occurs and further strains evolve.

This has been the case over the past year, with three major strains having emerged since the initial outbreak of a novel coronavirus from Wuhan, China. Big Pharma companies race to develop vaccines of higher efficacy for SARS-CoV-2 and its variants.

Referring back to the aforementioned publication, there are a few points raised that impede Bioinformaticians' comprehension of a virus that still need improving upon:

- “Study protein-protein and protein-nucleic acid recognition” (p. 429) [18],
- “Visualization interfaces (for information visualization and scientific visualization)” (p. 434) [18].

## **1.2 Solution**

To develop a faultless application, automating fundamental computational biology tasks; working with DNA, protein and reads datasets. Utilised by biology purists with no experience in data science, this application should handle all data cleansing and exception handling. All while accurately representing its scientific concepts, integrating efficiently with industry standards and reputable sources of data.

## **1.3 Benefits**

### **1.3.1 Bioinformaticians**

This solution would provide the following benefits to the job role:

- Performing invaluable analyses on novel viruses, within a moment.
- Reducing the time it takes to perform high-level analysis, freeing up time for more rigorous specialised work.
- Continuity of analyses, making it easier to compare and contrast research findings amongst different genomes, i.e. DNA datasets, and their proteins.
- Usability, actively be easier and friendlier to deploy over using manual toolkits and cheaper than what the competition offers.
- Accessibility for Biologists to enter the job space that have no experience with Computer Science, thus meeting employment demands.

### **1.3.2 Preclinical Research**

Obtaining an understanding of a new outbreak at the earliest possibility is crucial for not only disaster planning a population but in determining the direction of drug trial development for a vaccine, before the virus spreads, turning into an epidemic or pandemic.

Based on insights gathered, this would significantly narrow down possible avenues for preclinical research, preparatory to drug trials, delivering a vaccine sooner. Standardising routine essential tasks of a Bioinformatician also means results in the scientific community can be compared to more efficiently.



## **1.4 Goals**

To have a working demo by 11<sup>th</sup> December, 2020 showcasing the majority of requirements mostly working. Ultimately, all requirements implemented and robust by 23<sup>rd</sup> April, 2021.

# **2 System Requirements and Specification**

## **2.1 User Interactions**

The user base will navigate the application through ‘main.py’. All commands are commented. User inputs required include:

- Genome datasets imported to ‘src/’; from API or manually,
- Passing file names, with or without directory or extension; template code provided,
- ‘bio\_type’; either ‘Protein’ or ‘DNA’.

## **2.2 Assumptions**

- A distribution of Python v3.0+ and a local IDE must be installed onto the user’s machine; any OS with ideally 8GB RAM,
- Only DNA sequences are accepted as input data (.fasta/ .fna),
- Execute commands: “cd environment”, “pip install –r requirements.txt” in terminal.

## **2.3 Constraints**

There is no GUI, hence User Interactions. The context of the system is Virology, e.g. certain graphs’ titles display, yet the system works with any and all DNA or protein sequences.

## **2.4 Requirements**

Set out to meaningfully solve the problem area, chosen implementations complement each other and are actively practised by Bioinformaticians. These aim to provide a standardised workflow for genomic analysis.

### 2.4.1 System Characteristics

1. Data cleansing handled for the user,
2. Exception handling throughout - not only to prevent halting of Big Data tasks but to fit the scientific context appropriately,
3. Extensible for implementing additional features in the future.

### 2.4.2 Features

#### 2.4.2.1 *Must*

1. API – connecting to NCBI’s databases for either DNA or protein datasets (**Added**),
2. Normalised Frequency Algorithm – calculate to display similarities and or differences,
3. Protein Synthesis – class for converting DNA dataset into Protein for all other tasks,
4. Multiple Sequence Alignment Visualisation – using Bokeh library,
5. Molecular Docking – demo and report on AutoDock Vina (**Updated**),
6. Standard Functions – append to class containing all abstract functions (**Added**),
7. Testing – classes simulating input of application’s feature set,
8. Validation – verify each scientific approach via. existing resources and by processing genomes from outside the Coronaviridae taxon.

#### 2.4.2.2 *Should*

9. Cluster Analysis – simulated reads of genomes; PCA, t-SNE and k-means (**Updated**),
10. Pairwise Sequence Alignment – optimisation of existing alignment algorithms as well as evidence evolutionary, functional, and structural relationships (**Updated**),
11. Kelvin-2 – High Performance Computing Centre in Northern Ireland (HPC-NI). To be used for Big Data processing of Pairwise Sequence Alignment (**Removed**),
12. Alignments – built-in procedure to make .aln files for MSA (**Added**).

#### 2.4.2.3 *Could*

13. Ribosome Turing Machine – prove the ribosome is Turing-complete (**was Should**),
14. Graphical User Interface.

#### 2.4.2.4 *Won't*

15. Automate specialist work, i.e. microscopy, nuclear organisations, molecular networks.

### 2.5 Requirement Amendments

#### 1. API (**Added**)

Firstly, automation wouldn't be achieved without a standard for downloading datasets.

Data integrity is paramount in a scientific setting too. NCBI is a reputable source.

#### 4. Molecular Docking (**Updated**)

After extensive research and a brief discussion with the project supervisor, it was agreed that in-house Molecular Docking was in no way feasible for the time and resources allotted for the project. Instead, substituted with a demonstration and report of open-source software AutoDock Vina.

#### 5. Standard Functions (**Added**)

Code reduction and proceduralisation of abstract functions used regularly.

#### 9. Cluster Analysis

Simulated reads, instead of normalised frequencies of nucleotides; k-means added.

#### 10. Pairwise Sequence Alignment (**Updated**)

Optimisation of existing alignment algorithms. Aim is to reduce execution time.

#### 11. Kelvin-2 (**Removed**)

The latter was more successful than anticipated, removing the need for HPC-NI.

#### 12. Alignments (**Added**)

.aln files can be created on websites where the user uploads many individual sequence files. However, for the purposes of the Multiple Sequence Alignment tool, sequences in .aln files must be of the same length which online sources don't ensure.

#### 13. Ribosome Turing Machine (**was Should**)

Reduced priority for the introduction of essential tasks.

### 2.6 Development Technologies

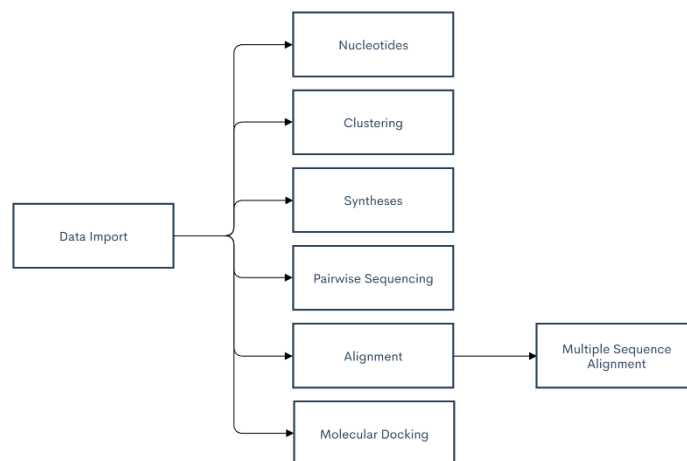
The solution will primarily be developed using Python 3.8 in PyCharm IDE (8GB RAM), along with Jupyter Notebooks and Anaconda distribution, accessed via. EEECS VM. GitLab repository will be used for version control. Docker Server for installing specialist libraries. Ubuntu 18.04 and Python 3.6.5 venv for installing SimLoRD, generating simulated reads.

## 3 Design

### 3.1 Application Overview

This illustrates the major constructs as a workflow of what processes need to be performed before the next; left-to-right, and the likely order of a user executing these; top-to-bottom.

- “Data Import” initial stage for downloading and cleansing genome or protein datasets,
- “Nucleotides” is responsible for generating DNA bases combinations, for the comparative analysis of normalised frequencies of nucleotide compositions,
- “Clustering” dimensionality reduction of genome’s simulated sequence reads,
- “Syntheses” solely translates DNA sequences into protein for further analyses,
- “Pairwise Sequencing” matches sub-sequences of similarity that may indicate evolutionary, functional, and or structural relationships,
- “Alignment” creates a .aln file of all sequences of the same biological type,
- “Multiple Sequence Alignment” visualises .aln files, e.g. for spike protein discovery,
- “Molecular Docking” demo of existing open-source molecular 3D modelling software.



*Figure 1: Composition Diagram depicting a high-level overview of the application's workflow.*

Modularity was adopted. A popular code reuse technique that separates tasks out in the application. In case of failure, other tasks may still be performed. Minimising the amount of code there is, to perform similar procedures, and thus much easier maintenance.

### 3.2 Protein Synthesis

Computationally, this involved: a data dictionary: of a key, three DNA characters; and value, the protein translation, and a loop to iterate over each character. To be of use, for analyses of protein data further on, the synthetisation component needed to be true to science. Ensuring this meant exception handling was of top priority.

Factual conditions were enforced to simulate the behaviours of a ribosome, by sequential if statements and a for loop condition. These include:

1. The DNA to protein translations themselves [17],
2. Translate only tri-nucleotides [22],
3. Synthesise one tri-nucleotide unit at a time [12].

Data handling conditions however had to be decided upon:

- Non-DNA alphabet character; prompt user to skip over character(s) or cancel process,
- Insufficient or empty DNA dataset; don't process nor save file and return,

Ultimately only hypothetical, but prevents the system from ever halting when performing big data tasks. A primary consideration mentioned for user experience. Genome datasets go through clearances to obtain an ascension number before data banks publish them [23].

### 3.3 Normalised Frequencies

This is a common technique in mathematics to derive measurements from arbitrary information for exploratory data analyses of any kind. Here, this has been applied to calculate the occurrences of nucleotides as a percentage of their wider genome.

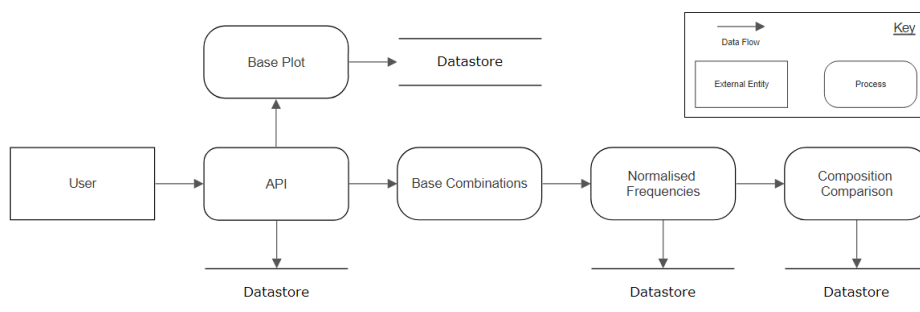


Figure 2: Data Flow diagram depicting the handling of genomic datasets for calculating normalised frequencies.

In “Nucleotides.py” is where these calculations and the composition comparison task are executed. “User” dictates what genomes to include, first either plotting their nitrogenous base pairs and saving the figures or generate all base combinations, of a given polymer length.

It’s after “Base Combinations” the application calculates “Normalised Frequencies” of the polymers, for each genome passed by the “User”. Outputs are stored and are passed to “Composition Comparison”, visualised on a line graph.

### 3.4 Bio Type

The ‘bio\_type’ variable is used throughout the code base as a flag to indicate whether the user is interested in the analysis of DNA or proteins of a given genome. Important as the location and structure of these files are different, and so require different granular processes.

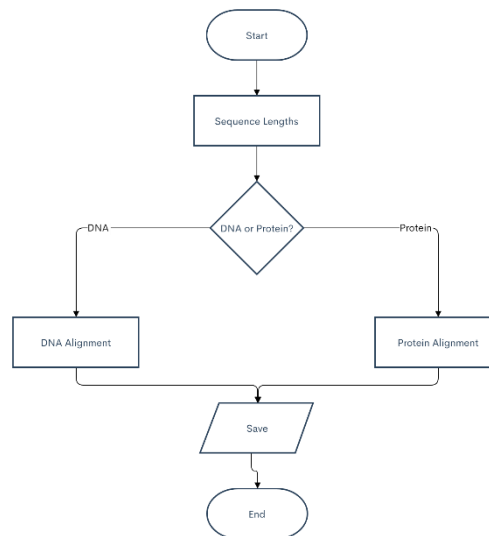


Figure 3: Flowchart of ‘bio\_type’ approach for establishing .aln files.

This was used as a flag to determine how to carry out the same contextual task but with different processes for datasets of either biological matter, DNA and protein. It can later be found across the application, i.e. ‘Alignment.py’, ‘MultipleSequenceAlignment.py’, ‘PairwiseSequencing.py’, and ‘StandardFunctions.py’.

### 3.5 Multiple Sequence Alignment

An Interactive visualisation search tool for spike protein discovery is what will enable Bioinformaticians to perform such tasks.

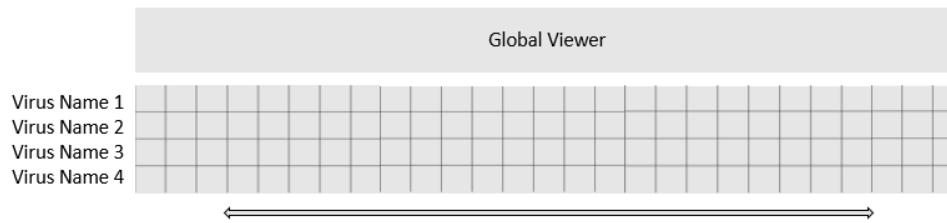


Figure 4: Multiple Sequence Alignment visualisation tool mock-up.

Gridlines are the cells for hosting individual amino-acids, colour coded for each character. The bi-directional arrow signifies movement along the  $x$ -axis for reading across the sequences. Global Viewer displays the total shared length of the sequences condensed down. Colour mapping indicates where along with the sequences there are relationships.

## 4 Implementation

### 4.1 Development Tools

#### 4.1.1 Python

The choice of language was simple, as Python hosts the most libraries for Bioinformatics. It's open-source, with plenty of free resources for learning and asking questions. Many libraries were used throughout development, including the below.

##### 4.1.1.1 Biopython

Used for constructing an API and Pairwise Sequencing; the module `Bio.pairwise2` is used for comparing and contrasting evolutionary relationships between pairs of genomes. A choice of global or local sequencing, as well as points and penalties for matching types.

##### 4.1.1.2 Bokeh

A package for developing bespoke visualisations. This was used to develop the Multiple Sequence Alignment tool. Each component of a visualisation tool is defined by a 'figure'. In

this case, two figures were made: a global view of all cells; holding the sequence characters, and an aligned sequences viewer; interactively scroll, e.g. protein motif discovery.

## **4.1.2 Integrated Development Environments**

### **4.1.2.1 PyCharm**

The first decision made, as it works well with Git for source control and is an industry-standard for Python projects. Used primarily for development and testing. Providing features practical for debugging, i.e. detecting import errors immediately, highlighted Git changes, and code coverage to help identify where parameters were passed and what values they adopted.

### **4.1.2.2 Jupyter Notebooks & Google Colaboratory**

Deploying interactive figures via. Bokeh, and plots via. AutoDock Vina only work in web browsers. All installation requirements are in the '.ipynb's.

## **4.1.4 Source Control**

GitLab worked well for separating development tasks into branches. Concise commit messages, few but regular. Some hiccups were experienced: cancelling commits, branch renaming, stashing, reverting pushes, merge conflicts, and a rebase. Most of them learnt the hard way; all were overcome.

## **4.1.3 Docker Server**

Shortly into development, it was discovered that many Python libraries would not install natively, by any means, into the project's virtual environment. Docker was deployed to install bioinformatics libraries, accompanied by Git Bash for access, and FileZilla to migrate files.

## **4.1.4 SimLoRD**

Simulates long reads. These are pseudo-random sequences from a genome. SimLoRD is given a complete genome, .fasta, and generates a .fastq file, storing these reads, each with a quality score. This can then be vectorised, across 32 columns, for computational analysis [27].



## 4.2 Code Approach

### 4.2.1 High Performance

#### 4.2.1.1 Tail Call Optimisation

In “Nucleotides.py”, `def base_combinations()` is an accumulator method for recursive function `def base_combinations_recursive()`. For each invocation, the recursive function retrieves its copy of all local variables; that is one more than the previous call.

The stack has to hold  $2k + 1$  as many variables as penultimate member  $k$ . Once a threshold is met, in traditional programming; i.e. mutable paradigm, the stack resolves itself by returning values, popping a member off the stack, for each invocation to clear memory. This is a lot more processing than is necessary.

Implemented is an accumulator function. The recursive function returns the value directly to the invoker, by linking the result of the most recent recursion to the calling frame, dropping all intermediate stack frames. There is no resolve time.

#### 4.2.1.2 Dynamic Programming

Another alternative to recursion; DP is an algorithmic technique that aims to solve optimisation problems with significantly less processing. By splitting tasks into sub-problems and storing their solutions into stack memory, instead of recalculating them recursively, reduces time complexities from exponential to polynomial.

Bottom-up tabulation approach; solving sub-problems first before optimising the solution of each parent process, scores outputs to processes and stores them in a table. These results are used to calculate the solution of its invocation. This was applied to the pairwise sequence alignment algorithms.

#### 4.2.1.3 Multi-Threading

The library ‘threading’ was used in ‘test.py’ to apply concurrency, running multiple light-weight test cases simultaneously. Overall, this gave an increase in performance. The only

negation was having to establish and delete the relevant temporary files during runtime for each ‘unittest’, to prevent inconsistent read/ write conflicts.

#### **4.2.2 Procedural Programming**

In ‘main.py’; this paradigm was adopted for ease of trialling out new implementations and possible parameters. Methods, treated as sub-routines, are called linearly to have a better sense of storyboarding the application as the intended user. Sectioned out appropriately, all method calls with their parameters are kept commented for execution.

#### **4.2.3 Exception Handling**

Essential throughout, to enforce calculations and their formatted outputs remained aligned to the science and industry standards. Sticking to one of the solution’s fundamental goals; to take out as much of the guesswork for the user as possible.

#### **4.4.3.1 Trinucleotide Protein Synthesis**

For example; as ribosomes synthesise proteins based on trinucleotide compositions in a DNA sequence, in ‘Syntheses.py’, conditions include sequence length  $n \% 3$ , non-DNA characters, etc. Value is the no. characters at the end of the sequence to discard.

#### **4.2.4 Storing Outputs**

Calculations and raw data outputs are stored in file locations. Performance-wise, this would be inefficient to pass as parameters to be stored on the stack. A file would be named appropriately at a path, opened, data read and written, and closed. The ‘data/’ folder contains directories for processes that store files under naming conventions.

### **4.3 Essential Features & Algorithms**

A description of how key functions and algorithms were implemented.

### 4.3.1 Normalised Frequencies

Essential for polynucleotide composition analysis of viral genomes. Here, they measure the number of instances of a given polymer as a percentage of its whole genome. Done by: calculating the product; no. occurrences of a given polymer multiplied by the shared length of all polymers, divided by the length of the sequence.

The number of occurrences of a  $k$ -nucleotide, in a given sequence, is denoted by  $a$ . We use  $n$  to denote the length of said sequence, that is the total number of nucleotides.

$$f = \frac{a}{n - k + 1} \quad (1)$$

Let  $x, y$  each denote a DNA sequence.  $K$  denotes the number of all possible  $k$ -nucleotides, where  $K = 4^k$ ; e.g. di-nucleotides:  $4^2 \Rightarrow K = 16$ .  $AB$  is a random di-nucleotide. Finally, the crux of the formula  $\rho_{AB}(x)$  denotes the normalised frequency of a random di-nucleotide  $AB$  in sequence  $x$ .

$$\delta(x, y) = \frac{1}{K} \sum |\rho_{AB}(x) - \rho_{AB}(y)| \quad (2)$$

Carrying out an iteration of sigma requires us to first calculate normalisation of a given di-nucleotide; for each of the chosen sequences, then subtracting them from one another; in order of which the variable sequences are passed to function delta, and finally retrieving their absolute values from 0; that is without its sign such that  $\pm x = x$ . Dividing the sum of positive differences, of the corresponding di-nucleotides, by 16.

Results provide insight into similarities amongst the sequences. Higher values indicate more occurrences of a given polymer. The closer the polymers' frequencies are across a set of genomes, the more alike they are in chemical make-up.

### 4.3.2 Cluster Analyses

Choosing the right clustering algorithm is salient for suitably classifying instances of a dataset to derive the most well representative insights possible. There are two attributes for determining the best model: dataset size; small, medium or large, and no. clusters.

Firstly, throughout the evaluation, we experiment with three to four genomes, anticipated clusters; secondly, there is a small to medium no. instances. Decided was supervised machine learning models PCA, t-SNE and k-means; including unsupervised, would be trialled.

### 4.3.3 Protein Synthesis

A Turing Machine, modelling the ribosome's quasi-mechanical movements, can be defined by 7-tuples  $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ , where:

$$Q = \{ a, c, g, t, a_a, a_c, a_g, a_t, \dots, t_t, a_{a_a}, \dots, a_{a_t}, \dots, a_{t_t}, \dots, t_{t_t} \}$$

$$\Sigma = \{ A, C, G, T \}$$

$$\Gamma = \{ A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y, * \}$$

$$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{ L, R \}$$

$q_0 \in Q$  is the start state.

$\{ t_{a_a}, t_{a_g}, t_{g_a} \} \in Q$  are accept states.

$q_{reject} \in Q$  is implicit.

(5)

Specifically, a Non-Deterministic Multi-Tape Turing Machine.

Abstraction of all possible states declared in set  $Q$ .

Accepts states are:  $\{ t_{a_a}, t_{a_g}, t_{g_a} \}$  as these represent the stop codons, when translation ends.

Reject states in Turing Machines are implicit; missing transitions denote reject.

Naming scheme of defined states have recurring subscripts; e.g. state  $a_{c_g}$  can be read in order as tri-nucleotide  $ACG$  in a DNA sequence, such that subscript  $g$  is a member of  $a_c$ . The total limit being a secondary subscript, due to the nature of the ribosome dealing only with tri-nucleotide compositions for protein synthesis.

Operators associated with transitions:

$A \rightarrow A, R$  – Read  $A$  from tape cell, write  $A$ , and move the tape head to the right on DNA tape.

$A \sqcup \rightarrow A S, R R$  – First tape (DNA) read  $A$  from tape cell, write  $A$ ; second tape (Protein) read  $\sqcup$  from tape cell, write  $S$ , and move both tape heads to the next right cell along each tape.

$\varepsilon \rightarrow \varepsilon, \varepsilon$  – After protein is written, move to start state without altering any tapes.

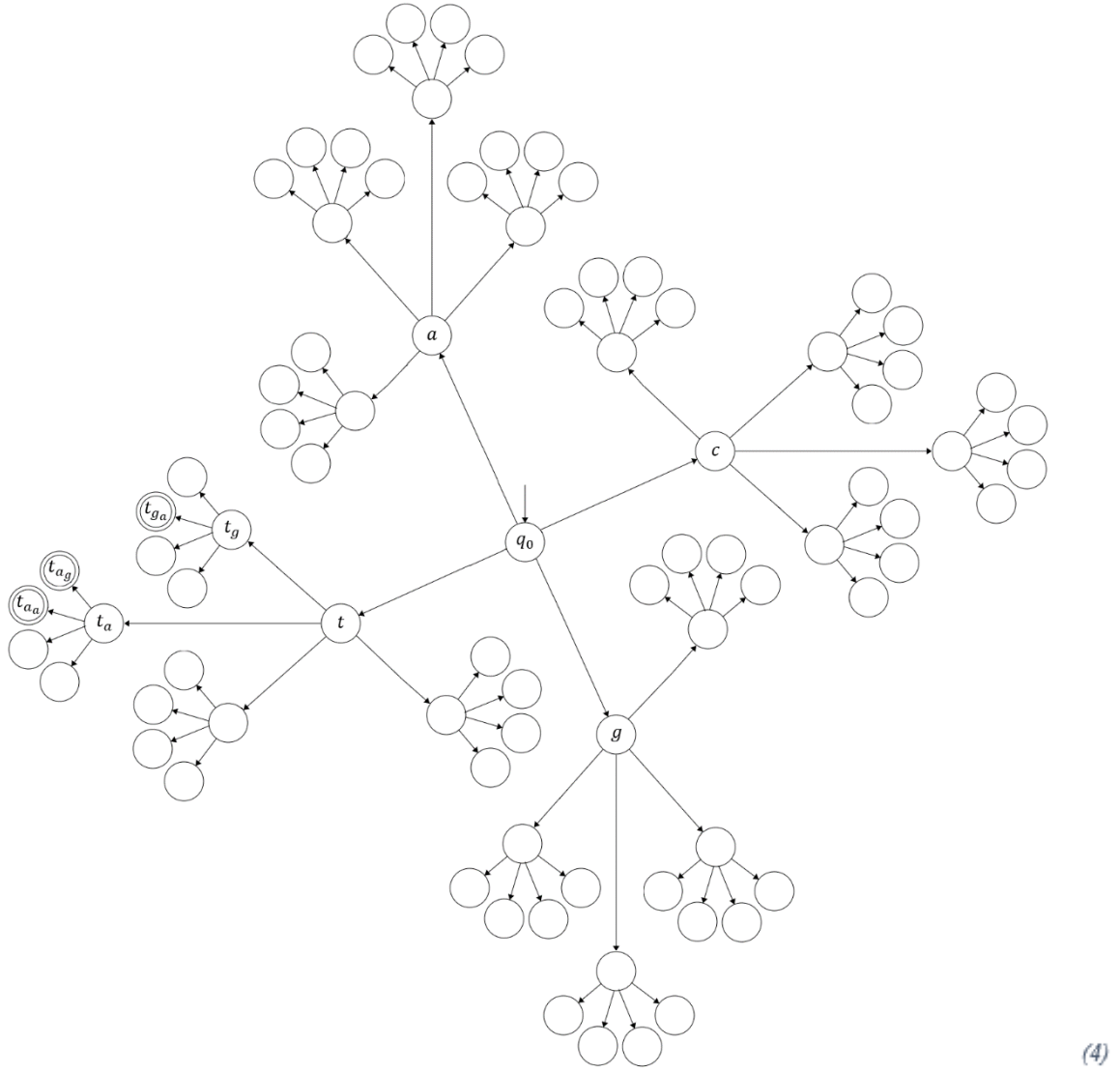


Figure 5: Non-Deterministic Multi-Tape Turing Machine represents a ribosome's role of synthesising proteins. An incomplete schematic. Table 1 demonstrates the entirety of the TM's transitions.

Comprising of two Tapes, each having its own Tape Head. We initialise the first tape, denoting a DNA sequence for input; assuming mRNA start codon *AUT* has been met, as mRNA is of a different alphabet. Second tape, denoting the output protein, initially is empty; comprising of blank entries '□', ready to be overwritten.

A	G	T	C	C	A	G	T	G	T	A	A
□	□	□	□	□	□	□	□	□	□	□	□

Figure 6: Turing read and write tapes before computation.

(5)

$\Sigma \backslash Q$	$A$	$C$	$G$	$T$	$\varepsilon$	$\Sigma \backslash Q$	$A$	$C$	$G$	$T$	$\varepsilon$
$q_0$	$(a, A, R)$	$(c, C, R)$	$(g, G, R)$	$(t, T, R)$	$\emptyset$	$a_{a_a}, \dots, g_{t_t}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$(q_0, \varepsilon, \varepsilon)$
$a$	$(a_a, A, R)$	$(a_c, C, R)$	$(a_g, G, R)$	$(a_t, T, R)$	$\emptyset$	$t_{a_a}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$c$	$(c_a, A, R)$	$(c_c, C, R)$	$(c_g, G, R)$	$(c_t, T, R)$	$\emptyset$	$t_{a_c}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$(q_0, \varepsilon, \varepsilon)$
$g$	$(g_a, A, R)$	$(g_c, C, R)$	$(g_g, G, R)$	$(g_t, T, R)$	$\emptyset$	$t_{a_g}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$t$	$(t_a, A, R)$	$(t_c, C, R)$	$(t_g, G, R)$	$(t_t, T, R)$	$\emptyset$	$t_{a_t}, \dots, t_{c_t}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$(q_0, \varepsilon, \varepsilon)$
$a_a$	$(a_{a_a}, A, R)$	$(a_{a_c}, C, R)$	$(a_{a_g}, G, R)$	$(a_{a_t}, T, R)$	$\emptyset$	$t_{g_a}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$a_c$	$(a_c, A, R)$	$(a_{c_c}, C, R)$	$(a_{c_g}, G, R)$	$(a_{c_t}, T, R)$	$\emptyset$	$t_{g_c}, \dots, t_{t_t}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$(q_0, \varepsilon, \varepsilon)$
$a_g$	$(a_{g_a}, A, R)$	$(a_{g_c}, C, R)$	$(a_{g_g}, G, R)$	$(a_{g_t}, T, R)$	$\emptyset$						
$a_t$	$(a_{t_a}, A, R)$	$(a_{t_c}, C, R)$	$(a_{t_g}, G, R)$	$(a_{t_t}, T, R)$	$\emptyset$						
$c_a$	$(c_{a_a}, A, R)$	$(c_{a_c}, C, R)$	$(c_{a_g}, G, R)$	$(c_{a_t}, T, R)$	$\emptyset$						
$c_c$	$(c_{c_a}, A, R)$	$(c_{c_c}, C, R)$	$(c_{c_g}, G, R)$	$(c_{c_t}, T, R)$	$\emptyset$						
$c_g$	$(c_{g_a}, A, R)$	$(c_{g_c}, C, R)$	$(c_{g_g}, G, R)$	$(c_{g_t}, T, R)$	$\emptyset$						
$c_t$	$(c_{t_a}, A, R)$	$(c_{t_c}, C, R)$	$(c_{t_g}, G, R)$	$(c_{t_t}, T, R)$	$\emptyset$						
$g_a$	$(g_{a_a}, A, R)$	$(g_{a_c}, C, R)$	$(g_{a_g}, G, R)$	$(g_{a_t}, T, R)$	$\emptyset$						
$g_c$	$(g_{c_a}, A, R)$	$(g_{c_c}, C, R)$	$(g_{c_g}, G, R)$	$(g_{c_t}, T, R)$	$\emptyset$						
$g_g$	$(g_{g_a}, A, R)$	$(g_{g_c}, C, R)$	$(g_{g_g}, G, R)$	$(g_{g_t}, T, R)$	$\emptyset$						
$g_t$	$(g_{t_a}, A, R)$	$(g_{t_c}, C, R)$	$(g_{t_g}, G, R)$	$(g_{t_t}, T, R)$	$\emptyset$						
$t_a$	$(t_{a_a}, A, R)$	$(t_{a_c}, C, R)$	$(t_{a_g}, G, R)$	$(t_{a_t}, T, R)$	$\emptyset$						
$t_c$	$(t_{c_a}, A, R)$	$(t_{c_c}, C, R)$	$(t_{c_g}, G, R)$	$(t_{c_t}, T, R)$	$\emptyset$						
$t_g$	$(t_{g_a}, A, R)$	$(t_{g_c}, C, R)$	$(t_{g_g}, G, R)$	$(t_{g_t}, T, R)$	$\emptyset$						
$t_t$	$(t_{t_a}, A, R)$	$(t_{t_c}, C, R)$	$(t_{t_g}, G, R)$	$(t_{t_t}, T, R)$	$\emptyset$						

Table 1: Transition Table represents all possible state-transitions, based on the current state and input. Instances in grey are the TM's accept states.

Consider the input string: *AGTCCAGTGTAA*

Let's determine whether it would be accepted by the defined Turing Machine.

DNA – Initial Input:

DNA – de facto Read:

Protein – de facto Write:

$q_0$ AGTCCAGTGTAA

$\vdash q_0$ AGTCCAGTGTAA

$\vdash q_0$  □□□□□□□□□□□□□□

$\vdash Aa$ GTCCAGTGTAA

$\vdash a$  □□□□□□□□□□□□□□

$\vdash AGa_g$ TCCAGTGTAA

$\vdash a_g$  □□□□□□□□□□□□□□

$\vdash AGTa_{gt}$ CCAGTGTAA

$\vdash Sa_{gt}$  □□□□□□□□□□□□□□

$\vdash AGTq_0$ CCAGTGTAA

$\vdash Sq_0$  □□□□□□□□□□□□□□

$\vdash AGTc_c$ AGTGTAA

$\vdash Sc$  □□□□□□□□□□□□□□

$\vdash AGTCCc_c$ AGTGTAA

$\vdash Sc_c$  □□□□□□□□□□□□□□

$\vdash AGTCCA_{ca}$ GTGTAA

$\vdash SPc_{ca}$  □□□□□□□□□□□□□□

$\vdash AGTCCAq_0$ GTGTAA

$\vdash SPq_0$  □□□□□□□□□□□□□□

$\vdash AGTCCAGg$ GTGTAA

$\vdash SPg$  □□□□□□□□□□□□□□

$\vdash AGTCCAGTg_t$ GTAA

$\vdash SPg_g$  □□□□□□□□□□□□□□

$\vdash AGTCCAGTG_{tg}$ TAA

$\vdash SPVg_{tg}$  □□□□□□□□□□□□□□

$\vdash AGTCCAGTGq_0$ TAA

$\vdash SPVq_0$  □□□□□□□□□□□□□□

$\vdash AGTCCAGTGTt$ AA

$\vdash SPVt$  □□□□□□□□□□□□□□

$\vdash AGTCCAGTGTa_a$ A

$\vdash SPVt_a$  □□□□□□□□□□□□□□

Accepted

$\vdash AGTCCAGTGTAA_{ta}$

$\vdash SPV * t_{ta}$  □□□□□□□□□□ (6)

At the end, the tapes contain the following contents:

A	G	T	C	C	A	G	T	G	T	A	A
S	P	V	*	□	□	□	□	□	□	□	□

Figure 7: Turing read and write tapes after computation.

(7)

#### 4.3.4 Pairwise Sequence Alignment

Compares two sequences with the objective of finding the optimal alignment of which yields the highest score. Performed by aligning the first character of each sequence, or any selected  $n^{\text{th}}$  place subsets, against one another to determine whether that alignment would have spawned due to evolutionary relationship or by statistical odds [13]. Scoring assesses matches, mismatches, and gaps.

Insights such as what common traits are shared across species? How similar are two species genetically? What are their distinct mutations?

Alignment algorithms can compute both nucleotide and protein sequences. Resulting aligned sequences; outputs from all algorithms abide by the following terms.

Let  $\Sigma = \{ A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V \}$  be the alphabet of amino acids in proteins,  $\{ * \}$  denote any stop codon, and  $\{ - \}$  be a gap in the alignment [2].

Let A and B be aligned protein sequences over  $\Sigma \cup \{ * \} \cup \{ - \}$  as a two-row matrix s.t. [2]:

1. Each row contains all members of one sequence in order, e.g.

$$\forall i: 1 \leq i \leq \text{size}(\text{row}_m). \forall j: 1 \leq j \leq \text{size}(A). \text{row}_m[i] = A[j]$$

2. Each column contains at least one member of  $\Sigma$ , e.g.

$$\exists i: 1 \leq i \leq \text{size}(\text{column}_n). \exists x. x \in \Sigma \wedge \text{column}_n[i] = x$$

3. One or more gaps ‘-’ may appear between any two members of  $\Sigma$ , in each row [2]. (8)

NB: a row or column of a matrix can be treated as a *mathematical* sequence.

$$\text{alignment} = \begin{bmatrix} V & H & - & L & T & E \\ V & A & P & L & - & E \end{bmatrix} \quad (9)$$

The first, fourth, and sixth columns are matches, the second column is a mismatch, the third column is an insertion, and the fifth column is a deletion.



#### 4.3.4.1 Needleman-Wunsch Algorithm

A global alignment algorithm; considers the entirety of the two sequences in search of the optimal alignment. Trace-back commences at the bottom-right index. Scores can be negative.

Let's assert the far-left and upper-most boundaries to be integers linearly decreasing from 0.

$$\forall i: 1 \leq i \leq |x|. \forall j: 1 \leq j \leq |y|. M_{i,0} = -i \wedge M_{0,j} = -j \quad (10)$$

$$M_{i,j} = \max \begin{cases} M_{i-1,j} & \text{above} \\ M_{i,j-1} & \text{left} \\ M_{i-1,j-1} & \text{above left} \end{cases} \quad (11)$$

	-	C	A	C	T	G	A
-	0	-1	-2	-3	-4	-5	-6
A	-1	<b>0</b>	0	-1	-2	1	-3
G	-2	<b>1</b>	<b>2</b>	<b>0</b>	<b>1</b>	0	-1
A	-3	0	1	0	<b>1</b>	-1	1
T	-4	1	0	1	-1	<b>0</b>	0
C	-5	-3	1	-1	-1	<b>1</b>	<b>0</b>

Table 2.1: Dynamic Programming matrix to compute the optimal score of Needleman-Wunsch Algorithm.

	-	C	A	C	T	G	A
-							
A		×	←	←	←	←	←
G		↑	←	←	←	↖	↖
A		↑	↑	↖	↑	↖	↖
T		↑	↖	↖	↑	↖	↑
C		↑	↖	↖	↖	↑	←

Table 2.2: Traceback matrix of Needleman-Wunsch Algorithm.

Once the algorithm has traced back to index [1, 1], the computation terminates. The resulting alignment includes all characters from both sequences, as this is a global alignment algorithm.

$$\text{alignment} = \begin{bmatrix} - & A & G & A & T & C & - \\ | & & & \cdot & | & \cdot & \\ C & A & - & C & T & G & A \end{bmatrix} \quad (12)$$

#### 4.3.4.2 Smith-Waterman Algorithm

A local alignment algorithm; discovers one-to-many alignments that best highlight the most similar no. regions between the pair. Trace-back commences at the index of the largest value. Scores must be positive, i.e.  $\geq 0$ .

$$\forall i: 1 \leq i \leq |x|. \forall j: 1 \leq j \leq |y|. M_{i,0} = 0 \wedge M_{0,j} = 0 \quad (13)$$

$$M_{i,j} = \max \begin{cases} M_{i-1,j} & \text{above} \\ M_{i,j-1} & \text{left} \\ M_{i-1,j-1} & \text{above left} \\ 0 & \text{otherwise} \end{cases} \quad (14)$$

	-	C	A	C	T	G	A
-	0	0	0	0	0	0	0
A	0	0	0	0	0	1	0
G	0	1	2	0	1	0	0
A	0	0	1	0	1	0	1
T	0	1	0	1	0	0	0
C	0	0	1	0	0	1	0

Table 2.3: Dynamic Programming matrix to compute the optimal score of Smith-Waterman Algorithm. This same scenario results in the same traceback matrix as Table 2.2.

The result is usually a sub-alignment of the two sequences, as this is a local alignment algorithm. Excluding some start and end characters as they are normally only aligned by gaps.

$$alignment = \begin{bmatrix} A & G & A & T & C \\ | & & \cdot & | & \cdot \\ A & - & C & T & G \end{bmatrix} \quad (15)$$

Both alignment algorithms have a time complexity of  $O(nm)$ . If any two or more of the elements above, left, and above-left are optimal, with respect to the current index, then the above-left is always chosen. Regardless if above-left is the lowest value.

$M_{i-1,j-1}$	$M_{-i,j}$
$M_{i,-j}$	$M_{i,j}$

0	1
1	-1

...	...
...	↖

(16)

An algorithm outputs aligned sequences, respect to each other, only. Illustration of relations: match; '|', mismatch; '.', and gap; '-', are done by a formatter method. Then placed between the pair of now aligned sequences, as its own row.

## 5 Testing

### 5.1 Unit Testing

Checks an individual function yields the correct output. Unit tests were explicitly performed on functions that were: reusable; invoked across the application, exception handling; inputs or values are to be prevented, file IO; stores data to or moves data from one file, for processing, to another file directory, or algorithmic; calculating and or outputting new data, i.e. numbers or sequences. ‘StandardsFunctions.py’ harbours the most extensive unit testing.

Unit tests inspired many enhancements to the solution. As an example; `def protein()` in ‘Syntheses.py’, hypothetically a DNA input sequence could have non-DNA characters before translation. Rectified by implementing `def alphabet_check()`, can be used for other types.

### 5.2 Usability Testing

In this example, we pass inputs ‘db’ and ‘ids’ to the API for the database and datasets we want to import from NCBI’s GenBank. Successfully cleansed and stored in ‘src/’, for rename.

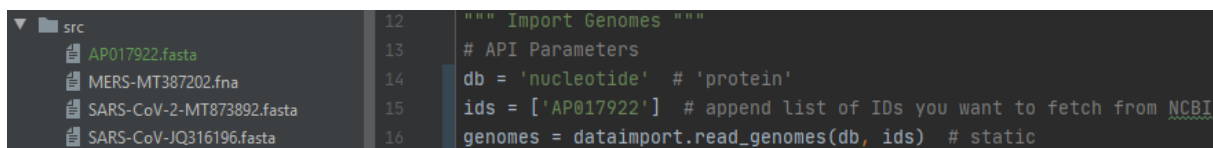
The image is a screenshot of a code editor with a dark theme. On the left, a file explorer shows a directory named 'src' containing four files: 'AP017922.fasta', 'MERS-MT387202.fna', 'SARS-CoV-2-MT873892.fasta', and 'SARS-CoV-JQ316196.fasta'. On the right, a code editor shows Python code. Line 12 has a docstring 'Import Genomes'. Line 13 has a comment '# API Parameters'. Line 14 defines 'db = 'nucleotide'' with a comment '# 'protein''. Line 15 defines 'ids = ['AP017922']' with a comment '# append list of IDs you want to fetch from NCBI'. Line 16 defines 'genomes = dataimport.read\_genomes(db, ids)' with a comment '# static'.

Figure 8: Connecting to ‘nucleotide’ database to download genome ‘AP017922’ (right). Output in ‘src/’ (left)

Tested for all functions in the demo; where the filename is simply passed as a string literal.

### 5.3 Integration Testing

Tests aspects of the application are extensible for new features. This was trialled and proved in ‘Clustering.py’ and ‘PairwiseSequencingAlgorithms/’. In ‘Clustering.py’, appending a clustering algorithm was as simple as defining a function of the same parameters and invoking it in `def run()`. Adding a new pairwise sequence algorithm required creating a separate script, and simply invoking all of the operative methods available in ‘tasks.py’.

## 5.4 Performance Testing

The primary metric used throughout development was the time elapsed. Measuring every implementation throughout development, to ensure computational complexity was reduced yet yielding the same effect. We assess the performance of optimising pairwise sequencing alignment algorithms in this way, in [Section 6.5.1](#).

# 6 System Evaluation and Experimental Results

## 6.1 Success Summary

Judgement of success is based on the implementations delivered against the requirements set as a solution to the problem area. The emphasis on development was to create a software product for Biologists considering Bioinformatics as a career, without knowledge of data science; thus filling the skill gap that is preventing employment rates from fulfilling demands.

“Performing invaluable analyses on novel viruses, within a moment.” A working range of different research techniques were all successfully implemented, as per requirements.

Systemised to establish an appropriate data analysis workflow, to be used in many scenarios.

“Study protein-protein and protein-nucleic acid recognition” (p. 429) [18]. An API for downloading nucleotide and protein datasets, pairwise sequence alignment algorithms, and multiple sequence alignment support this. The option to generate protein data, when not yet publically available, further supports the goal of “*analysis within a moment*”.

Data handling was stated as a key system characteristic. The data import stage: the API connecting to NCBI’s databases, this ensures data integrity; a reputable source being a scientific method, and data cleansing; before storage.

Multiple Sequence Alignment viewer achieving: “Visualization interfaces (for information visualization and scientific visualization)” (p. 434) [18] and comparative analysis, along with adaptive composition line graphs, axes scale depending on the level of granularity.

Lastly, pairwise sequence alignment algorithms were successfully optimised and are an example of extensibility for future implementations.

## 6.2 Nucleotide Composition

### 6.2.1 Nitrogenous Base Pairs

GC and AT content, that's guanine-cytosine and adenine-thymine, are percentages of nitrogenous bases in DNA. GC-content is a basic indicator of various structural properties of a given DNA sequence, indicative of higher biological functioning [21].

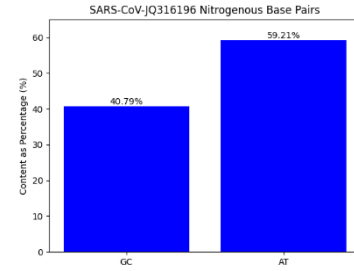


Figure 9: GC and AT content of SARS-CoV.

### 6.2.2 Normalised Frequencies

Outputs include Staphylococcus Aureus's composition, a bacteria completely unrelated to the Coronaviridae taxon, for demonstrative purposes.

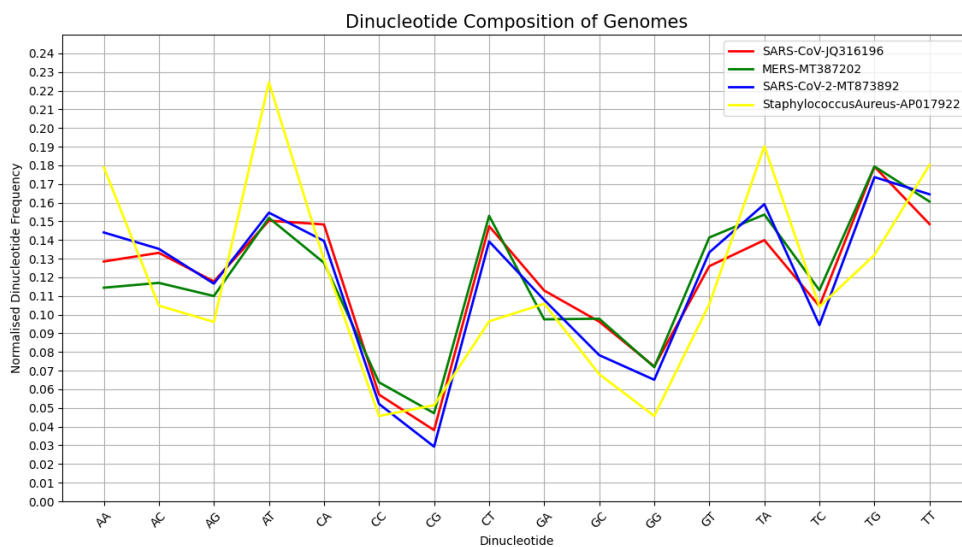


Figure 10: Line graph comparing di-nucleotide composition of SARS-CoV, MERS, SARS-CoV-2 viruses and Staphylococcus Aureus bacteria. The theoretical  $p$ -value of a given random DNA sequence is 1.0.

We can observe that there are staggering similarities in di-nucleotide composition between the three Coronaviridae, bar from the Staphylococcus. This is an idealistic result in confirming the belief that the novel coronavirus possesses similar characteristics amongst past viruses and has been classified appropriately into the correct taxon.

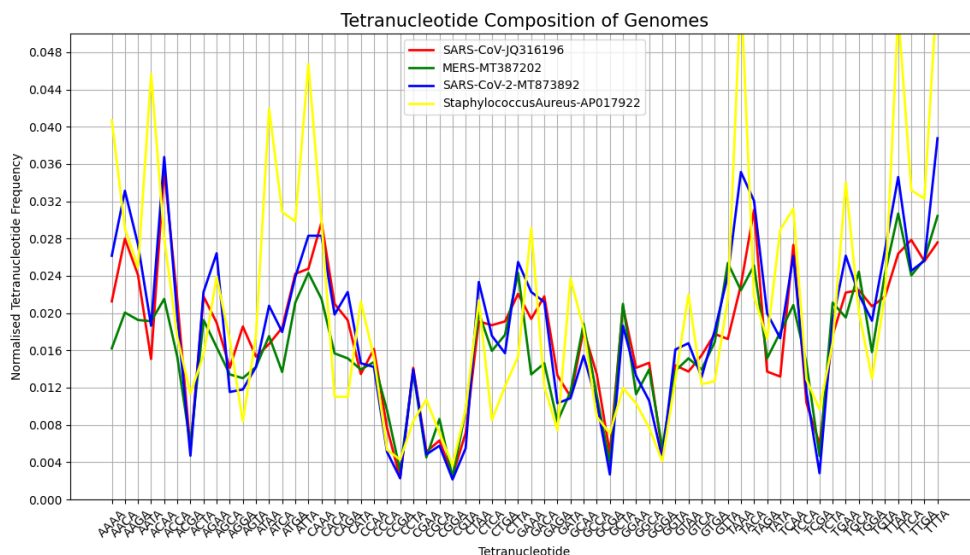


Figure 11: Line graph comparing tetra-nucleotide composition of SARS-CoV, MERS, SARS-CoV-2 viruses and *StaphylococcusAureus* bacteria. The theoretical  $p$ -value of a given random DNA sequence is 1.0.

Skipping to tetra-nucleotide composition; it is at this level of granularity that we begin to uncover variation and differentiate the genomes. Higher frequencies in one region means less frequency elsewhere in the genome.

SARS-CoV-2, whilst certainly following trends of the other coronaviruses, seems to have the least exaggerated frequencies; i.e. spikes, and thus, arguably, its line is closer to the centre of the graph. This suggests the virus has greater variation in its DNA than the other genomes. Notably at regions: AACA, GAAA, TAAA, TGAA and TTAA. Theory explored further, next.

This insight highlights the importance of performing analysis at different scales. We ran this from di to hexa-nucleotide compositions with much similar results. Further validating their evolutionary relationships. Overall, the coronaviruses remain very much similar in terms of chemical make-up.

### 6.3 Metagenomic Cluster Analyses

Plotting a high number of dataset features in a 2D space requires a technique known as Dimensionality Reduction; the transformation of big data into lower dimensions. Aiming to maintain mathematical relations between instances.

### 6.3.1 Principal Component Analysis Results

PCA is an orthogonal dimensionality reduction algorithm. It generates a coordinate system; for the data to be transformed onto, one vector; a.k.a. component, at a time [28]. The first component, termed the principal component, is the direction that best fits the data; i.e. that maximises the variance [28]. Subsequent components are orthogonal to components preceding it [28].

Calculated is the covariance of two variables of a given square matrix, where  $M = n \times n$ .

$$cov(X, Y) = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{x})(Y_i - \bar{y}) \quad [28] \quad (17)$$

Then find the eigenvectors and eigenvalues of the covariance matrix.

$$T(v) = \lambda v \quad [28] \quad (18)$$

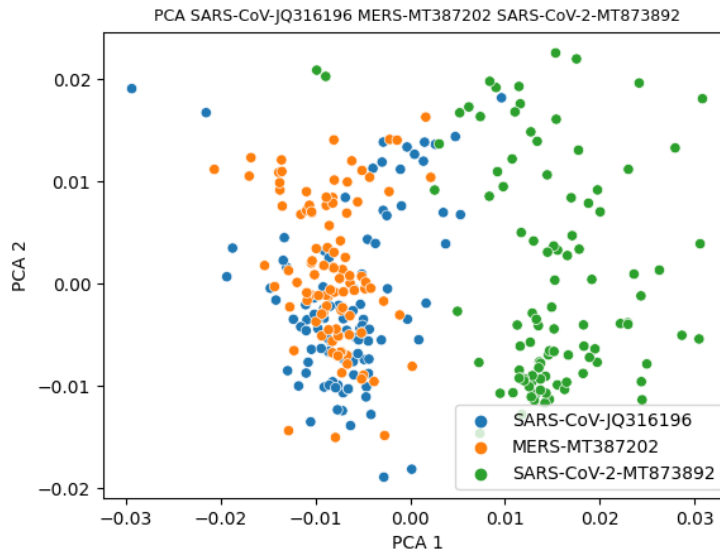


Figure 12: PCA results 3 clusters, each of 100 labelled simulated reads  $\geq 10,000bp$ , of SARS, MERS and SARS-CoV-2.

### 6.3.2 t-Stochastic Neighbour Embedding Results

t-SNE is a non-linear dimensionality reduction algorithm [29]. It generates a probability distribution, representing all relationships instances have with every other in the dataset, then compresses the information down to two or three-dimensional planes for visualisation [29].

First, the probability of an instance  $x_i$  being neighbours with an instance  $x_j$  is calculated.

$$p_{j|i} = \frac{\exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-||x_i - x_k||^2/2\sigma_i^2)} \quad (\text{Eq. 1}) [29] \quad (19)$$

Then, we calculate the joint probability distribution of each instance.

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2n} \quad (\text{Eq. 6}) [29] \quad (20)$$

These two steps are taken as preparation for dimensionality reduction.

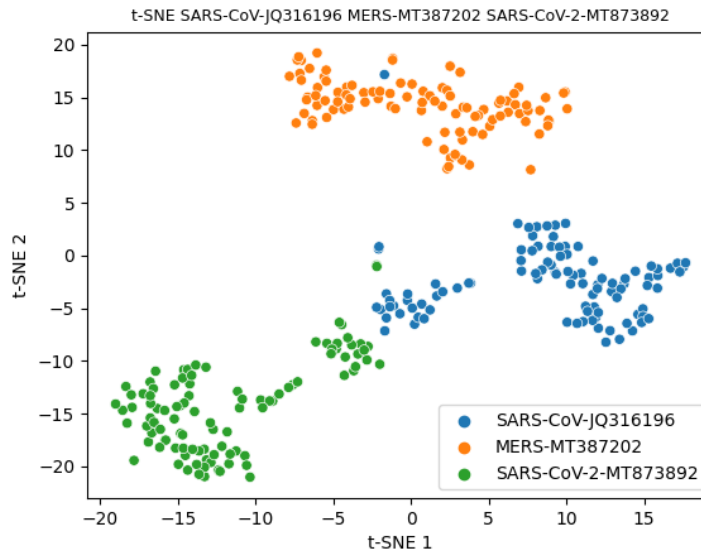


Figure 13: t-SNE results in 3 clusters, each of 100 labelled simulated reads  $\geq 10,000bp$ , of SARS, MERS and SARS-CoV-2.

### 6.3.3 Interpretations

The distance between clusters indicates the similarities amongst genomes. SARS-CoV-2 is the furthest apart cluster; thus being the most distinct species. SARS-CoV and MERS are the closest together; sharing space in PCA's plot, indicating their compositions are most alike, yet separate in t-SNE's plot, evidencing the need for their classifications. This information backs up the current taxonomy.

The sparsity of a cluster indicates how varied its genome composition is, in contrast to others. We can deduce that SARS-CoV-2 has the most varied metagenomics reads. Supporting previous analysis from earlier; tetra-nucleotide composition line graph preliminary.

In both scenarios, three distinct clusters were produced. A key difference between the two clustering algorithms is how they compute relationships, i.e. similarities or distances, across the dataset. PCA is concerned with preserving local relations, to maximise the variance



between collective data points, whereas t-SNE aims to preserve larger-scale relations; to maximise the variance between clusters.

We observe this in the above results. PCA's clusters are sparser and less separated, whilst t-SNE's clusters are less sparse and much further apart. Analysing the same data through both clustering algorithms has allowed us to see not only the similarities of these species but also further identify the variance of nucleotide composition, respectively.

### 6.3.4 k-Means

$k$ -Means is a heuristic dimensionality reduction algorithm, partitions data and assigns each to the cluster with the nearest mean centroid; the centre of said cluster [30].

Let  $x$  be the set of data points,  $c$  be centroids, and  $a(i, j)$  be binary flags; determining whether  $x(i)$  is assigned to centroid  $c(j)$  [30].

$$J(x, c, a) := \sum_{i=1}^n \sum_{j=1}^k a(i, j) \|x(i) - c(j)\|^2 \quad (\text{Eq. 1}) [30] \quad (21)$$

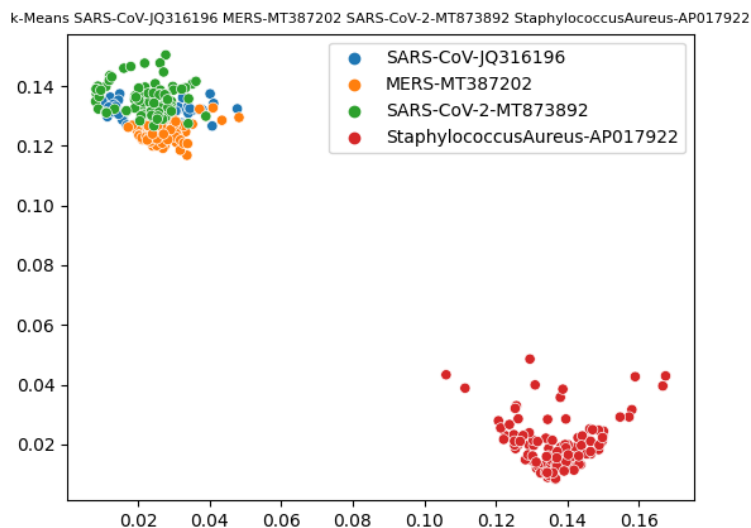


Figure 14:  $k$ -means of 4 clusters, each of labelled 100 simulated reads  $\geq 10,000bp$ , of SARS, MERS, SARS-CoV-2 and *Staphylococcus Aureus*.

Results depict the difference between coronaviruses and a staphylococcus bacteria well, reducing distortion effectively. However, we are already satisfied with our classification and differentiation of genomic data. What other value can  $k$ -means bring to research?

#### 6.3.4.1 Unsupervised Machine Learning

While these are simulated reads from already classified genomes, a common real-world research scenario may entail analyses of collected reads from an outbreak origin of a novel virus. Comparing organic reads to various classified genomes helps paint a picture in determining whether a sample is a novel strain of an existing virus or a new distinct species.

Unsupervised machine learning is to train models without labels, i.e. without knowledge of collected reads' cultures. Bioinformaticians research new findings based on the unknown.

##### 6.3.4.1 Elbow Method

Already knowing the optimal number of clusters, 4; one per genome, there may still be benefit of insight in confirming our belief using the Elbow Method. To plot  $x$  and  $y$  axes on such a graph, we train many models of varying numbers of clusters. Here, we experiment with 9  $k$ -means models, ranging from 2 to 10 clusters. We use the Within Cluster Sum of Squares method to calculate relationships between data points of a given cluster and their centroid.

WCSS is defined as the sum of distances, of cluster members and the centroid, squared.

$$WCSS = \sum_{i=1}^m (x_i - c_i)^2 \quad [31] \quad (22)$$

We should expect to see the WCSS score decrease as we increase the number of clusters. This is simply because the more centroids we add, the higher probability an individual data point has a closer distance to a given centroid than the previous  $k$ -means model.

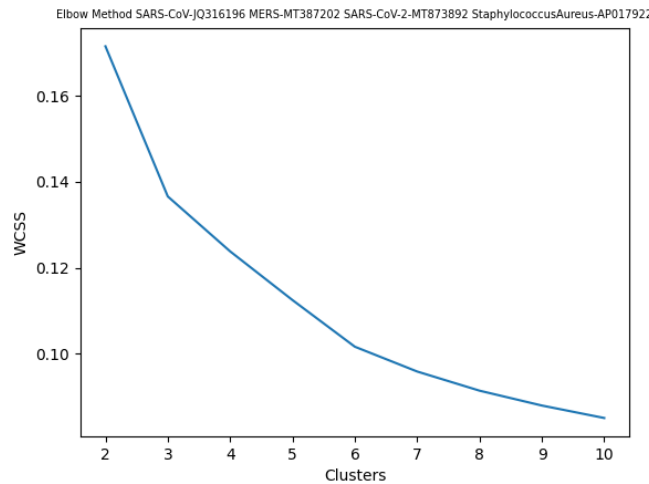


Figure 15: The elbow method instantiates many  $k$ -means models of ranging no. clusters, computing the average WCSS score of all clusters for each. The point of inflexion on the curve being 4 clusters is the optimal number.

As hypothesised, 4 clusters were optimal. The linear downward slope, from clusters 3 to 4 and 4 to 5 indicate this; followed by diminished returns is the *elbow* point.

#### 6.3.4.2 Centroids

Having confirmed the optimal number of clusters, we experiment with whether *k*-means modelling is capable of classifying unlabelled metagenomics reads.

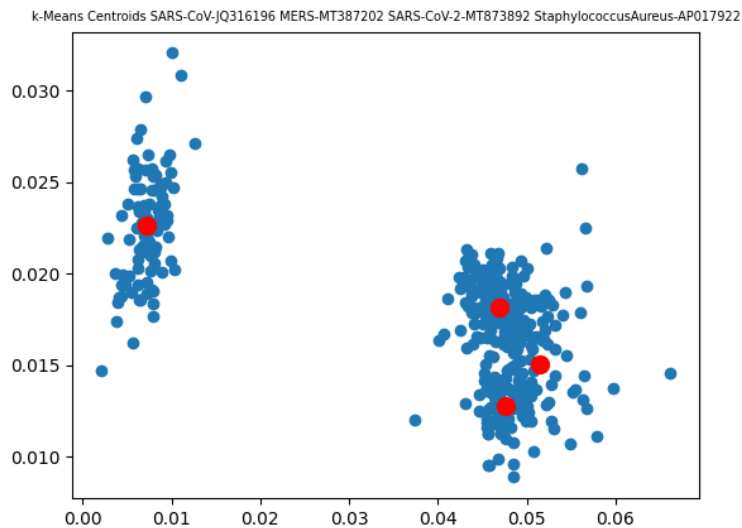


Figure 16: *k*-means results in 4 clusters, each of 100 unlabelled simulated reads  $\geq 10,000$ bp, of SARS, MERS, SARS-CoV-2 and Staphylococcus Aureus.

Interesting to note are the centroids, bottom-right. It appears the top centroid has substantially more data points closest to it, having brought closer its neighbouring clusters' data points. Considering all genomes have 100 simulated reads each, this suggests the three clusters share similar information.

We can deduce that the unsupervised model can discern the correct number of classifications for an unlabelled dataset, i.e. it can accurately predict new genomes for real-world use.

## 6.4 Protein Synthesis

Judging the correctness of the implementation; we assess the first 300 characters of each genome. Comparing outputs to NCBI's protein translation [tool](#), [here](#).

### SARS-CoV-JQ316196:

```
ATATTAGGTTTTTACCTACCCAGGAAAAGCCAACCAACCTCGATCTCTGTAGATCTGTTCTCTAAACGAACTTT
AAAATCTGTGTAGCTGTCGCTCGGCTGCATGCCTAGTGCACCTACGCAGTATAAACAATAATAATTTTACTGTC
GTTGACAAGAAACGAGTAACTCGTCCCTCTCTGCAGACTGCTTACGGTTTCGTCCGTGTTGCAGTCGATCATCA
GCATACCTAGGTTTCGTCCGGGTGTGACCGAAAGGTAAGATGGAAGCCTGTTCTTGGTGTCAACGAGAAAACA
```

Solution:

```
ILGFYLPKRSQPTSI SCRSL*TNFKICVAVARLHA*CTYAV*TIINF TVVDKKRVTRPSSADCLRFRPCCSRSS
AYLGFVRV*PKGKMESLVLGVNEKT
```

NCBI:

```
ILGFYLPKRSQPTSI SCRSL*TNFKICVAVARLHA*CTYAV*TIINF TV 50
VDKKRVTRPSSADCLRFRPCCSRSSAYLGFVRV*PKGKMESLVLGVNEKT 100
```

Total length= 100

### MERS-MT387202:

```
GATTTAAGTGAATAGCTTGGCTATCTCACTTCCCCTCGTTCTCTTGCACTACTTTGATTTTAACGAACTTAAATA
AAAGCCCTGTTGTTTAGCGTATTGTTGCACTTGTCTGGTGGGATTGTGGCATTAATTTGCCTGCTCATCTAGGCA
GTGGACATATGCTCAACACTGGGTATAATTCTAATTGAATACTATTTTTCAGTTAGAGCGTCGTGTCTCTTGTA
GTCTCGGTCACAATATACGGTTTCGTCCGGTGCGTGCAATTCGGGGCACATCATGTCTTTCGTGGCTGGTGTGA
```

Solution:

```
DLSE*LGYLTSRSLAVL*F*RT*IKALLFSVLLHLSGGIVALICLLI*AVDICSTLGIILIEYYFSVRASCLLY
VSVTIYGFVRCVAIRGTSCLSWLV*
```

NCBI:

```
DLSE*LGYLTSRSLAVL*F*RT*IKALLFSVLLHLSGGIVALICLLI*A 50
VDICSTLGIILIEYYFSVRASCLLYVSVTIYGFVRCVAIRGTSCLSWLV* 100
```

Total length= 100

### SARS-CoV-2-MT873892:

```
TTGTAGATCTGTTCCTAAACGAACCTTAAAACTGTGTGGCTGTCACCTCGGCTGCATGCTTAGTGCACTCACGC
AGTATAATTAATAACTAATTACTGTGCTGACAGGACACGAGTAACCTCGTCTATCTCTGCAGGCGCTTACGGTT
TCGTCCGTGTTGCAGCCGATCATCAGCACATCTAGGTTTGTCCGGGTGTGACCGAAAGGTAAGATGGAGAGCCT
TGTCCCTGGTTTCAACGAGAAAACACACGTCCAACCTCAGTTTGCCTGTTTACAGGTTTCGCGACGTGCTCGTACG
```

Solution:

```
L*ICSLNEL*NLCGCHSAACLVHSRSIINN*LLSLTGHE*LVYLLQAAYGFVRVAADHQHI*VLSGCDRKVRWRA  
LSLVSTRKHTSNSVCLFYRFATCSY
```

NCBI:

```
L*ICSLNEL*NLCGCHSAACLVHSRSIINN*LLSLTGHE*LVYLLQAAYG 50  
FVRVAADHQHI*VLSGCDRKVRWRALSLVSTRKHTSNSVCLFYRFATCSY 100
```

Total length= 100

## 6.5 Pairwise Sequence Alignment

### 6.5.1 Algorithmic Optimisation

Comparing the elapsed times is the best metric for judging algorithmic efficiency. Passing the same datasets and length; the first 100 nucleotides of SARS-CoV-JQ316196 and SARS-CoV-2-MT873892, ensured validity in these results.

DNA	
Algorithm	Time
Needleman-Wunsch	0.0312s
Smith-Waterman	0.0469s
Bio.pairwise2	2.4062s

Protein	
Algorithm	Time
Needleman-Wunsch	0.0312s
Smith-Waterman	0.0312s
Bio.pairwise2	2.1094s

Tables 3.1 & 3.2: Optimised Needleman-Wunsch and Smith-Waterman versus Bio.pairwise2 in process time, trialling the first 100 nucleotides and polynucleotides of genome pairs SARS-CoV and SARS-CoV-2.

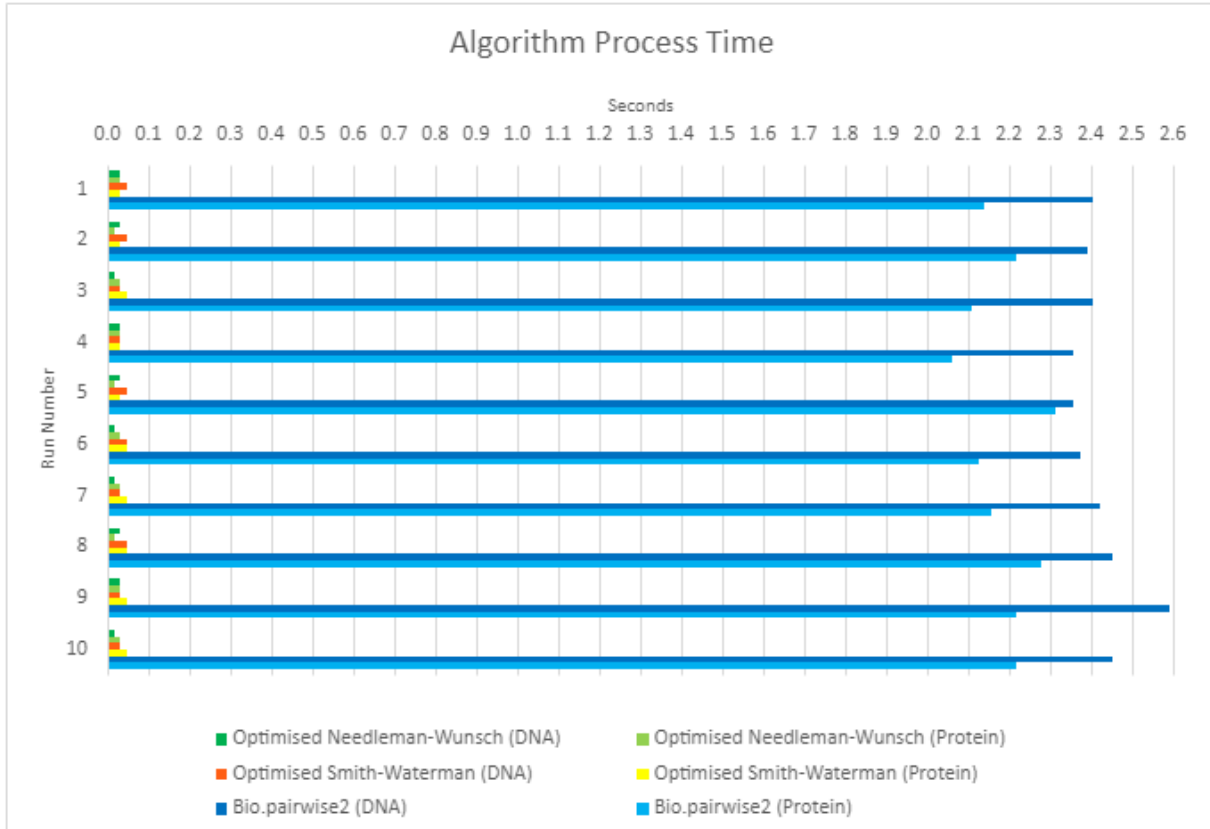


Figure 17: Optimised Needleman-Wunsch and Smith-Waterman versus Bio.pairwise2 algorithm in process time, trialling the first 100 characters of DNA and protein sequences of pairs SARS-CoV and SARS-CoV-2, over 10 runs.

Observed is a stark difference between the open-source module Bio.pairwise2 and optimised algorithms Needleman-Wunsch and Smith-Waterman. An experimental quantity, when in reality global alignments of complete genomes would take significantly longer. This test case suffices however, remember their time complexities are  $O(nm)$ . They have no exponentials.

Implementations benefitted from a menagerie of code practices, including but not limited to: reduced calculations; storing indexes, code reuse; less storage on stack memory, conditional while loops; without for loops' declarations, return early strategy; exiting a function upon exception, critical for saving time in sub-processes.

These same code practices can be applied to further pairwise sequence alignment algorithms.

### 6.5.2 Genomic Relationships

Since we are interested in comparing whole genomes, we use the Needleman-Wunsch global alignment algorithm. Controlled is the points scheme, match: 1, mismatch:  $-1$  and gap:  $-1$ .

Expected are identities  $\sim \geq 75\%$ , as we are analysing closely related coronaviruses. SARS-CoV and SARS-CoV-2 genomes should yield the highest scores.

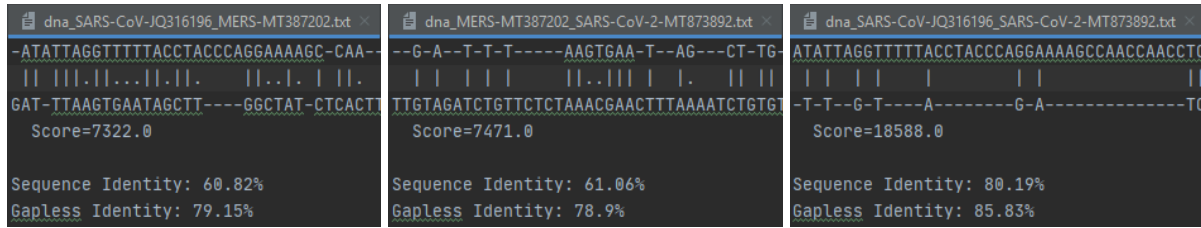


Figure 18: Pairwise sequence alignments of the highest scores yielded from the first 100 nucleotides, of genome datasets SARS-CoV, MERS and SARS-CoV-2 using Needleman-Wunsch.

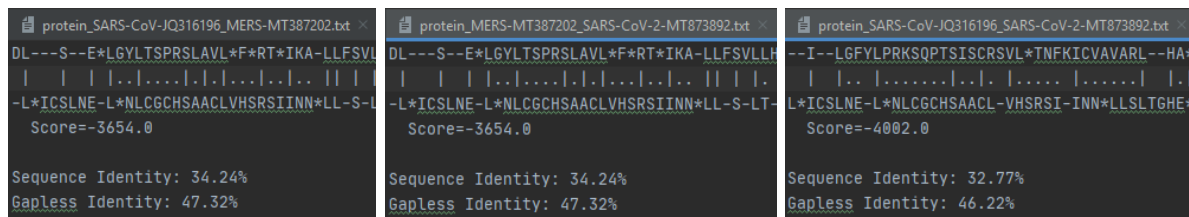


Figure 19: Pairwise alignments of the highest scores yielded from the first 100 polypeptides, of synthesised protein datasets SARS-CoV, MERS and SARS-CoV-2 using Needleman-Wunsch.

Pairwise sequencing evidences that viruses SARS-CoV and SARS-CoV-2 have the closest evolutionary relationship, with a top score of 18,588.

- MERS and SARS-CoV – a plausible direct relation if this were an isolated insight. This holds true as there is a shared evolution in their taxonomy but not immediate,
- MERS and SARS-CoV-2 – an equally plausible relationship as the prior. It has more gaps within its aligned sequences although negligible; a gapless identity of 78.9% and the latter 79.15%,
- SARS-CoV and SARS-CoV-2 – high score evidences the apparent relation. Pairwise sequencing of their proteins evidences they are the most functionally alike too.

If this test were to be conducted again we would assert a separate point scheme for handling of protein datasets, as all top scores are negative. Since both penalties; mismatch and gap, were only  $-1$ , the match score could be increased from 1 to 5. Current results do make sense however, as the protein alphabet holds 20 characters, while DNA holds just 4. Thus paired protein characters have a far lower probability of matching.

### 6.5.3 Comparison with EMBL-EBI

The European Bioinformatics Institute, part of the European Molecular Biology Laboratory research group, offers pairwise sequence alignment tools on their [website](#). It's unclear how their algorithms are implemented; bespoke or open-source, but a selection including those used in this solution is available.

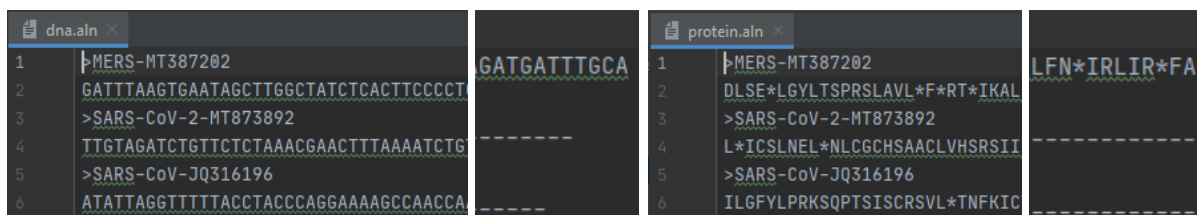
Where their solution differs from tradition is with pairwise alignment options. Instead of entering a point scheme for a match, mismatch, and gap; the only parameters adjustable are various gap scores, i.e. gap extends and gap ends. Unfortunately, without knowledge of their other points that aren't adjustable, we see different results than with our own optimised algorithms.

However, assessing their output format is interesting. They include gaps as a percentage of the total alignment length and all sub-optimal alignments trialled, which this solution doesn't. Additionally, e-mail notifications. Optimised algorithms Needleman-Wunsch and Smith-Waterman by design share the same finalised output layout; akin to Bio.pairwise2 for ease of comparison. To improve, such options could be incorporated into this application.

## 6.6 Multiple Sequence Alignment

### 6.6.1 Alignment Files

Standardised alignment files follow this [format](#). However, a caveat of the MSA visualisation is that sequences need to be of the same length. Filling end gaps with '-' resolves this.



```
dna.aln
1  |MERS-MT387202      GATGATTGCA
2  GATTTAAGTGAATAGCTTGGCTATCTCACTTCCCT
3  >SARS-CoV-2-MT873892
4  TTGTAGATCTGTTCTCTAAACGAACTTTAAATCTG
5  >SARS-CoV-JQ316196
6  ATATTAGGTTTTTACCTACCCAGGAAAAGCCAACCA

protein.aln
1  |MERS-MT387202      LFN*IRLIR*FA
2  DLSE*LGylTSPRLAVL*F*RT*IKAL
3  >SARS-CoV-2-MT873892
4  L*ICSLNEL*NLCGCHSAACLVHSRSII
5  >SARS-CoV-JQ316196
6  ILGFYLPRKSQPTSISCRSVL*TNFKIC
```

Figure 20: 'dna.aln' and 'protein.aln' file contents, demonstrating same lengths are enforced.

### 6.6.2 Visualisation Tool

Sequences are aligned by the first characters, one on each row. A new colour palette is made every run by the `def map_colors()` method.



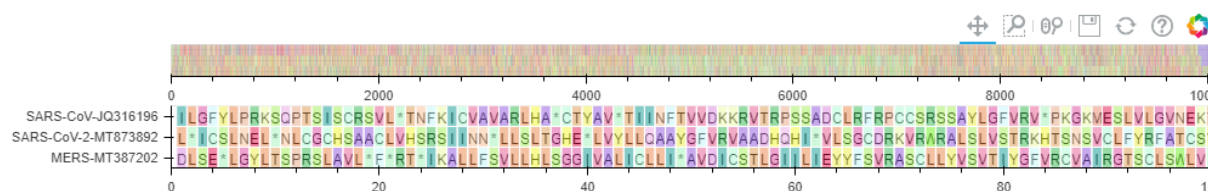


Figure 21: Multiple Sequence Alignment tool, visualising a global (top) and sequence view (main).

It is impressive how lightweight the implementation is, and is browser friendly. However, there are a few nuances that could easily be improved upon. Firstly, entry of a lower and upper-bound of sequence characters; one-indexed. No user would scroll their cursor for ~10,000 nucleotide characters. Secondly, the ability to save an area of sequences, e.g. that exhibit spike proteins, as a file instead of an image would be more convenient for further analysis.

Never the less, the vast majority of features to be expected from an MSA tool have been implemented successfully. Inspiration came from [NCBI's MSA tool](#).

## 6.7 Macromolecular Docking

Macromolecular Docking is the first stage and only stage of drug trials that require Bioinformaticians. It stress-tests all common molecules by binding them onto the subject protein strain to predict which docks the best; i.e. which has a higher likelihood of chemically bonding to a protein in the easiest and most amount of ways.

AutoDock Vina is the most popularly used molecular docking simulator. Its design aim is similar to this project's; mitigating technical intervention and holding scientific validity. Covalent bond lengths are automatically adjusted to generate 3D structures, similar to the automated MSA files task. Controllable variable by user input, where appropriate, i.e. no. atoms, angles, search space dimensions, and iterations. Most importantly, there was no scientific bias in outputs with user inputs.

Any problems encountered were stability related. Occasionally, the tool would unexpectedly terminate a run early. Integration with 'ipywidgets' library doesn't seem formal, sometimes not providing interactivity until refreshing cache. Whilst were rare, they proved to be a challenge. Furthermore, attempting protein-protein docking wasn't successful. AutoDock Vina is only intended for ligand-receptor docking.

## 6.8 Significance of Results

All tasks meaningfully presented outputs in a clear concise manner. One output holding multiple insights is what is needed to speed along analysis for Bioinformaticians.

Reproducibility of results is essential in any research, which is why connecting to NCBI's databases for our API was important. Results are also transferable, meaning the tools developed, as shown in the demo, can easily be applied to new datasets. An updated project goal, e.g. analyses of novel viruses.

## 6.9 Developed Improvements

[Galaxy](#) is an open-source platform for computational biology for non-Data Scientists. The upsides they have are a web interface, account administration, and a wider range of research tools available. The community is made up of over 30 active pro bono developers.

As open-source contributors tend to do, they may very well have stuck to existing libraries and tools, i.e. Biopython, for their back-end processing. Bio.pairwise2 module pales in comparison to the mathematically optimised pairwise sequence algorithms in this project. Although, users have to instruct which alignment algorithm to run.

What Galaxy and this project have in common is the elimination of miscellaneous technical tasks; e.g. syntheses and MSA files requiring practically no user input. Users don't face nuisances, i.e. learn to deploy and use various toolkits via command lines.

Other upsides this application has over Galaxy are different research tools and comparative analysis workflows. All output charts and data processes were developed to handle multiple datasets at once for direct side-by-side analysis in the same calculations and view.

Overall, the convenience of performing multiple analyses in one action is exactly what sets this application apart from individual open-source tools; an all-in-one suite that's extensible for immediate expansion.

## 6.10 User Feedback

[Google Form response](#). The last question: "How well does this solution fulfil your tasks as a Biologist?" got 10/10. Whilst protein and pairwise sequence alignment got high feedback for

effectiveness they received the lowest for value as a research technique, with this user. Solution can be improved upon by implementing a more research tasks in the future.

## **6.11 Implications**

### **6.11.1 Societal**

Tomorrow's Bioinformatician will still primarily have a background Biology but be expected to obtain a certification in such software. Professional certification platforms, such as Coursera, will soon supply courses to learn these tools. Biotechnology companies will deploy graduate programmes for upskilling Biologists with such skills, soon to be essential.

Change of standards in the biomedical research industry will encourage FE and HE colleges to integrate teaching such a tool into their vocational courses. Increasing employment.

### **6.11.2 Commercial**

The only incentive for consumers to change their practices over to using such software, i.e. research institutions and Biotechnology companies, is convenience. That is; is it cost-effective and is it work efficient? Like all monetised technology solutions, this focuses on convenience.

A one-stop source for all biological analyses means less equipment/ technologies to consider paying for variably as well as the admin and decision making that comes with it.

To quantify potential savings of utilising this software, The University of Tennessee Health Science Center's Molecular Bioinformatics (mBIO) research core supplies similar work. They are: "data analysis workflows", such as whole-genome sequencing at \$89.99 (p. 1) [25]; and "individual tasks", including alignments costing \$31.83 per sample, and Principal Component Analysis at \$22.28 per image, in 2021 (p. 2) [25].

They appear one of the lowest costing servicers from market research. Consumers wouldn't hesitate to perform the same task on all genome datasets if paying a subscription instead, knowing costs would be capped. So going back to traditional payments of one-time services would no longer be appealing.

The work efficiency improvement to this, innately, is continuity. As more groups of researchers adopt such a prototype into their workstream, there wouldn't be a need to invest time in scaling data or re-interpreting findings for comparative analysis amongst others in the scientific community. Currently, Bioinformaticians use various libraries and open-source GUIs that are technically dissimilar in background processes and or are no longer supported.

The biggest risk is not being scientifically accurate with data processing. This has been an R&D project for that very reason. Failed research based on incorrect findings would permanently damage the reputation of any commercialised software. To tackle this, it was crucial to start with the scientific method first, before its development.

Liability; who is responsible for false analyses? The scientist or the service provider? It could be different depending on the circumstances. On the other hand, a scientific product wouldn't sell if there were a disclaimer stating: "all outputs should not be interpreted as evidence".

### **6.11.3 Economic**

Over the past several years, companies have shifted from selling one-time products to adopting a service model [24]. A great example of genomics data analysis specifically as a software service is Illumina's DRAGEN. Offering a platform of various tasks and resources.

If this project's prototype were to be brought to market, deployment as a service model would align with existing commercial success. This would motivate the providers of such a service to be result-oriented first, from the need for high reliability and accuracy. A long-term benefit not just for customers; frequent updates of a wider range of automated tasks and without downtimes for repair, but businesses too; receiving continual income and expansion of potential customer base as a result.

A subscription service has customers expect continual value. This requires investing updates and further features. A risk as these efforts can be put into upgrades that aren't demanded. Regular market research to understand current industry needs would mitigate this.

## **7 Conclusion**

### **7.1 Project Management Evaluation**

The Software Development Life Cycle model followed was the Waterfall Model. Listed are benefits and drawbacks discovered throughout the project.

#### **7.1.1 Benefits**

- Clear milestones to adhere to,
- Structure made it easy to append further tasks to,
- Tasks could be treated as separate development cycles,
- Obvious what the end goal was.

#### **7.1.2 Drawbacks**

- Requirements – if incorrect or useless, implementations may not be worthwhile,
- Research – specifying domain knowledge was essential as this model has no iteration for allowing major changes to be made at later stages,
- No stage needs to be completed before the next, i.e. no enforcement against dereliction of duty,
- Overrun – rarely development suffered from unexpected time extensions and resources for certain tasks, but was necessary to fulfil the requirements,
- Development never ends; only further implementations, testing, and maintenance.

### **7.2 Solution Evaluation**

#### **7.2.1 Strengths**

Integration of new features actively proved easier throughout development, thanks to excellent code practice. Maintenance of reusable code and documentation not only meant performing abstract procedures on one line but updates were much more approachable.

For example, implementing a second pairwise alignment algorithm Smith-Needleman only required knowledge of the raw math. Every other process was already reusable. This experience was common throughout in development 2<sup>nd</sup> semester.

### **7.2.2 Weaknesses**

Regarding cluster analysis as a research technique; an API for '.sra' file transfer was not possible, due to NCBI's FTP servers being discontinued. Instead, an Ubuntu 18.04 VM was set up for using SRA Toolkit; to download .sra files and convert to .fastq, and SimLord; to simulate sequence reads from the latter.

### **7.2.3 Learning by Failure**

Initially, cluster analysis used the normalised frequency data. Whilst outputs portrayed relations amongst the polymers accurately, this was not at all insightful. The intention was to classify genomes apart and determine their similarities. Using this data achieved neither aim.

This highlights the importance of selecting the correct data for the job. Data Science is all about experimentation, hence the name. Gained was an inspiration to research what data could be used for classifying genomes: '.fastq' files and vectorisation technique. Leading to the finalised results providing insights successfully.

### **7.2.4 Implementation**

The finalised requirements were suitably ordered in precedence of importance. Code modularly ensured completed features weren't affected by the latest bugs. Development was as concise as possible; abstract methods in 'StandardFunctions.py' meant less repetitious code throughout, making updates easier.

### **7.2.5 Development Technologies**

Most of the Python libraries implemented were learnt on the spot; forums detailing their use. It proved simple for rudimentary developments, i.e. data manipulation and plotting outputs, as well as simplifying code constructs with comprehensions and subroutines.

PyCharm provides access to hardware, e.g. increasing RAM was beneficial for both experimentation and testing. JetBrains' IDEs seem to be the best experience for Git, providing immediate live updates, and notifying you of warnings and confirming changes clearly. Git became a lot easier to deal with over time.

## **7.3 Future**

### **7.3.1 Possible Features**

#### ***7.3.1.1 Cloud Computing***

##### **7.3.1.1.1 Web Application**

A user-friendly Web Application for navigating this solution's research techniques implemented. Administered by users' organisations. Download or upload results from existing and published analysis that used this application; the system would have awareness of datasets to reference the organisation's back-end database or through the NCBI API. Lastly, 3-FA security: knowledge; password, possession; ID card, inherence; finger-print.

##### **7.3.1.1.2 Publication Referencing Engine**

Researchers need to reference datasets and results discussed in their publications. A public search engine to download output analyses files. Such files must store metadata, containing references to open-data that the Web Application could then download and process. Useful for those in the scientific community who wish to either verify research findings within moments or to conduct much similar analysis further.

#### ***7.3.1.2 Biological Syntheses***

'Syntheses.py', like all features, was developed modularly so as other biological matter could be synthesised from genomic datasets; e.g. mRNA, or tRNA. Allowing for the analyses of this kind of data not yet public to be performed there and then.

### 7.3.1.3 Exploratory Data Analyses

An EDA workflow in one run. Examples: restriction maps; visualising and labelling positions of restriction sites in DNA, phylogenetic trees; representing evolutionary relationships, or network analysis; insights into any molecular system.

### 7.3.2 Prospects

Further implementations to this software could one day achieve total automation of the Bioinformatics job role. Centralising all computational biology tasks in the application. Further offering the chance for Biologist with little to no computer science skills to branch out into this field of work.

## 7.4 Summary

A scientifically accurate and robust solution that streamlines common research tasks of a Bioinformatician was delivered. Relevant research was studied and demonstrated, all system requirements were delivered without major flaws, with a range of testing and verifiable criteria. Adversity was overcome, both in the developmental process and in amending the work plan. Therefore, the solution and project management should be considered a success.

## 8 Appendices

This dissertation does not assume the reader to have any prior knowledge in Bioinformatics and Virology. Concisely detailing only the relevant pre-request knowledge needed for understanding my solution, and not the entirety of the respective disciplines involved.

Relevant discussion of the Bioinformatics role as well as fundamental theories: coronaviridae taxonomy, SARS-CoV-2 epidemiology, and biochemistry; all of which are incorporated as computational biology in my solution.

### A – Functions

Alignment.py	
Function	sequence_lengths()
Parameters	sequence_filenames, bio_type, directory



Description	Captures lengths of all sequences
Steps	<ol style="list-style-type: none"> <li>1. Loop 'sequence_filenames'.</li> <li>2. Condition 'bio_type'.</li> <li>3. Capture lengths of sequences.</li> <li>4. Return lengths.</li> </ol>
Function	dna_alignment()
Parameters	sequence_filenames, seq_lengths, directory
Description	Ensures sequences are of equal length, for MSA. Done by appending '-'s to short sequences, no. which calculated by the longest sequence.
Steps	<ol style="list-style-type: none"> <li>1. Loop files.</li> <li>2. Condition 'bio_type'.</li> <li>3. Capture sequences.</li> <li>4. Calculate and append no. blanks '-'.</li> <li>5. Save.</li> </ol>
Function	protein_alignment()
Parameters	sequence_filenames, seq_lengths, directory
Description	Same as 'dna_alignment()' but for protein
Steps	<ol style="list-style-type: none"> <li>1. Loop files.</li> <li>2. Capture sequence.</li> <li>3. Calculate and append no. blanks '-'.</li> <li>4. Save.</li> </ol>
Function	create_file()
Parameters	bio_type
Description	Wrap-around function conditions 'bio_type' to create an .aln file of 'bio_type' sequences. Invokes: 'sequence_lengths()' & 'establish_alignment_file()'. Establishes: 'dir' & 'sequence_filenames', based on 'bio_type' sequences
Steps	<ol style="list-style-type: none"> <li>1. Condition 'bio_type'.</li> <li>2. Capture filenames.</li> <li>3. Invoke 'sequence_lengths()'.</li> <li>4. Invoke either 'dna_alignment()' or 'protein_alignment()'.</li> </ol>

<b>Clustering.py</b>	
Function	read_normalised_frequencies()
Parameters	polymer_len, *names
Description	Imports produced Normalised Frequency files (depending on: polymers & genomes of interest). Compiled as a Dataframe for Cluster Analysis. Minimum of 2 Coronaviridae names required – ‘*others’ is optional.
Steps	<ol style="list-style-type: none"> <li>1. Loop normalised frequency files, append to dataframe.</li> <li>2. Invoke clustering model methods, PCA &amp; tSNE.</li> </ol>
Function	seaborn_scatterplot()
Parameters	model, results, genome_names, polymer_len
Description	Standardised method for using Seaborn's Scatterplot to visualise Machine Learning results.
Steps	<ol style="list-style-type: none"> <li>1. Scatterplot figure.</li> <li>2. Save.</li> </ol>
Function	principal_component_analysis()
Parameters	df, genome_names, polymer_len
Description	Linear dimensionality reduction algorithm. Machine Learning algorithm for Cluster visualisation. Time in seconds, printed
Steps	<ol style="list-style-type: none"> <li>1. Instantiate.</li> <li>2. ‘fit_transform()’.</li> <li>3. Elapsed time.</li> <li>4. Invoke ‘seaborn_scatterplot()’.</li> </ol>
Function	tSNE()
Parameters	df, genome_names, polymer_len
Description	t-distributed Stochastic Neighbour Embedding. Machine Learning algorithm for Cluster visualisation. Time in seconds, printed
Steps	<ol style="list-style-type: none"> <li>1. Instantiate.</li> <li>2. ‘fit_transform()’.</li> </ol>

	3. Elapsed time. 4. Invoke 'seaborn_scatterplot()'.
Function	run()
Parameters	n_reads, *names
Description	Imports reads from genomes of interest.
Steps	1. Loop simulated reads, append to dataframe. 2. Invoke clustering model methods, PCA & tSNE.

dataimport.py	
Function	api()
Parameters	db, ids
Description	Established connection to NCBI servers
Steps	1. Creates Entrez identifier, 2. Loop each id: connects and downloads dataset, 3. Cleanses data, 4. Stores.
Function	read_genomes()
Parameters	db, ids
Description	Imports Coronaviridae datasets from src
Steps	1. Invokes api(). 2. Read filenames from 'src/'. 3. Loop each file: open file, capture 'name', 'description', 'sequence', and append to dict 'coronaviridae'. 4. Print no. coronaviridae.

MolecularDocking.ipynb	
Function	protein_structure()
Parameters	display, pdb, color
Description	Constructs 3D protein structure.
Steps	1. Construct model from data.

	2. Set style.
Function	ligand_structure()
Parameters	display, pdb
Description	Constructs 3D molecule that best binds to target protein.
Steps	<ol style="list-style-type: none"> <li>1. Construct model from data.</li> <li>2. Set style.</li> </ol>
Function	visualisation()
Parameters	N/A
Description	Plots 3D protein structure and configs interactivity.
Steps	<ol style="list-style-type: none"> <li>1. Instantiate py3dmol.</li> <li>2. Invoke 'protein_structure()'. Configure interactivity.</li> </ol>
Function	docking()
Parameters	protein, ligand
Description	Plot docked ligand and respective protein structure and configs interactivity.
Steps	<ol style="list-style-type: none"> <li>1. Instantiate py3dmol.</li> <li>2. Invoke 'protein_structure()'.</li> <li>3. Invoke 'ligands_structure()'.</li> <li>4. Configure interactivity.</li> </ol>

<b>MultipleSequenceAlignment.py/.ipynb</b>	
Function	map_colors()
Parameters	sequences, bio_type
Description	Sets a distinct colour for 'bio_type"s each unique chars in 'sequences'
Steps	<ol style="list-style-type: none"> <li>1. Capture all characters from sequences.</li> <li>2. Determine no. colours to generate.</li> <li>3. Assign unique colour to unique character.</li> </ol>
Function	visualisation()
Parameters	alignment, bio_type

Description	Multiple Sequence Alignment Viewer. Global Viewer - overall of coloured sequences' characters. Aligned Sequences Viewer - interactive view for analysis, coloured coded characters & ability to scroll.
Steps	<ol style="list-style-type: none"> <li>1. Capture sequence and characters.</li> <li>2. Invoke 'map_colors()'</li> </ol>
Function	run()
Parameters	bio_type
Description	Wrap-around function conditions 'bio_type' to create an .aln file of 'bio_type' sequences. Establishes: 'filename'
Steps	<ol style="list-style-type: none"> <li>1. Condition 'bio_type'.</li> <li>2. Invoke MSA.</li> <li>3. Plot.</li> </ol>

Nucleotides.py	
Function	base_combinations()
Parameters	polymer_len
Description	Accumulator method for invoking recursive method (memory efficiency).
Steps	<ol style="list-style-type: none"> <li>1. Check 'polymer_length' is an integer &gt; 1.</li> <li>2. Invoke 'base_combinations_recursive()'.</li> <li>3. Stores.</li> </ol>
Function	base_combinations_recursive()
Parameters	polymer, polymer_len, combinations
Description	Generates all possible polymers possible, each of a 'polymer_len'. Each recursion, we append a complete 'polymer' to 'combinations'. We pass 'combinations' back to 'base_combinations()' and drop the Stack. Stack does not need to be unravelled, once threshold is met, per iteration.
Steps	<ol style="list-style-type: none"> <li>1. Check each element is 3 characters, start next.</li> <li>2. Loop in alphabetical order, establish new and drop intermediate stack frame.</li> <li>3. Append start of next 'new_polymer' and replace until completed.</li> </ol>

	4. Link and return stack directly to calling frame.
Function	base_content()
Parameters	Name
Description	Calculates the composition of each polymer, from 'base_combination', in a given genome as a %.
Steps	<ol style="list-style-type: none"> <li>1. Print genome.</li> <li>2. Calculate a genomes compositions for each polynucleotide.</li> <li>3. Invoke 'base_content_plot()'.</li> </ol>
Function	base_content_plot()
Parameters	name, sequence
Description	Nitrogenous Bases - AT/GC Ratio: Guanine & Cytosine as % of DNA (always paired together), Adenine & Thymine as % of DNA (always paired together).
Steps	<ol style="list-style-type: none"> <li>1. Calculate GC and AT content of sequence.</li> <li>2. Plot as bar chart.</li> <li>3. Save.</li> </ol>
Function	composition_comparison()
Parameters	polymer_len, *names
Description	Plots a chart - Normalised polynucleotide frequencies of bases composition, for n genomes of interest.
Steps	<ol style="list-style-type: none"> <li>1. Title is plural or singular noun.</li> <li>2. Plot <i>x, y</i> axes.</li> <li>3. Loop over genomes and assign a 'COLOUR_MAP', invoking 'normalised_frequencies()'</li> <li>4. Plot.</li> <li>5. Save.</li> </ol>
Function	normalised_frequencies()
Parameters	polynucleotides, name, sequence

Description	Calculates no. appearances each polymer, as a subsequence, appears in a given genome, as a % . Stores csv - headers = polymers, nf = content (filename "nf_[polymers]_[coronavirus].csv").
Steps	<ol style="list-style-type: none"> <li>1. Loop each sequence, calculate 'nf' for each 'base_combination'.</li> <li>2. Save.</li> <li>3. Return 'normalised_freq'.</li> </ol>

<b>PairwiseSequencing.py</b>	
Function	save()
Parameters	bio_type, output_name, sequencing
Description	Stores matches and their scores w/ appropriate file naming convention.
Steps	<ol style="list-style-type: none"> <li>1. Establish file and Save.</li> </ol>
Function	gap_penalty()
Parameters	x, y
Description	Deducts points for gaps when matching up sequencing, using a logarithmic scale.
Steps	<ol style="list-style-type: none"> <li>1. Condition new gap or consecutive.</li> <li>2. Return respective penalty.</li> </ol>
Function	sequence_identity()
Parameters	align1, align2
Description	Calculates matches btwn a pair of aligned sequences, as %
Steps	<ol style="list-style-type: none"> <li>1. Loop, list of: does n<sup>th</sup> character for each aligned seq match? (T/F).</li> <li>2. Calculate Sequence Identity, using list.</li> <li>3. Loop, list of: does n<sup>th</sup> character for each aligned seq match w/out gaps '-'? (T/F).</li> <li>4. Calculate Gapless Sequence Identity.</li> </ol>
Function	run()
Parameters	bio_type, *names

Description	Performs Pairwise Sequencing task. Matches sub-sequences of similarity that may indicate evolutionary, functional, and structural relationship. Higher the score, higher the relationship.
Steps	<ol style="list-style-type: none"> <li>1. Condition 'bio_type' for prefix and directory.</li> <li>2. Capture filenames.</li> <li>3. Loop filenames of 'bio_type' to capture genome data.</li> <li>4. Declare target and query sequences.</li> <li>5. Perform Pairwise Sequencing.</li> <li>6. Save outputs.</li> </ol>

StandardFunctions.py	
Function	capture_filenames()
Parameters	bio_type
Description	Captures all file names in directory, based on 'bio_type'.
Steps	<ol style="list-style-type: none"> <li>1. Condition 'bio_type'.</li> <li>2. Loop, capture filenames in directory.</li> </ol>
Function	output_name()
Parameters	Genomes
Description	Concatenates coronavirus names for an output filename.
Steps	<ol style="list-style-type: none"> <li>1. Loop genomes, gather names to construct appropriate name in figure.</li> </ol>
Function	output_filename()
Parameters	sequence_filename
Description	Concatenates coronavirus names for Alignment file setup.
Steps	<ol style="list-style-type: none"> <li>1. Condition 'bio_type'</li> <li>2. Remove prefix and or file extensions.</li> </ol>
Function	remove_firstline()
Parameters	directory_filename
Description	e.g. DNA datasets have a description line on top, followed by a line break.
Steps	<ol style="list-style-type: none"> <li>1. Open file.</li> </ol>



	2. Remove top line.
Function	remove_prefix()
Parameters	text, prefix
Description	Removes prefix of filenames in data/Syntheses (eg. "protein_<VIRUS>" -> "<VIRUS>").
Steps	1. Conditions prefix is in string for removal.
Function	polynucleotides()
Parameters	polymer_len
Description	Returns list of polynucleotides of interest
Steps	1. Open file contents, strip, and return as list.

<b>Syntheses.py</b>	
Function	write_file()
Parameters	coronavirus_name, protein
Description	Stores synthesised protein w/ appropriate file naming convention.
Steps	1. Save.
Function	protein()
Parameters	Coronavirus
Description	Converts DNA sequences into Protein for further tasks.
Steps	<ol style="list-style-type: none"> <li>1. Condition at least tri-nucleotide.</li> <li>2. Condition non-dna alphabet chars. Prompt user – cancel or skip char.</li> <li>3. Declare dict 'dna_codons', holding all translations.</li> <li>4. Loop over 3 chars, appending 'get()' to 'protein'.</li> <li>5. Invoke 'write_file()'.</li> </ol>
Function	alphabet_check()
Parameters	name, sequence, alphabet
Description	Compares all characters/ elements of 'sequence' against its bio_type 'alphabet'.

Steps	<ol style="list-style-type: none"> <li>1. Check sequence for non-‘alphabet’ character.</li> <li>2. Prompt User to continue or cancel synthesis.</li> </ol>
-------	--

## B – Genome Datasets

Downloaded from the [National Center for Biotechnology Information \(NCBI\)](https://www.ncbi.nlm.nih.gov/), each have an “Accession”, a unique ID. This project experiments with those of the Coronaviridae taxon.

Accession	Definition
JQ316196	SARS coronavirus HKU-39849 isolate UOB, complete genome.
MT387202	Middle East respiratory syndrome-related coronavirus isolate.
MT873892	Severe acute respiratory syndrome coronavirus 2 isolate.
AP017922	Staphylococcus aureus DNA, complete genome.

## C – Data Model

Listed is key data used in calculations, analysis, and is stored for future implementations.

Tables of developed functions can be found in Appendices, section [8.1](#).

Coronavirus		
Coronaviridae	dict	Store the name, description, and sequence of viruses as lists.
Polymers	list	Strings of all combinations of bases of n lengths for normalised freq.
Normalised Freq	list	Values calced by occurrences of polymers, proportional to total, 0-1.

Cluster Analysis		
SimLoRD	.txt	Vectorised simulated reads data from genome datasets (32 cols).
PCA Results	ndarray	Dimensionality reduced normalised freqs., fit and transformed.
t-SNE Results	ndarray	Normalised freq. data fit and transformed.

Pairwise Sequencing		
Protein	list	Assessing functional relationships of genomes.
Sequencing	list	All matches between 2 sequences.

Multiple Sequence Alignment		
Alignment	aln	Holds sequences, of a type, of the same length for visualisation.

Molecular Docking		
Protein Structure	pdb	3D model for plot proteins for interactive plotting.
Ligands	pdb	Visualising protein-ligand interactions.

## D – Code

### D.1 – Code Style

Pythonic conventions, consistent with the official style [guide](#), include but not limited to:

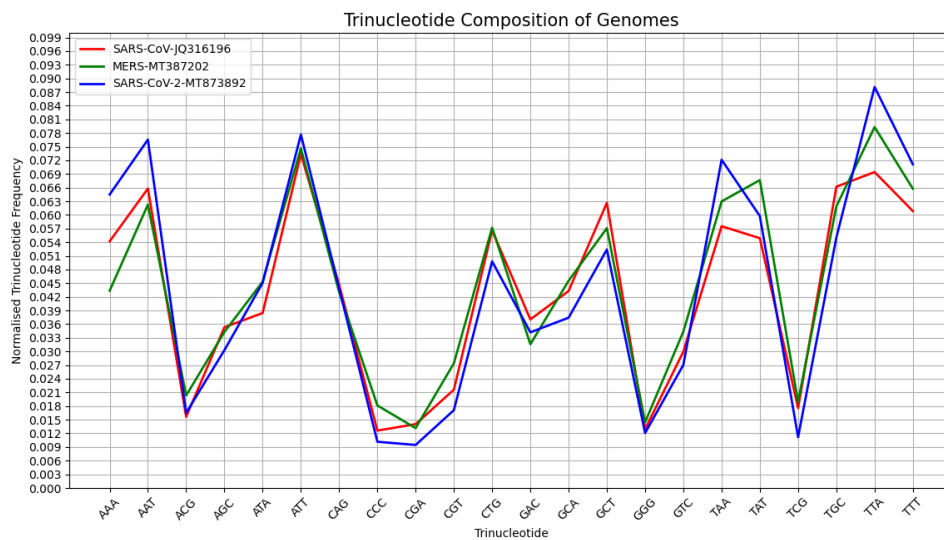
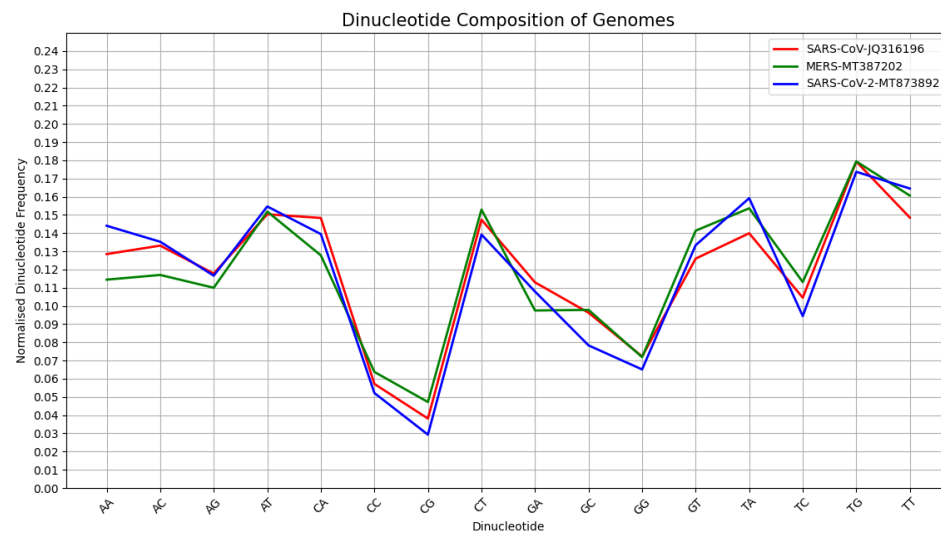
- Imports placed at top of files, same line if from the same parent library,
- Constants, global variables not altered, declared as all upper-case,
- Functions and variable are lower-case, underscores, `def foo_bar(variable_name)`,
- Variables named identically, if the same in both the data they store and contextually,
- Indentation, whitespace, and maximum line length enforced by `Ctrl + Alt + L`,
- String quotes: `" "` – human input/ speech, and `' '` – a default or static parameter option,
- Comments:
  - Inline comments are adjacent to the subject line of code; 2 space gap (`#`),
  - Docstrings hold complete sentences, max 72 characters (`""" """`),
- BIDMAS – dealing with mathematical lines.

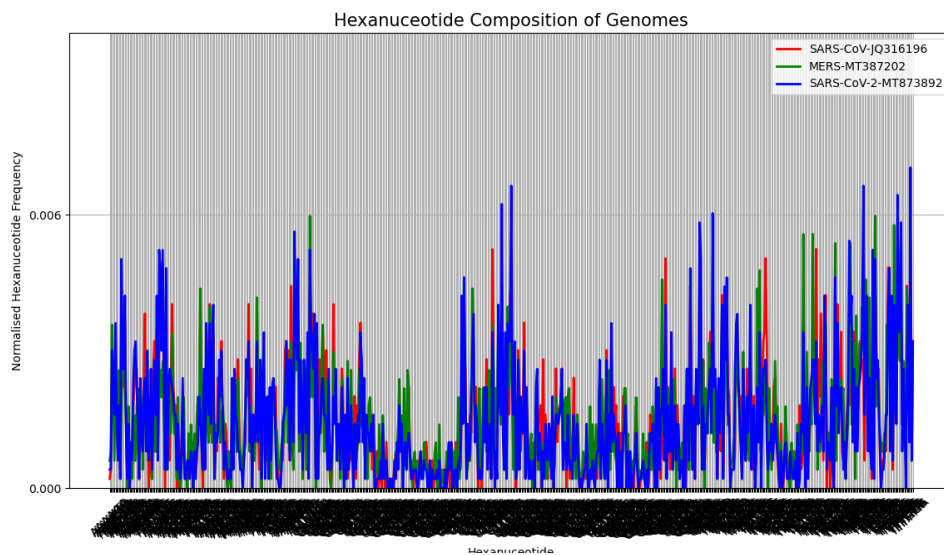
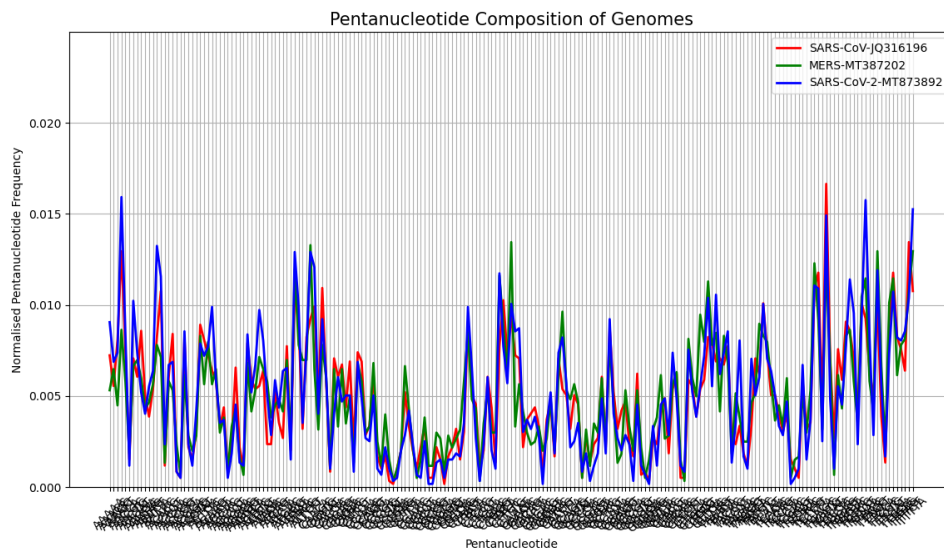
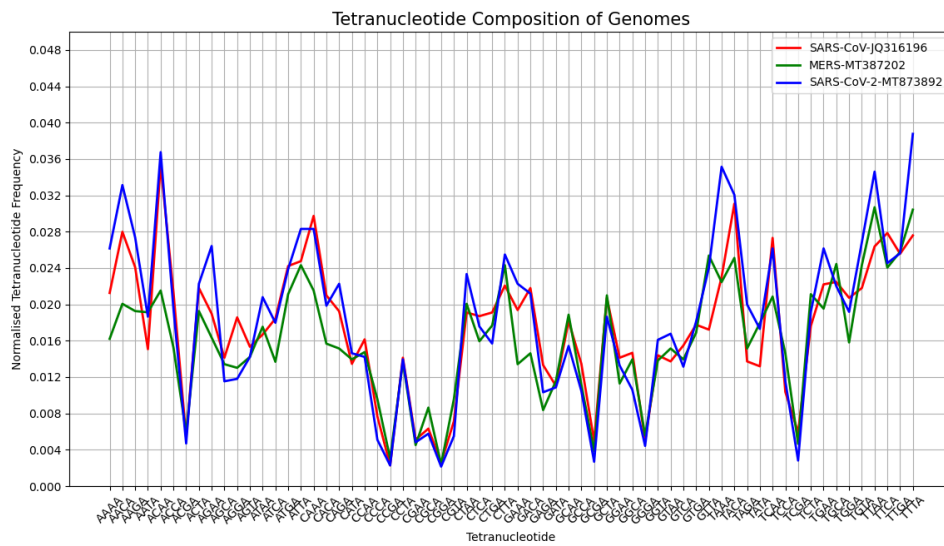
### D.2 – List Comprehensions

Technique of assigning elements to a list works by allocating the list variable to memory first before adding each element for faster performance, so `‘.append()’` isn’t needed. For example, if a for loop is applied to a list comprehension the list doesn’t have to resize at each iteration. Complementing two general solution aims: optimised performance and code style.

## E – Experimental Findings

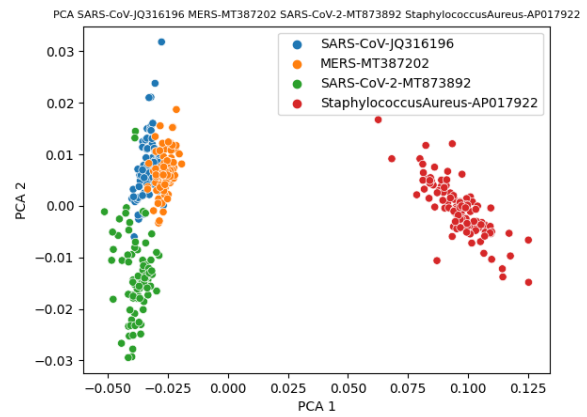
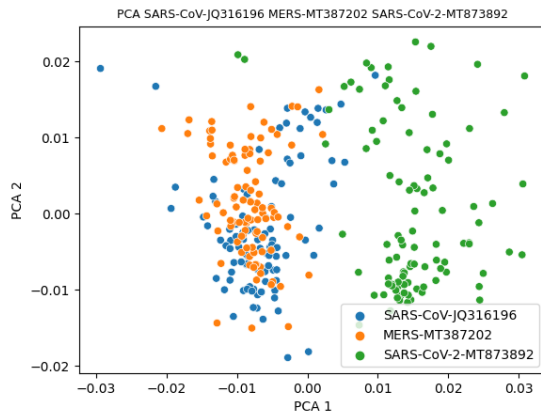
### E.1 – Normalised Frequencies



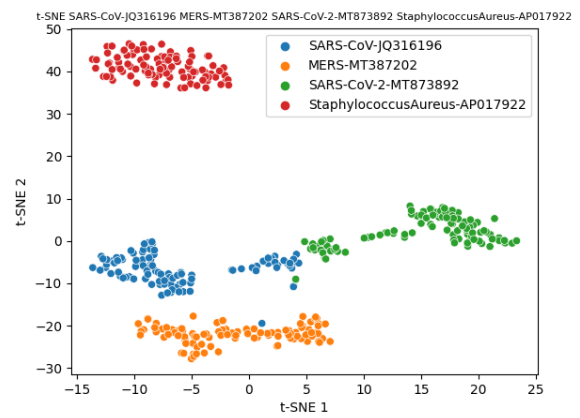
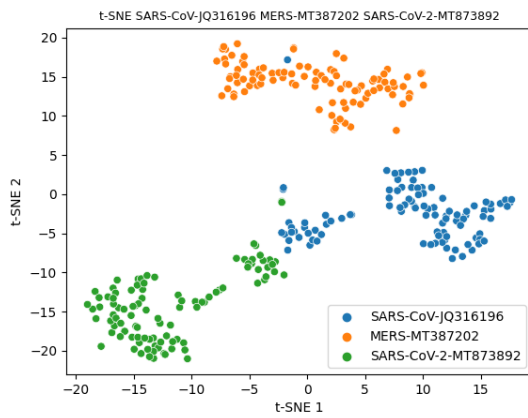


## E.2 – Cluster Analysis

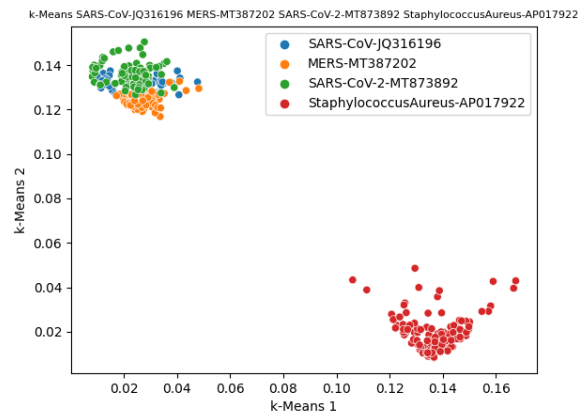
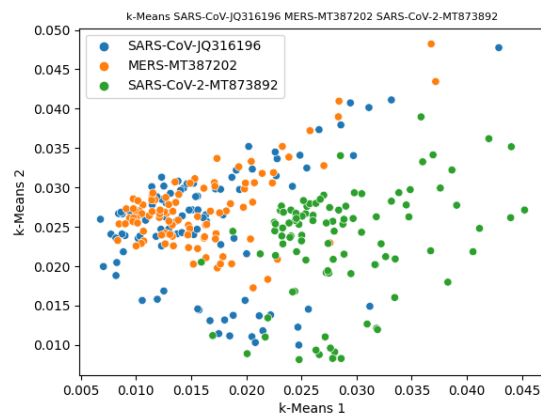
### E.2.1 – PCA

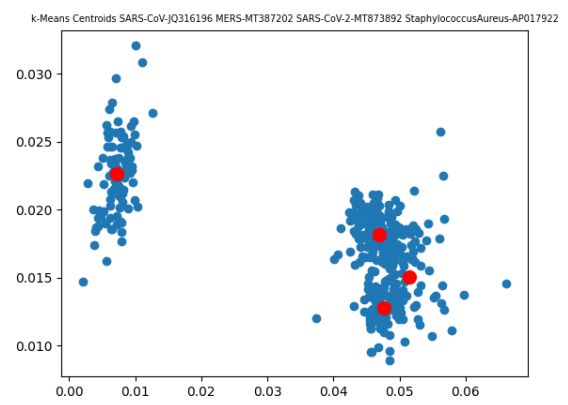
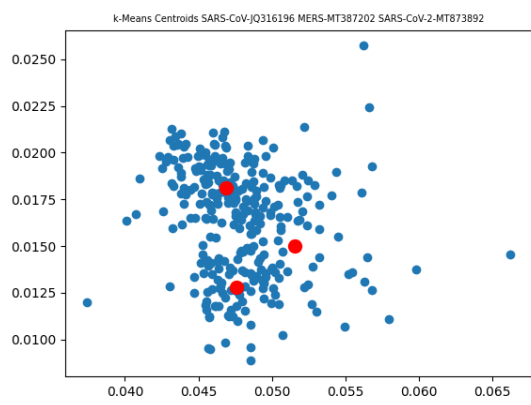
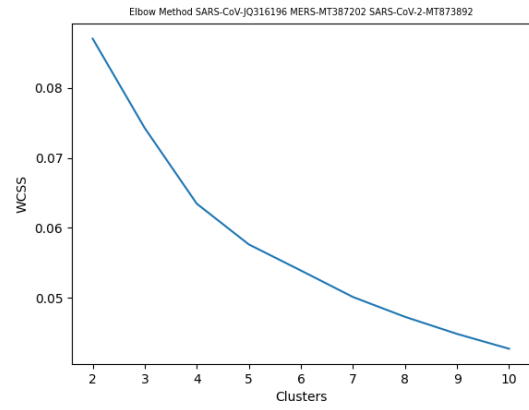
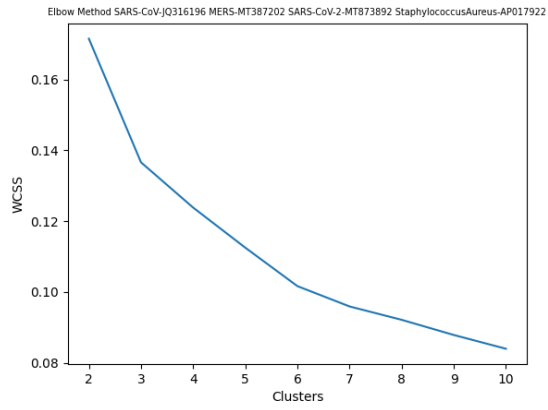


### E.2.2 – t-SNE

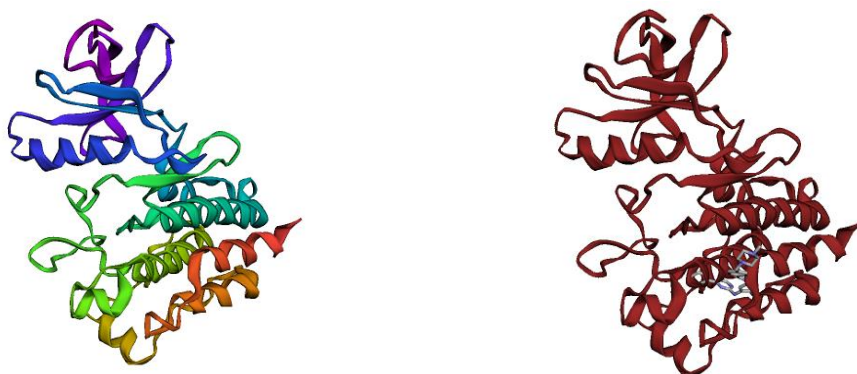


### E.2.3 – k-Means





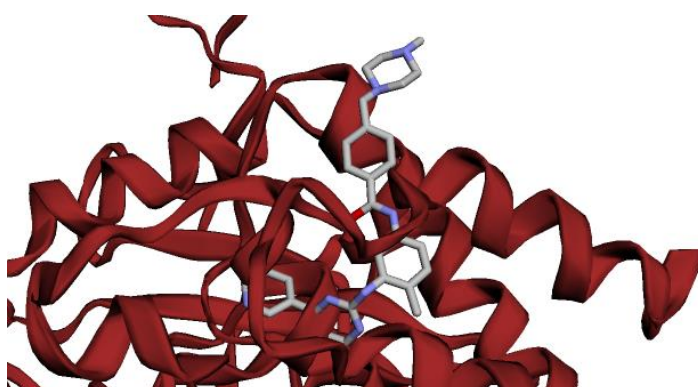
## E.5 – Macromolecular Docking



```
Detected 2 CPUs
Reading input ... done.
Setting up the scoring function ... done.
Analyzing the binding site ... done.
Using random seed: -1936117844
Performing search ...
0%  10  20  30  40  50  60  70  80  90 100%
|----|----|----|----|----|----|----|----|----|----|
*****
done.
Refining results ... done.
```

mode	affinity (kcal/mol)	dist from best mode rmsd l.b.   rmsd u.b.
1	-12.3	0.000   0.000
2	-10.6	2.605   12.374
3	-10.4	3.208   13.864
4	-10.4	3.269   12.448
5	-10.2	1.790   13.820
6	-9.9	3.097   12.059
7	-9.4	1.910   13.957

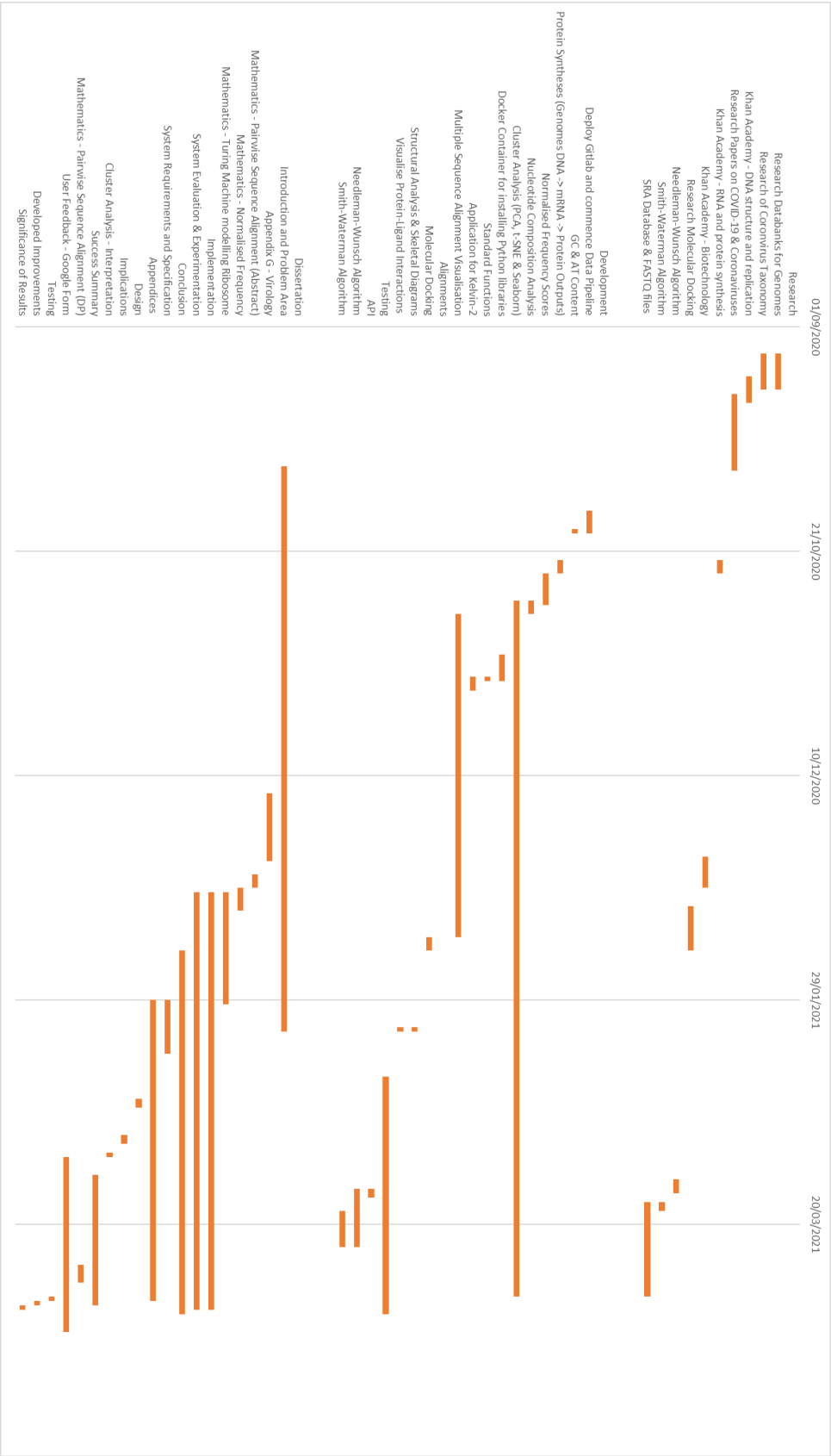
```
Writing output ... done.
```





F – Conclusion

F.1 – Gantt Chart



## F.2 – Learning by Failure

Original results from performing cluster analysis of the calculated normalised frequency data.

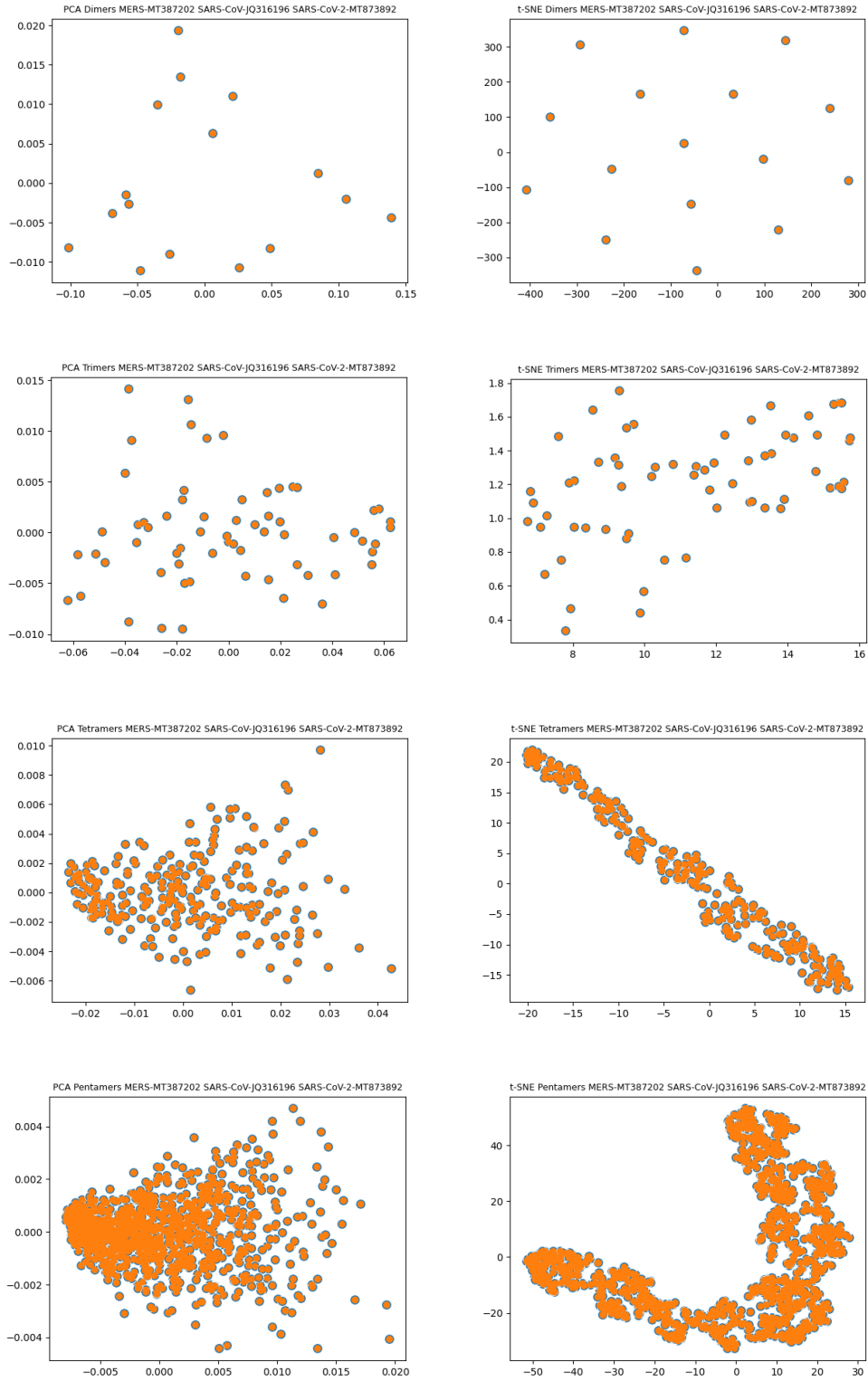


Figure 22: PCA & t-SNE results in  $n$  data points, representing polymers' scores shared across the three genomes.

PCA was capable of providing insight with less normalised polymer frequency scores than t-SNE. t-SNE requires more instances to provide substantial insight than PCA. Apparent between the models' clustering of dimers' normalised frequency scores; where in t-SNE's plot the data points are near-equally spread, indicating a lack of correlation.

Its form begins to take shape when clustering trimers, of equal measure to PCA's. Thereafter, accelerating in a distinct one-structured pattern. Notably; t-SNE's pentamers' plot, which can be described as one long stretch that bends, is a fairly common behaviour. Indicating a stark trend amongst data points; a sign of a high performing model.

## G – Virology

### G.1 – Coronavirus Taxonomy

SARS-CoV-2 resides with-in the Coronaviridae family. The Coronaviruses' classification has the Coronaviridae family listed as having a total of 39 species; across 27 subgenera, 5 genera, and 2 subfamilies [14]. Set by the International Committee on Taxonomy of Viruses' (ICTV) Coronaviridae Study Group (CSG), who are responsible for determining the relation of novel viruses to known viruses of their genus [14].










Category	Coronaviruses	Humans	Divergence
Realm	<i>Riboviria</i>		
Order	<i>Nidovirales</i>	Primates	
Suborder	<i>Cornidovirineae</i>		
Family	<i>Coronaviridae</i>	Hominidae	
Subfamily	<i>Orthocoronavirinae</i>	Homininae	
Genus	<i>Betacoronavirus</i>	<i>Homo</i>	
Subgenus	<i>Sarbecovirus</i>		
Species	<i>Severe acute respiratory syndrome-related coronavirus</i>	<i>Homo sapiens</i>	
Individuum	SARS-CoVUrbani, SARS-CoVGZ-02, Bat SARS CoVRf1/2004, Civet SARS CoVSZ3/2003, SARS-CoVPC4-227, SARSr-CoVBtKY72, SARS-CoV-2 Wuhan-Hu-1, SARSr-CoVRatG13, and so on.	Dmitri Ivanovsky, Martinus Beijerinck, Friedrich Loeffler, Barbara McClintock, Marie Curie, Albert Einstein, Rosalind Franklin, Hideki Yukawa, and so on.	

Figure 23: Taxonomy of SARS-CoV-2 Individuum [14].

## **G.2 – Epidemiology**

### ***G.2.1 – SARS-CoV-2***

SARS-CoV-2 is a virus of interest that causes the coronavirus disease of 2019, a.k.a. COVID-19 [3]. Originating from the Huanan Seafood Market in Wuhan, China in December 2019 [3]. The World Health Organization (WHO) announced a pandemic on 11<sup>th</sup> March 2020, meaning the virus spread internationally [3].

COVID-19 as a high mortality, over 1,000,000 deaths worldwide, but with varying morbidity, that is the severity of symptoms case-by-case [3].

### ***G.2.2 – VOC 202012/01 Variant***

As of accessed, a new strain of SARS-CoV-2 has emerged in the United Kingdom known as ‘Variant of Concern 202012/01’ or ‘B.1.1.7’ [4]. Having acquired 17 mutations [4], all of which are associated with its DNA; an alarmingly high number of changes in the SARS-CoV-2 viral genome.

Detected after analyses, in October 2020, of a sample taken in September [4]. So far, it has been responsible for nearly 2/3<sup>rds</sup> of cases in mid-December [4]. This variant’s transmissibility is inconclusive as numerous ongoing studies are yet to be published.

James Gallagher, BBC’s Health and Science correspondent, states three points that necessitate vaccine research for this new strain [4]:

1. It’s continually evolving; rapidly replacing weaker strains of the novel virus,
2. Mutations affecting part of the virus most likely countering current vaccines (circa December 2020),
3. Certain mutations have increased the virus’s ability to cause infection; that is binding to human host cells.

### ***G.2.3 – Cluster 5 Variant***

Discovered in Denmark, early November 2020, farmed minks had spread the strain to citizens [15]. Soon after, minks were culled in the hopes of reducing the risk of transmission [15].

On-going research in Denmark evidences a number of mutations not previously observed in coronaviruses; with apprehension that this variant may result in reduced virus neutralisation in humans, i.e. indicating a decreased timeframe that the immune system and or a vaccine has to respond and level of efficacy against the infection [16]. Danish authorities have identified 12 human-host cases of the Cluster 5 variant, circa 6<sup>th</sup> November 2020 [16].

#### ***G.2.4 – 501.V2 Variant***

Further, during December 2020; another variant of SARS-CoV-2 named ‘501.V2’ emerged from South Africa. Initially observed in October with over 300 confirmed cases in the country since then [5], it’s believed to have been circulating since August 2020 [5].

501.V2 variant’s cause for concern is transmissibility. Preliminary studies suggest it is of a higher viral load, increasing chances of transmission and replacing other lineages circulating South Africa [16].

ECDC (2020) evidencing “Two cases of 501.V2 have been detected in the United Kingdom in contacts of a person who had travelled to South Africa” (p. 14) [5]; with reports of it having already spread to Finland, Switzerland, Australia and Japan [5].

### **G.3 – Biochemistry**

#### ***G.3.1 – DNA***

Deoxyribonucleic acid (DNA) polymer, the form in this case being ‘Double Stranded DNA’ (ds-DNA), holds genetic information needed for inheritance [2]. Nucleotides, or bases, compose of  $n$  carbon compounds Adenine, Cytosine, Guanine, Thymine – denoted as  $[A, C, G, T]$  [2]. DNA encodes the amino acid sequences of proteins [10].

Amino Acid	Abbreviation	Code
Alanine	Ala	A
Arginine	Arg	R
Asparagine	Asn	N
Aspartic acid	Asp	D
Cysteine	Cys	C

Glutamine	Gln	Q
Glutamic acid	Glu	E
Glycine	Gly	G
Histidine	His	H
Isoleucine	Ile	I
Leucine	Leu	L
Lysine	Lys	K
Methionine	Met	M
Phenylalanine	Phe	F
Proline	Pro	P
Serine	Ser	S
Threonine	Thr	T
Tryptophan	Trp	W
Tyrosine	Tyr	Y
Valine	Val	V
Termination Codon	TERM	*

Table 1: Amino acids and their codes used in synthesised protein datasets [20].

### G.3.2 – RNA

Ribonucleic acid (RNA), the form in this case being ‘Messenger RNA’ (mRNA), is a single-stranded polymer that consists of long nucleotide chains. RNA is transcribed from DNA, an intermediary for carrying out the instructions for translation of amino acids into protein [2].

There are differences: RNA is mostly single-stranded; DNA is double-stranded, RNA contains sugar ribose; DNA contains deoxyribose, RNA holds uracil; DNA holds thymine [2].

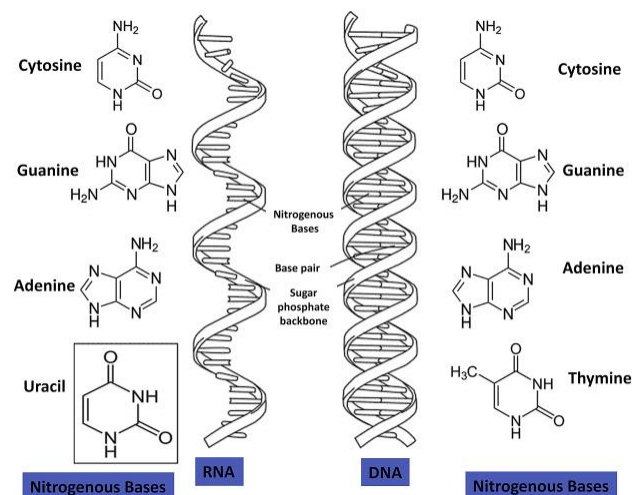


Figure 28: Structure of DNA and RNA (Fig. 8.1.7) [11].

### G.3.3 – Ribosome

A ribosome is a molecular machine that performs protein synthesis, by reading RNA templates, transcribed from DNA, to chemically bond amino acids into proteins [11]. Mention of quasi-mechanical movements are the outputs based on chemical inputs [26]. It can be interpreted synonymously to a computer compiler. DNA is the high-level software language, the ribosome is the compiler, RNA is the machine code, and proteins are the runtime methods.

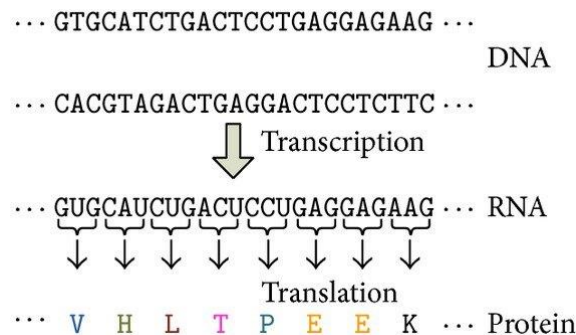


Figure 24: Sequence outputs of Protein Synthesis steps Transcription and Translation (p. 2) [10].

### G.3.4 – Protein

Polypeptides, i.e. protein sequences, are the linear chains of organic compounds constructed of amino acids that are folded into a globular protein [10]. Globular proteins are three-dimensional structures (certain figures demonstrate this). Subsets of “amino acids in a polymer chain are joined together by the peptide bonds between the carboxyl and amino groups of adjacent amino acid residues” (p. 1) [10].

It's these globular proteins that are derived from DNA. Typically, a protein e.g. Tyrosine, initiates one specific action. Protein is the literal biological functioning of viruses performed at the molecular level. If we plan on disabling a virus's ability to cause infection, not only must we then assess what protein structures are involved with infection but too the very differences in DNA that encodes them.

### G.3.5 – Protein Synthesis

Turing Machine represents this process of protein synthesis, performed by the ribosome.

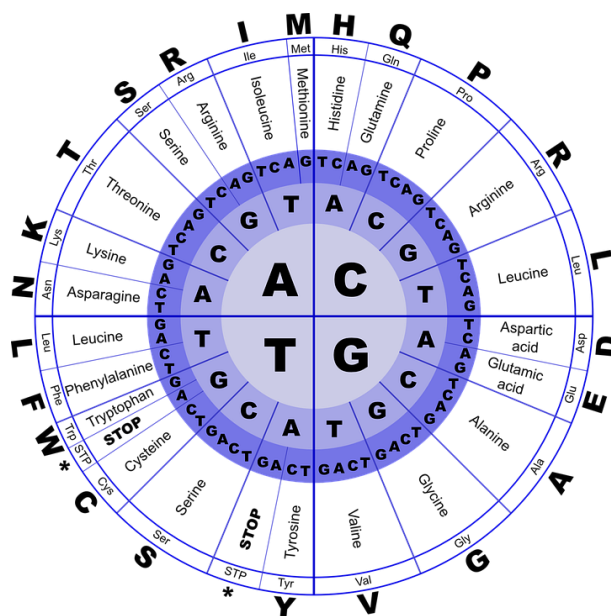


Figure 25: Standard Genetic Code wheel represents the translation of tri-nucleotides into proteins and terminations [19].

### G.3.6 – Mutations

Mutations are alterations in the nucleotide sequence of a genome [12]. Occurring during DNA replication; when the ribosome copies the genetic information. They are either benign and are no cause of effect, or cascade and affect the way genes are expressed and thus how proteins are synthesised [12].

#### G.3.6.1 – N501Y Spike Protein Mutation

N501Y is an amino acid modification present in the aforementioned variants [5]. This can be read as amino acid ‘Asparagine’ (N) has mutated to amino acid ‘Tyrosine’ (Y), at character position ‘501’ in the protein sequence, one-indexed. Altering the most crucial part of the spike protein that causes infection, in the Receptor-Binding Domain (RBD) [4]. An example area of interest – where protein binds to a specific atom or molecule, i.e. DNA of the host cell [7].

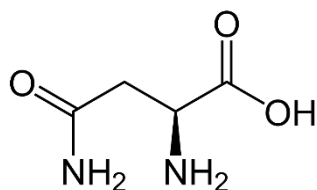


Figure 26: Skeletal formula of Asparagine [8].

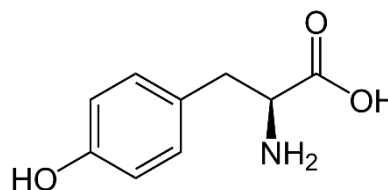


Figure 27: Skeletal formula of Tyrosine [9].



Whilst VOC 202012/01 and 501.V2 variants have this spike protein mutation, amongst other similarly evolved characteristics, it is not believed that they evolved from either or [7]. As Dr. Filip Fratev (2020) states in his preliminary paper – not yet peer reviewed, “An urgent understanding of N501Y mechanism of action at molecular level is highly required” (p. 1) [6].

## H – Other Terminology

### H.1 – Sequence Analysis

The study of identifying areas of interest in two or more DNA or protein sequences that denote functional, structural and or evolutionary relationships [13]. The differences assessed are substitution, insertion, and deletion; all forms of mutation as previously discussed.

### H.2 – Metagenomics

Metagenomics is the study of sampling DNA fragments from a case-study environment, to analyse all microorganisms present; pre-existing or novel, without obtaining pure cultures. It is common practice for researchers to artificially simulate reads, from complete genome datasets, with quality scores; a probabilistic measurement of a natural isolated occurrence. Short reads are ~100s of nitrogenous base pairs, long reads range +10,000bp [27].

## 9 References

- [1] D. Bell, CSC3002, EECS, 2021. <https://gitlab2.eecs.qub.ac.uk/401981941/csc3002>
- [2] T. A. Orlova, *Mathematical Models, Algorithms, and Statistics of Sequence Alignment*, University of South Carolina, 2010. [E-book] Available: [https://people.math.sc.edu/czabarka/Theses/thesis\\_orlova.pdf](https://people.math.sc.edu/czabarka/Theses/thesis_orlova.pdf) [Accessed: 20 October 2020]
- [3] P. Chellapandi, S. Saranya, *Genomics insights of SARS-CoV-2 (COVID-19) into target-based drug discovery*, Medicinal Chemistry Research, 2020. [E-book] Available: <https://link.springer.com/article/10.1007/s00044-020-02610-8> [Accessed: 22 October 2020]

- [4] J. Gallagher, *New coronavirus variant: What do we know?* BBC News, 2020. [Article]  
Available: <https://www.bbc.co.uk/news/health-55388846> [Accessed: 27 December 2020]
- [5] European Centre for Disease Prevention and Control, *Risk related to spread of new SARS-CoV-2 variants of concern in the EU/EEA*, ECDC: Stockholm, 2020. [E-book] Available: <https://www.ecdc.europa.eu/en/publications-data/covid-19-risk-assessment-spread-new-sars-cov-2-variants-eueea> [Accessed: 30 December 2020]
- [6] F. Fratev, *The SARS-CoV-2 S1 spike protein mutation N501Y alters the protein interactions with both hACE2 and human derived antibody: A Free energy of perturbation study*, bioRxiv, 2020. [E-book]  
<https://www.biorxiv.org/content/10.1101/2020.12.23.424283v1> [Accessed: 30 December 2020]
- [7] D. C. Phillips, *The Three-dimensional Structure of an Enzyme Molecule*, Scientific American, a division of Nature America, Inc., 1966. [E-book]  
[https://www.jstor.org/stable/24931327?seq=1#metadata\\_info\\_tab\\_contents](https://www.jstor.org/stable/24931327?seq=1#metadata_info_tab_contents) [Accessed: 30 December 2020]
- [8] NEUROtiker, *L-Asparagin - L-Asparagine.svg*, Wikipedia Commons, 2007. [Online Image] [https://commons.wikimedia.org/wiki/File:L-Asparagin\\_-\\_L-Asparagine.svg](https://commons.wikimedia.org/wiki/File:L-Asparagin_-_L-Asparagine.svg) [Accessed: 30 December 2020]
- [9] NEUROtiker, *L-Tyrosin - L-Tyrosine.svg*, Wikipedia Commons, 2007. [Online Image] [https://commons.wikimedia.org/wiki/File:L-Tyrosin\\_-\\_L-Tyrosine.svg](https://commons.wikimedia.org/wiki/File:L-Tyrosin_-_L-Tyrosine.svg) [Accessed: 30 December 2020]
- [10] J. Cao and L. Xiong, *Protein Sequence Classification with Improved Extreme Learning Machine Algorithms*, Hindawi, 2014. [E-Book]  
<https://www.hindawi.com/journals/bmri/2014/103054/> [Accessed: 31 December 2020]
- [11] B. D. Malhotra and Md. A. Ali, *Nanomaterials for Biosensors: Fundamentals and Applications*, William Andrew, 2018. [E-Book] <https://doi.org/10.1016/B978-0-323-44923-6.00001-7> [Accessed: 31 December 2020]
- [12] *RNA and protein synthesis review*, Khan Academy, 2017. [MOOC]  
<https://www.khanacademy.org/science/high-school-biology/hs-molecular-genetics/hs-rna-and-protein-synthesis/a/hs-rna-and-protein-synthesis-review> [Accessed: 31 December 2020]
- [13] R. Durbin, S. Eddy, A. Krogh and G. Mitchison, *Biological Sequence Analysis. Probabilistic Models of Proteins and Nucleic Acids*, Wiley, 1999. [E-Book]

- [https://onlinelibrary.wiley.com/doi/10.1002/\(SICI\)1099-0844\(199903\)17:1%3C73::AID-CBF799%3E3.0.CO;2-8](https://onlinelibrary.wiley.com/doi/10.1002/(SICI)1099-0844(199903)17:1%3C73::AID-CBF799%3E3.0.CO;2-8) [Accessed: 1 January 2021]
- [14] A. E. Gorbalenya et al, *The species Severe acute respiratory syndrome-related coronavirus: classifying 2019-nCoV and naming it SARS-CoV-2*, Nature Microbiology, 2020. [E-Book] <https://doi.org/10.1038/s41564-020-0695-z> [Accessed: 3 January 2021]
- [15] SARS-CoV-2 mink-associated variant strain – Denmark, The World Health Organisation (WHO), 2020. [Online] <https://www.who.int/csr/don/06-november-2020-mink-associated-sars-cov2-denmark/en/> [Accessed: 3 January 2021]
- [16] SARS-CoV-2 Variants, The World Health Organisation (WHO), 2020. [Online] <https://www.who.int/csr/don/31-december-2020-sars-cov2-variants/en/> [Accessed: 3 January 2021]
- [17] C. Rye et al, *The Genetic Code*, OpenStax, 2016. [Online] <https://openstax.org/books/biology/pages/15-1-the-genetic-code> [Accessed: 4 January 2021]
- [18] J. C. Wooley and H. S. Lin, *Catalyzing Inquiry at the Interface of Computing and Biology*, The National Academies Press, 2005. [E-Book] <https://doi.org/10.17226/11480> [Accessed: 26 January 2021]
- [19] *Standard Genetic Code*, Gene Infinity, 2021. [Online Image] [http://www.geneinfinity.org/sp/sp\\_gencode.html](http://www.geneinfinity.org/sp/sp_gencode.html) [Accessed: 27 January 2021]
- [20] , *Peptide Nomenclature Guide*, Tocris Bioscience. [Online] <https://www.tocris.com/resources/peptide-nomenclature-guide> [Accessed: 4 February 2021]
- [21] S. C. J. Parker et al, *The relationship between fine scale DNA structure, GC content, and functional elements in 1% of the human genome*, World Scientific, 2008. [E-Book] [https://doi.org/10.1142/9781848163003\\_0017](https://doi.org/10.1142/9781848163003_0017) [Accessed: 7 February 2021]
- [22] S. Brenner et al, *Distribution of Proflavin-Induced Mutations in the Genetic Fine Structure*. Nature, 1958. <https://doi.org/10.1038/182983a0> [Accessed: 21 February 2021]
- [23] S. Brunak et al, *Nucleotide Sequence Database Policies*, International Nucleotide Sequence Database Collaboration, 2002. [Online] <http://www.insdc.org/policy> [Accessed: 21 February 2021]
- [24] I. Bellos and M. Ferguson, *Moving from a Product-Based Economy to a Service-Based Economy for a More Sustainable Future*, George Mason University, 2015. [E-Book] <https://mason.gmu.edu/~ibellos/bfchapter.pdf> [Accessed: 28 February 2021]

- [25] *FY21 Molecular Bioinformatics Core (mBIO) Services and Fees*, The University of Tennessee Health Science Center, 2020. [E-Book]  
<https://www.uthsc.edu/research/institutional-cores/mbio/documents/fy21-price-list-mbio.pdf> [Accessed: 28 February 2021]
- [26] R. Ballardini et al, *Artificial Molecular-Level Machines: Which Energy To Make Them Work?*, ACS Publications, 2001. [E-Book] <https://doi.org/10.1021/ar000170g> [Accessed: 7 March 2021]
- [27] B. K. Stöcker et al, *SimLoRD: Simulation of Long Read Data*, International Society for Computational Biology, 2016. [E-Book]  
<https://doi.org/10.1093/bioinformatics/btw286> [Accessed: 4 April 2021]
- [28] K. Pearson, *On lines and planes of closest fit to systems of points in space*, Philosophical Magazine, 1901. [E-Book] <https://doi.org/10.1080/14786440109462720> [Accessed: 6 April, 2021]
- [29] L. Maaten and G. Hinton, *Visualizing Data using t-SNE*, Journal of Machine Learning Research, 2008. [E-Book] <http://www.cs.utoronto.ca/~hinton/absps/tsnefinal.pdf> [Accessed: 6 April 2021]
- [30] J. Drake and G. Hamerly, *Accelerated k-means with adaptive distance bounds*, Baylor University, 2012. [E-Book]  
[http://opt.kyb.tuebingen.mpg.de/papers/opt2012\\_paper\\_13.pdf](http://opt.kyb.tuebingen.mpg.de/papers/opt2012_paper_13.pdf) [Accessed: 12 April 2021]