

Investigating the frequency of tandem stop codons in eukaryotic & prokaryotic genomes

CS39440 Major Project Report

Author: Your Name (tob31@aber.ac.uk)

Supervisor: Dr Wayne Aubrey (waa2@aber.ac.uk)

28th February 2021

Version 1.0 (Draft)

This report is submitted as partial fulfilment of a BSc degree in
Computer Science with foundation year(G40F)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work, I understand and agree to abide by the University's regulations governing these issues.

Name: Thomas William Boyle

Date: 03/05/2023

Consent to share this work

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name: Thomas William Boyle

Date: 03/05/2023

Acknowledgements

I am grateful to my supervisor Wayne Aubrey for guiding me throughout this project.

I am also grateful to Liang, H., Cavalcanti, A.R. & Landweber, L.F. Who wrote the paper on Conservation of tandem stop codons in yeasts. Their research was the inspiration for my project.

I am grateful to Aberystwyth university for letting me undertake this project in my final Year.

I am grateful to JetBrains for creating the Pycharm IDE that I used to develop software for this project and for them making the Community edition open source and free to use.

Abstract

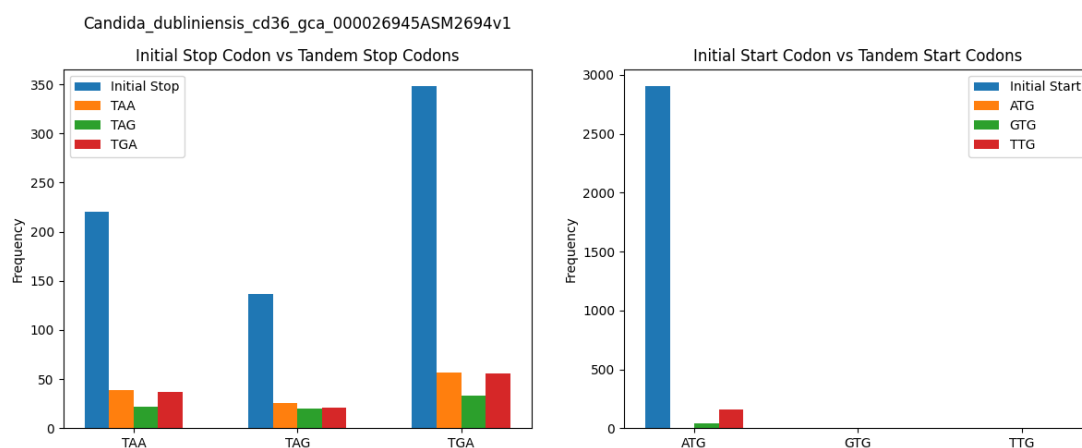
This Project in the area of bioinformatics aims to investigate the presence and usage of tandem stop codons in intergenic regions of DNA, both in prokaryotic genomes and eukaryotic genomes. Recent studies have confirmed the existence of these tandem stop codons in genomes, and this project seeks to better understand and elaborate why and how they are being used.

To achieve this goal, Python code has been developed to analyse data and generate a text file containing the frequencies of tandem stop codons based on their initial stop codon, as well as the GC content. The project utilises top down dna Fasta files and their GFF files to extract relevant information for analysis. The generated data is then used to create bar charts, which visually represent the patterns in tandem stop codon usage in different genomes. These charts are stored in a separate directory within the project for easy access and reference. For example,

C:\Users\Tommy\PycharmProjects\test1\Output\graphs\bar_chart_Candida_dubliniensis_cd36_gca_000026945ASM2694v1.png

The results of this investigation are expected to provide insights into the usage of tandem stop codons in different genomes, further shedding light on their functional significance. The analysis of tandem stop codon usage in both eukaryotic and prokaryotic genomes could elaborate further information about their distribution within these organisms. The findings of this project could be useful in fields such as molecular biology, genetics, and biotechnology.

Lastly, this project uses software engineering techniques to study the presence and usage of tandem stop codons in intergenic regions of DNA, with the aim of contributing to our current understanding of their biological significance. The Python code I developed in this project could be utilised for further research in this area, potentially by a team of researchers in the future.



Contents

1. BACKGROUND, ANALYSIS & PROCESS.....	9
1.1. Background	9
1.2. Analysis.....	10
1.3. Process.....	13
2. EXPERIMENT METHODS	15
2.1. Introduction.....	15
2.2. Overall Hypothesis	15
3. DESIGN	16
3.1. Introduction.....	16
3.2. Design Flow diagram	16
3.3. Detailed Class Design.....	17
3.3.1. Openfile.....	17
3.3.2. GcContent.....	17
3.3.3. parse_fasta.....	18
3.3.4. parse_gff.....	18
3.3.5. get_End_Seq.....	19
3.3.6. get_initial_StopCodon	19
3.3.7. get_start_Seq.....	20
3.3.8. get_initial_StartCodon.....	20
3.3.9. tandemStopCount.....	20
3.3.10. tandemStartCount.....	21
3.3.11. displayTandemStopCodon	22
3.3.12. displayTandemStartCodon.....	22
3.3.13. displayFinalinfo.....	23
3.3.14. barchartCreate.....	23
3.3.15. stopVariableInitialisation	24
3.3.16. startVariableInitialisation	25
3.3.17. Main	25
3.4. Input.....	27
3.5. Output	28
4. IMPLEMENTATION	30
4.1. Introduction.....	30
4.2. Issues encountered.	30

4.2.1.	Initial Changes of requirements in this Project.....	30
4.2.2.	Opening the files	30
4.2.3.	Wrong Fasta File provided.	30
4.2.4.	Problems with Synder python ide	30
4.2.5.	Prokarotyoic geneomes?	30
4.3.	Time specific project developments	31
4.4.	Minor last-minute Changes	32
	33
5.	TESTING	34
5.1.	Introduction and Strategy.....	34
5.2.	Overall Approach to Testing	34
5.3.	Unit Testing.....	34
5.3.1.	Unit Tests table.....	34
5.3.2.	Screenshots	35
5.4.	Integration and System Testing.....	40
5.4.1.	Integration and System Tests table	40
5.4.2.	Integration and System Testing Screenshots.....	42
6.	RESULTS AND CONCLUSIONS.....	44
6.1.	Introduction.....	44
6.2.	Eukaryotic genomes Results.....	44
6.3.	The Eukaryotic genomes graphs.	45
6.4.	Prokaryotic genomes?.....	48
6.5.	Conclusions	52
7.	CRITICAL EVALUATION	52
7.1.	Introduction.....	52
7.2.	Project Aim	52
7.3.	Requirements.....	53
7.4.	Design.....	54
7.4.1.	Choice of Tools, IDE, Programming language	54
7.4.2.	Single file python file design.....	54
7.5.	Project management.....	55
7.6.	Testing strategy	55
7.7.	Future Development.....	55

8. REFERENCES.....	56
9. FIGURES.....	56
10. APPENDICES	56
10.1. Tools used.....	56
10.1.1. Winzip.....	56
A. Third-Party Code and Libraries.....	57
B. Code Samples.....	59
10.1.2. Code examples provided	59

1. Background, Analysis & Process

1.1. Background

Ever since taking the bioinformatics module in my third year I had a keen interest in the field of bioinformatics and undertaking this project would allow me to bring together much of what I have learnt through my course to produce this project. My background research/preparation was to reaffirm my knowledge of stop codons usage in genetic sequences and potentially further my knowledge, I did this by doing my own online research and reading an academic paper containing work of similar nature [1] Conservation of tandem stop codons in yeasts.

What are Stop codons, stop codons are a set of three nucleotides that signal the termination of protein synthesis during translation. There are three. TAA, TAG, TGA. They must be in frame with the start codon, so in sets of three.

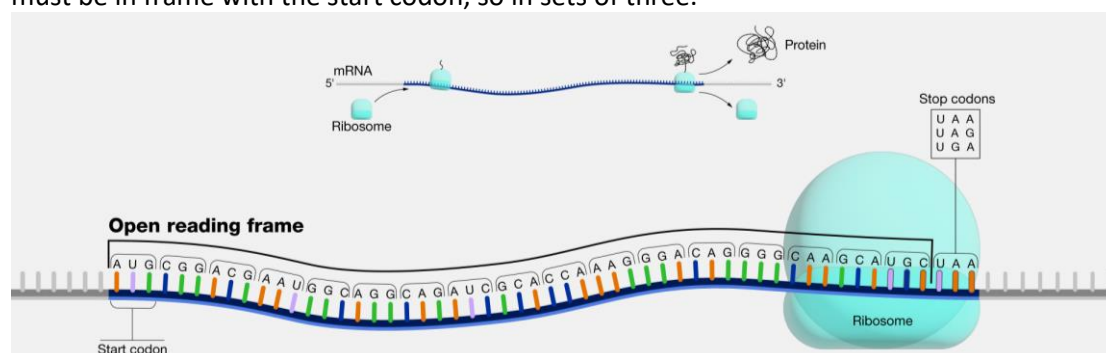


Figure 1

What are nucleotides, Nucleotides are organic molecules that are the building blocks of nucleic acids which are large molecules. Three nucleotides make a codon. They are composed of three main components. A nitrogenous base, a sugar molecule, and a phosphate group. The nitrogenous base can be adenine (A), cytosine (C), guanine (G), or thymine (T) in DNA, or uracil (U) in RNA. The sugar molecule in DNA is deoxyribose, while in RNA, it is ribose.

What are tandem Stop codons, Tandem stop codons are two or more stop codons located in close proximity at the end of a gene in a genetic sequence. they are extra stop codons that appear almost directly after the gene ends, it acts as a backup in case of a read through error by the ribosome. Currently their role in gene expression is not completely understood and may require further research. In a recent paper it was concluded that they occur in the first 12 bases after the sequences ends, the paper was called [1]“Conservation of tandem stop codons in yeasts”

What are Start codons and tandem start codons?

Similar, to stop codons they are a set of three nucleotides used to represent the start of the gene. However, are their tandem start codons and will they act as a backup in the case of a read through error by the ribosome. Also, where might they be located? These are things my project intends so shed some light on.

What are Eukaryotic genomes, they are a group of organisms that include fungi, protist, and animals, and they have complex cells with a nucleus that contains the genetic material in the form of DNA. They are typically larger and more complex than prokaryotic genomes, Eukaryotic genomes are organised into multiple linear chromosomes. Eukaryotic genomes are quite diverse, they display varying sizes and gene content.

What are Prokaryotic genomes, they are a group of organisms that includes archaea and bacteria, they lack a nucleus and other membrane-bound organelles. They are typically smaller and simpler than eukaryotic genomes. They consist of a single circular DNA molecule. They are efficient and compact, with barely non-coding regions or introns.

1.2. Analysis

This Project aims to investigate the presence and usage of tandem stop codons in intergenic regions of DNA, both in eukaryotic and prokaryotic genomes. These tandem stop codons could possibly affect gene expression because it alters the process of the sequence translation.

Objectives for this project and what will be evaluated at the end.

1. Collect the data
2. Extract the correct info from the files
3. Identify the tandem stop codons/start codons and there frequencies and indexes.
4. Produce an output Text file and Barchart graph of this information for visual format.

The output Text files and barcharts for each genome will be evaluated at the end, to draw conclusions for this research project and also assess the projects success.

The data used for this project.

I am using a collection of “top down dna Fasta files” and “Gff files” obtained from the ensemble.org website. <https://ftp.ensembl.org/pub>. “Ensembl is a genome browser for vertebrate genomes that supports research in comparative genomics, evolution, sequence variation and transcriptional regulation.” These files where collected for me by my supervisor, Wayne Aubrey, and placed into a shared Onedrive directory that I had access to. I have also used some Ensembl bacteria files provided to me by Wayne later in the project. However these file are stored in a different format to the ensemble fungi files from Onedrive



Figure 2

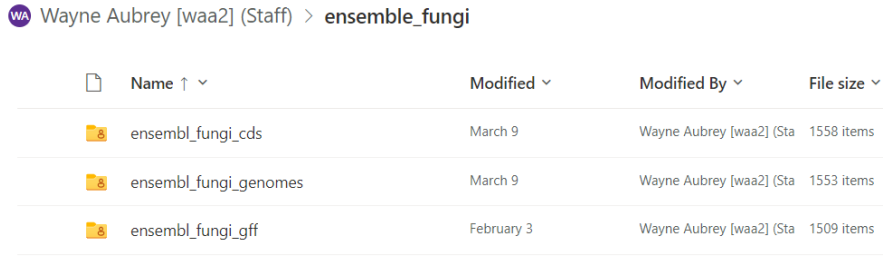
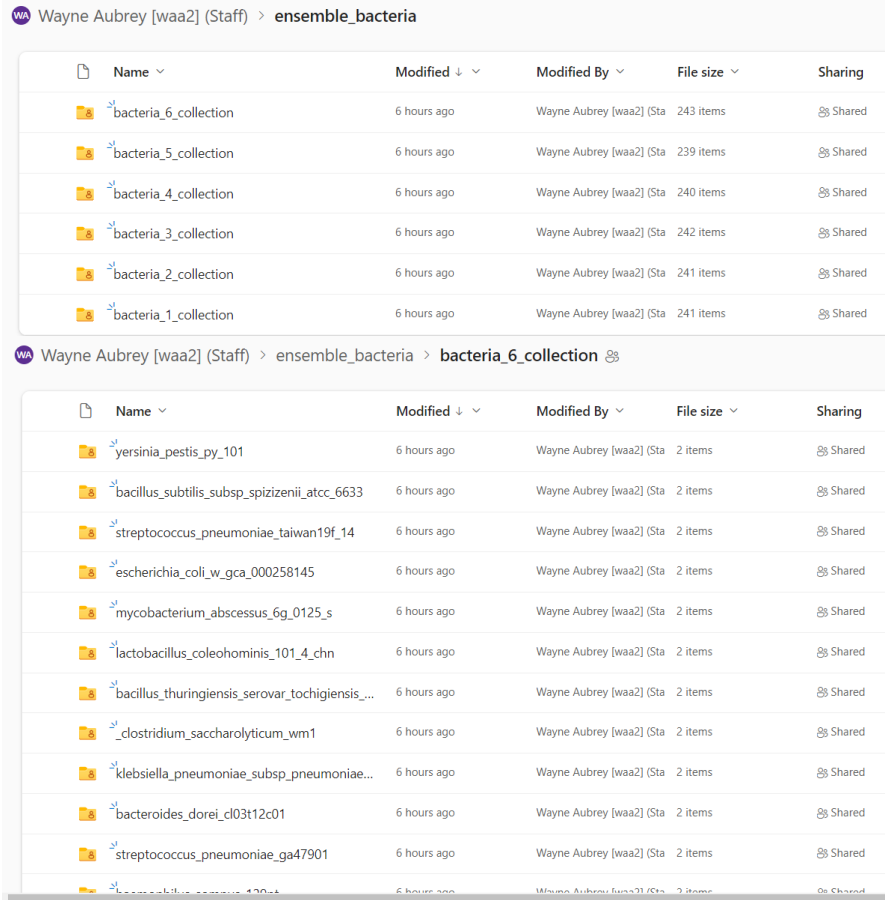


Figure 3



Figures 4

I was using the collection of files provided to me by Wayne via OneDrive however the fasta files were only the cds reads not the toplevel dna files so there were no tandem stop codons in the original files used. This was a setback in my project that was only discovered briefly before the mid project demonstration.

What do these files look like, and what needs to be extracted/are we looking for.

Candida_dubliniensis_unzipped_read_gff3 - Notepad

```
File Edit Format View Help
##gff-version 3
##sequence-region 1 1 3214061
##sequence-region 2 1 2289089
##sequence-region 3 1 1863824
##sequence-region 4 1 1641709
##sequence-region 5 1 1245899
##sequence-region 6 1 1073895
##sequence-region 7 1 1022435
##sequence-region R 1 2267510
#!genome-build Wellcome Trust Sanger Institute ASM2694v1
#!genome-version ASM2694v1
#!genome-build-accession GCA_000026945.1
#!genebuild-last-updated 2015-02
1 ASM2694v1 chromosome 1 3214061 . . . ID=
###
1 ena gene 12543 13448 . + . ID=gene:CD36_00030;l
1 ena mRNA 12543 13448 . + . ID=transcript:CAX44;
1 ena exon 12543 13448 . + . Parent=transcript:CAX44;
1 ena CDS 12543 13448 . + 0 ID=CDS:CAX44265;Par
###
1 ena_mobile_element biological_region 12547 13148 . +
1 ena_mobile_element biological_region 13149 13702 . +
1 ena_mobile_element biological_region 13703 15131 . +
1 ena_mobile_element biological_region 15163 20699 . +
1 ena_mobile_element biological_region 15365 20691 . +
1 ena gene 15930 16871 . + . ID=gene:CD36_00050;l
1 ena mRNA 15930 16871 . + . ID=transcript:CAX44;
1 ena exon 15930 16871 . + . Parent=transcript:CAX44;
1 ena CDS 15930 16871 . + 0 ID=CDS:CAX44267;Par
... ..
```

This is a snapshot example GFF file for *Candida dubliniensis*,
The information highlighted in yellow is what my program collects from the gff files. In this example the 0th element is the chromosome is 1, the 3rd element is the start region of the gene, and the 4th element is the stop region on the gene, the 8th element is the gene ID which I also record.

Highlighted in red is the strand the gene is on +/- . The reason this is highlighted in red is because give more time, I would implement a way of using this when extracting the information from the GFF file. However, my program does not currently do this.

Candida_dubliniensis_unzipped_read_fasta - Notepad

```
File Edit Format View Help
>1 dna:chromosome chromosome:ASM2694v1:1:1:3214061:1 REF
GATCAAGTTGAGAGACAAATAGAGTTGTTTATTTAATTCAGAGAAGAATCAGTTGTTCA
TTGTTAAGATCACAGACAGAATTCTGTTGTTTGTAGTCGCAAAAGAATCAGCTACAAT
ACAGTTAGAGATACAGTATAGATTAGAGAGAGCAGTTAATTTGTGATCAAGTCGAGAGA
CAATAGAGTTGTTTATTTAATTCAGAGAAGAATCAAAATTTAGTTATTCATAGACAGAA
TACTGTTGCTCGTTTAGTCACATAAGAACCGGATACAATACAGTTAGAGATGCAATATAG
ATTTAGAGAGAACGCTAGAGCAGTAAATCTATGTTGAGCTTCAGAGACTAACTGAGTTG
TTTATTTACTTCCAGGTTGATAACAAATTTGTTTATTTAATTCACAAAAGAATCA
AATTTAGTTGTTTATTTGAGTTACGGAGTTTGTGTTATTTTGAGTTACAGATCAAA
TACTGTTGTTTGTAGAAAAATCAGATATATTTCAAGTTACCAATACTATTTCAAGTTACA
GATCCCTTATAGCTTGAGAGCAGTTAACCTAGACTCCAATTTATGCTCCGTGAGTTCTGC
TCGTAATTGTGCTCATTGTCAATAACTTCTACTCTTCACTCTTCCCAATTTCAATTGT
CATATTTCTCTCTATTAATCATTTCTCTGTTATCAATTTTGAATTATGACAAGTTTAA
CAGTTTCCTGACTTCCATGAATTTGTGATGTTGATTAAACAAATGACAAATTTGATT
CACATTTCAACCAAGAATTTGAAGTAGATGCAACAAGAAGTTATTACTACTCTCTGTGT
GATTGTCTATTTTGTGCTGAATGATGCTTTTCCAATTTCTAATATCTGCCAATCTATT
CCGGAACACTGAGATATAGCCATTGAATTAAGCAAGTGACCAAAATTCATGCCTGTGTG
ATTGCTTGTCTATTCTTCCCGGACGTTGTTTCTTTAATTGTCAAATCTTCTGCCAATC
TGTTTCTGAACACTGCAATATTGCCATTGAATCTATTGAAAGTGCTTTCCTTGAAAAAGA
```

This is a snapshot example top down dna file for *Candida dubliniensis*, this is where the start and stop regions of the genes are indexed from for each chromosome.

Future directions

With more time on the project I would like to implement a way of displaying the indexes of the tandem codons on the barchart. The indexes are currently recorded in the output txt file however this is for each gene, and there isn't a recorded frequency.

I would also like to make my program more defensive, when extracting information that might not be present in the files.

As well as, implementing a way to record what strand the gene is on.

I would also like to create a gui to display information and make the program more user friendly especially to biologists.

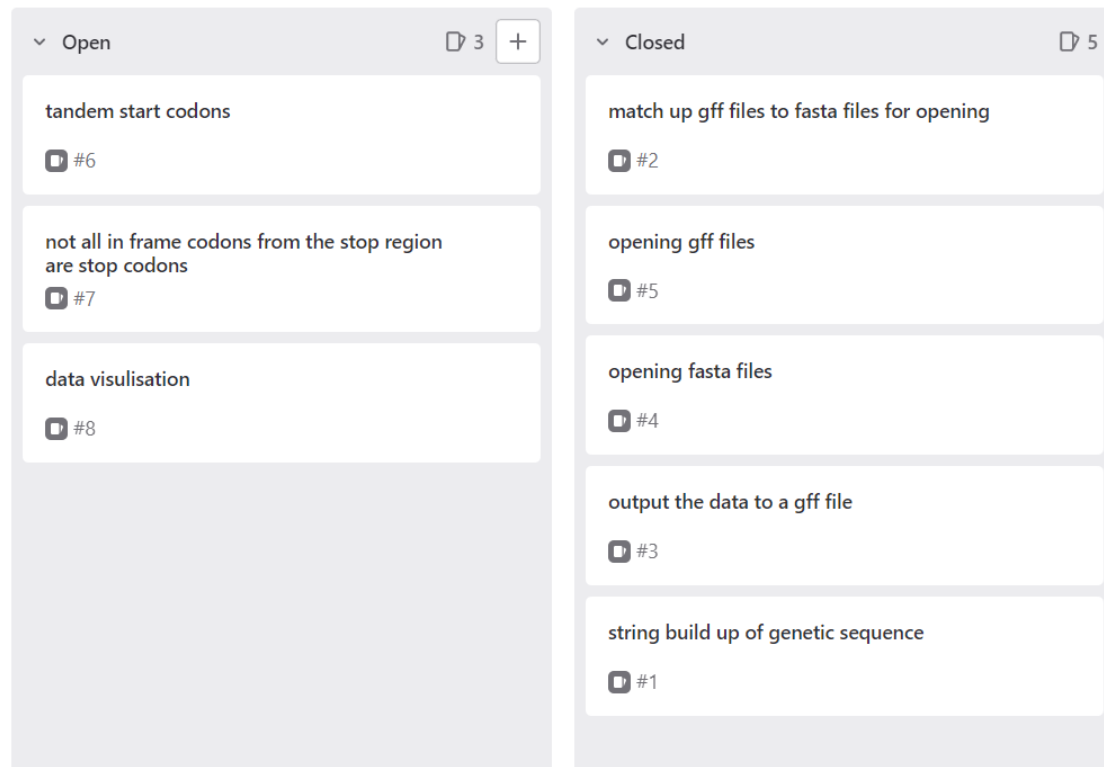
1.3. Process

You need to describe briefly the life cycle model or research method that you used. You do not need to write about all of the different process models that you are aware of. Focus on the process model that you have used. It is possible that you needed to adapt an existing process model to suit your project; clearly identify what you used and how you adapted it for your needs.

For this project I used the kanban agile development methodology,

I've used the built in gitlab Kanban board, this allows me to effectively keep track of tasks that need completing, I choose Kanban because of its visual nature, and I am more of a visual learner, and it helps me manage the workload better.

Kanban was also a good fit for this project, as it gave me flexibility to adapt while I was developing the project since the scope and the requirements changed slightly over time, for example I began to look for tandem start codons. Likewise, the continuous delivery aspect of kanban was useful, since I had weekly group meetings on Mondays and personal meetings with Wayne on Thursdays this meant I could continuously deliver improvements weekly.



Continuous integration

For Continuous integration, I choose to use GitLab because of how easy it is to use and setup, pipeline can be configured directly, without the need for plugins. Its use in the second-year group Project has also given me experience using this form of CI before, thus it was an obvious choice.

Gitlab also has a robustly high level of security, and although this isn't necessarily of high importance for this research project it is good addition to have.

Gitlab's collaboration services which usually allow teams to continuously work of developing software together is also useful, since it allowed my supervisor to view my work whilst I developed it.

IDE and Programming language Choice

The programming language I choose to use for this bioinformatics project was python. Python has a wide list of libraries that can be utilised for bioinformatics projects, for example Biopython and NumPy. I do not use these libraries in my project however I do use matplotlib which is a visualising and graphing data. Matplotlib was very useful in my project when creating the bar-charts of tandem start/stop codons. Python is also used for many bioinformatics projects world wide.

Python is also a very versatile language for analysing data, and initially my project was going to utilise machine learning "Using a machine learning model to predict coding and non-coding sequence in DNA" This project description changed however I stuck with the programming language.

My own previous experience with Python, learning it as my first programming language at 14 for GCSE and using it for the bioinformatics module in my third-year 1st semester, made it the clear option for a project of this type.

Pycharm IDE, I chose to use the Pycharm IDE after using synder python initially for this project, the code seemed to be unexplainably slow to run and often crashed. This lead be to switch to PyCharm at Wayne's suggestion, immediately fixing these issues I had with synder.

PyCharm is well suited to this project since PyCharm supports version control systems like git to manage the code saves, as well as collaboration systems making it possible to program and discuss code with the project supervisor online. PyCharm's debugging tool has been particularly useful for this project, allowing me to fix issues effectively.

2. Experiment Methods

2.1. Introduction

In this section I will discuss the overall hypothesis being tested for by the undertaking of this project and what I hope to achieve and produce results for.

2.2. Overall Hypothesis

The hypothesis for this project is that the;

"Type of tandem Start codon and stop codon used by be specific genes and their frequency could potentially be genome specific instead of random".

It is not currently fully understood why specific tandem stop codons are used, we know that tandem stop codons are used as a backup in case of a readthrough error when the gene is being read by the ribosome but there are 3 stop codons, are specific tandem stop codons used more than others and why. We know that the type of stop codon used could is gene specific however is their tandem stop codon also gene specific and could there potentially be whole genome specific.

The same could be said for start codons part of this project will be to identify if tandem start codons exist and if they do exist will the type of initial start codon be genome specific and will the type tandem start codons be genome specific or specific to the type of initial start codon.

Also, it has being confirmed that tandem stop codons appear directly after the initial stop codon in the first 12 bases. So, this project will also determine if tandem start codons will appear in the first 12 bases similar to stop codons but before the initial start codon.

I hope to produce graphs for each genome depicting the frequency of tandem stop and tandem start codons being use for each genome.

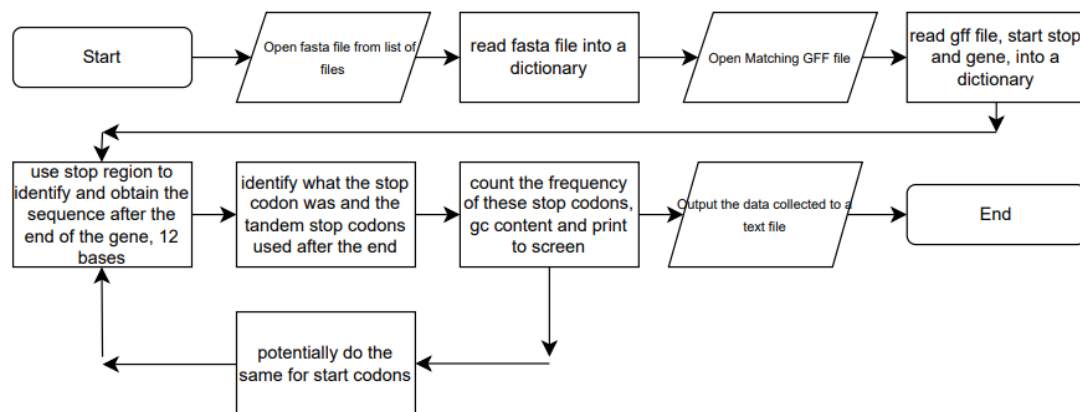
3. Design

3.1. Introduction

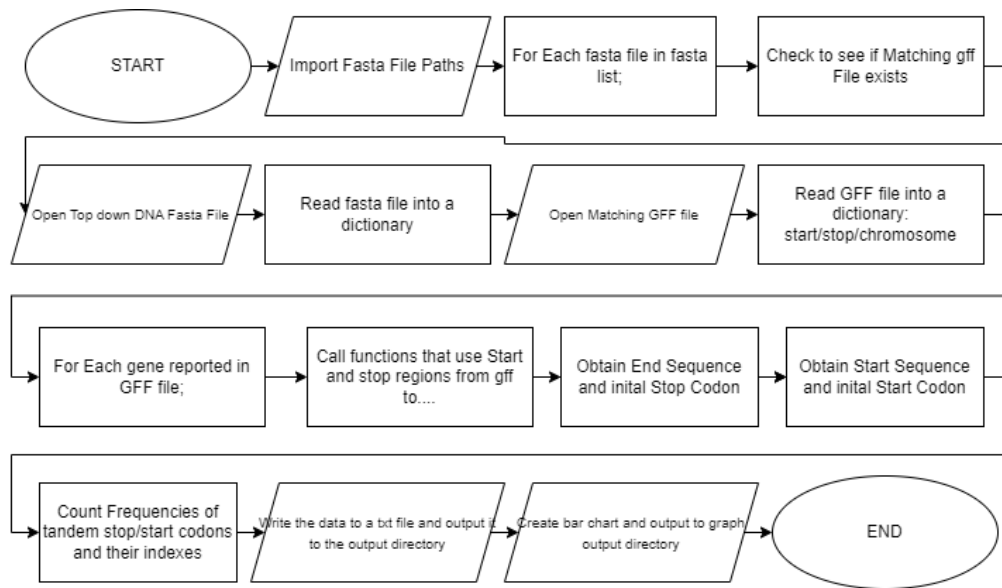
In this section I will discuss the programming design of this project, the functions I created, what they do and what the output for using this application will look like.

3.2. Design Flow diagram

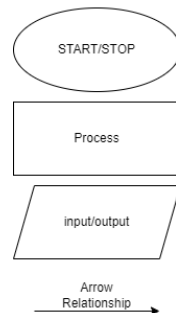
Initial early flow chart design:



Complete Flow chart design:



Key:



3.3. Detailed Class Design

3.3.1. Openfile

This function is used for opening files, more specifically, it is used to check if the input file is gz zipped or not and opens the file, accordingly, returning a list of each line in the file as a list item using the “readlines” inbuilt python method.

```

def openfile(filePath): # opening and reading the fastafile and gfffile
    if filePath.endswith(".gz"):
        with gzip.open(filePath, 'rt') as f:
            return [l.strip() for l in f.readlines()]
    else:
        with open(filePath, 'r') as f:
            return [l.strip() for l in f.readlines()]
  
```

3.3.2. GcContent

This function calculates the gc content for the fasta file, it's a simple calculation, counting the c and g then dividing by the length of the sequence and multiplying by 100 to express it as a percentage not a decimal.

```

def GcContent(seq): # calculating the gc content percentage
    return round((seq.count('C') + seq.count('G')) / len(seq) * 100, 6)
  
```

3.3.3. parse_fasta

This function is used to create a dictionary for the top down dna fasta file with the **chromosome id** as the key and the **nucleotide sequence** as the value, this function takes in the filepath for the fastafile, calls the openfile function, puts the contents of the fasta file into a dictionary and returns that dictionary.

Candida_dubliniensis_unzipped_read_fasta - Notepad

File Edit Format View Help

```
>1 dna:chromosome chromosome:ASM2694v1:1:1:3214061:1 REF
GATCAAGTTGAGAGACAAATAGAGTTGTTTATTTTAAATTCAGAGAAGAATCAGTTGTTCA
TTGTTAAGATCACAGACAGAATTCTGTTGTTTGTAGTCGCAAAAGAATCAGCTACAAT
ACAGTTAGAGATACAGTATAGATTAGAGAGAGCAGTTAATTTGTGATCAAGTCGAGAGA
CAAATAGAGTTGTTTATTTTAAATTCAGAGAAGAATCAAATTTAGTTATTCATAGACAGAA
TACTGTTGCTCGTTTAGTCACATAAGAACCGGATACAATACAGTTAGAGATGCAATATAG
ATTAGAGAGAACGCTAGAGCAGTAAATCTATGTTGAGCTTCAGAGACTAACTGAGTTG
TTTATTTACTTCCAGGTTGATAAACAAATGTTTATTTATTTTAAATTCACAAAAGAATCA
AATTTAGTTGTTCAATTTTGTGTTTACGGAGTTTGTATTTTGTAGTTACAGATCAAA
TACTGTTGTTTGTAGAAAAATCAGATATATTTTCACTTACCAATATCTTTTCACTTACA
GATCCCTATAGCTTGAAGCAGTTAACCTAGACTCCAATTTATGCTCCGTGAGTTCTGC
TCGTAATTGTGCTCATTGTCAATAACTTCTACTCTTCACTCTTCCCAATTTTCAATGT
CATATCTTCTCTATTAATCAATTTCTCTGTTATCAATTTTGAATTATGACAAGTTTAA
CAGTTTCTCGACTTCCATGAATTGTGATGTTGATTAAACAACTGACAAATTTGATT
CACATTTCAACCAAGAATTGAAGTAGATGCAACAAGAAGTTATTACTACTCTCTGTGT
GATTGTCTATTTTGTGCTGAATGATGCTTTTCAATTTTCTAATATTCTGCCAATCTATT
CCGGAACACTGAGATATAGCCATTGAATTTAAGCAAGTGACCAAAATTCATGCCTGTGTG
ATTGCTTGTCTATTCTTCCGGACGTTGTTTCTTTTAAATGTCAAATCTTCTGCCAATC
TGTTTCTGAACACTGCAATATTGCCATTGAATCTATTGAAAGTGCTTTCCTTGAAAAAGA
```

```
def parse_fasta(filename):
    FastaFile = openfile(filename)
    FDict = {}
    chromosomeID = ""
    nucleotide_sequence = ""
    for line in FastaFile:
        line = line.strip()
        if line.startswith(">"):
            if chromosomeID != "":
                FDict[chromosomeID] = nucleotide_sequence
                nucleotide_sequence = ""
            chromosomeID = line.strip().split(' ')[2].split(':')[2]
        else:
            nucleotide_sequence += line
    # adds the last sequence to the dictionary because there isn't a > after the last line
    FDict[chromosomeID] = nucleotide_sequence
    return FDict
```

3.3.4. parse_gff

Similar to the parse_fasta function, this function is used create a dictionary with the information from the corresponding gff file, the **gene id** is the key and the values are the **start region**, **stop region** and the **chromosome**. There is a colour coded picture to represent the values extracted from the gff file.

```
###
1      ena      gene      12543      13448      .      +      .      ID=gene:CD36_00030;b
1      ena      mRNA      12543      13448      .      +      .      ID=transcript:CAX442
1      ena      exon      12543      13448      .      +      .      Parent=transcript:CA
1      ena      CDS      12543      13448      .      +      0      ID=CDS:CAX44265;Pare
###
```

```
def parse_gff(gff_file):
    gff_dict = {}
    f = openfile(gff_file)
    for line in f:
        if line.startswith('#'):
            continue
        fields = line.strip().split('\t')
        if fields[2] == 'gene':
            gene_id = fields[8].split(':')[0]
            start = int(fields[3])
            stop = int(fields[4])
            chromosome = fields[0]
            gff_dict[gene_id] = (start, stop, chromosome)
    return gff_dict # Press Ctrl+F8 to toggle the breakpoint.
```

3.3.5. get_End_Seq

This function “get the end” sequence that appears after the stop codon, this sequence is where the tandem stop codons might be found thus this region must be obtained. It takes in the GFF dictionary and the fasta file dictionary and the current gene being accessed. It indexes the 1st and 2nd element of the gff dictionary with the gene as the key, to access that genes stop region and its chromosome. Then indexes the fasta File with the chromosome as the key then the stop region. The stop region refers to where the stop codon is, and is a singular nucleotide of that stop codon. Using slice notation, we can gain the sequence of nucleotides that appear after the stop codon by indexing them. The academic study [1] “The conservation of tandem stop codons” mentions these tandem stop codons appear in the first 12 bases after the stopcodon, so that’s where I have obtained. StopRegion + 2 : StopRegion + 14.

```
def get_End_Seq(gff_dict, FDict, gene):
    stopRegion = gff_dict[gene][1] #stop coordinates of gene
    chromosome = gff_dict[gene][2]
    endSeq = (FDict[chromosome][stopRegion + 2:stopRegion + 14])
    return endSeq
```

3.3.6. get_initial_StopCodon

This function, Is the same as the previous one accepts that it uses the slice notation to index a different part of the sequence so that it can obtain just three nucleotides these being the initial stop codon, it takes in the GFF and the fasta file dictionary and the current gene. It indexes the 1st and 2nd element of the gff dictionary with the gene as the key, to access that genes stop region and its chromosome. Then indexes the fasta File with the chromosome as the key then uses slice notation with the stop region to obtain the initial stopcodon. This function returns the initial stop codon.

```
def get_Initial_StopCodon(gff_dict, FDict, gene):
    stopRegion = gff_dict[gene][1]
    chromosome = gff_dict[gene][2]
    initialStopCodon = (FDict[chromosome][stopRegion - 1:stopRegion+2])# stop codons ???
    return initialStopCodon
```

3.3.7. get_start_Seq

This function gets start sequence, this start sequence is before the initial start codon that signals the start of the gene, this region is where I predicted that start codons might appear. In the academic paper [1] “the conservation of tandem stop codons” concludes that these stop codons appear in the first 12 bases after the stop codon, so similarly for start codons I looked for the first 12 bases before the start codon. The function takes in the gff dictionary and the fasta file dictionary and the current gene, to index the 0th and 2nd elements of the gff dictionary to obtain the start region and the chromosome. These could then be used with slice notation to index the sequence. This function returns that sequence that the start codons may appear.

```
def get_start_Seq(gff_dict, FDict, gene):
    startRegion = gff_dict[gene][0]
    chromosome = gff_dict[gene][2]
    startSeq = (FDict[chromosome][startRegion - 13:startRegion - 1])_# front end
    return startSeq
```

3.3.8. get_initial_StartCodon

Similar previous functions this function is used to obtain the initial start codon for the gene and return it as a string. It takes in the gff dictionary fasta file dictionary and current gene as arguments and uses them to index these nucleotide locations.

```
def get_Initial_StartCodon(gff_dict, FDict, gene):
    startRegion = gff_dict[gene][0]
    chromosome = gff_dict[gene][2]
    initialStartCodon = (FDict[chromosome][startRegion - 1:startRegion + 2])_# start codons ???
    return initialStartCodon
```

3.3.9. tandemStopCount

This function is used to obtain all of the tandem stop codons and their indexes from the “end sequence” the string of the nucleotides that appear after the stop. Freq and indexes are lists of the codons and their indexes. Stop_codons is a list of all the possible stop codons to look for. It uses for loops to loop through the end sequence and the stop codons to iterate integers for each of the stop codons, and then return a list of all of these individual stop codon counts based on their initial start codon. So that it can be displayed later.

```

def tandemStopCount(endSeq, initialStopCodon, total_stop_codons, total_stop_codons_dict):
    stop_codons = ["TAA", "TAG", "TGA"]
    initialStopCodon_count = 0
    tan_TAA_count = 0
    tan_TAG_count = 0
    tan_TGA_count = 0
    freq = {codon: 0 for codon in stop_codons}
    indexes = {codon: [] for codon in stop_codons}
    for i in range(0, len(endSeq) - 2, 3): # loops over all stop codons in the endseq
        if len(endSeq) - i >= 3: # Check if there are at least 3 characters left to form a codon
            codon = endSeq[i:i + 3]
            if codon in stop_codons:
                if (i + 3) % 3 == 0:
                    freq[codon] += 1
                    indexes[codon].append(i)
                    if codon == 'TAA':
                        tan_TAA_count += 1
                    if codon == 'TAG':
                        tan_TAG_count += 1
                    if codon == 'TGA':
                        tan_TGA_count += 1
                total_stop_codons += 1
    if initialStopCodon in stop_codons:
        initialStopCodon_count += 1
    total_stop_codons_dict[initialStopCodon] = (
        total_stop_codons_dict[initialStopCodon][0] + initialStopCodon_count,
        total_stop_codons_dict[initialStopCodon][1] + tan_TAA_count,
        total_stop_codons_dict[initialStopCodon][2] + tan_TAG_count,
        total_stop_codons_dict[initialStopCodon][3] + tan_TGA_count
    )
    tandemStopCountData = [freq, indexes, total_stop_codons, total_stop_codons_dict]
    return tandemStopCountData

```

3.3.10. tandemStartCount

This function is exactly the same as the last functions for the stop codons except it does this for start codons. So the key differences here to look at are the list of potential codons contains start codons, and the function returns a list of the individual start codon counts based on their initial start codon. So that it can be displayed later.

```
def tandemStartCount(startSeq, initialStartCodon, total_start_codons, total_start_codons_dict):
    start_codons = ["ATG", "GTG", "TTG"]
    initialStartCodon_count = 0
    tan_ATG_count = 0
    tan_GTG_count = 0
    tan_TTG_count = 0
    freq = {codon: 0 for codon in start_codons}
    indexes = {codon: [] for codon in start_codons}
    for i in range(0, len(startSeq) - 2, 3): # loops over all stop codons in the startseq
        codon = startSeq[i:i + 3]
        if codon in start_codons:
            if (i + 3) % 3 == 0:
                freq[codon] += 1
                indexes[codon].append(i)
                if codon == 'ATG':
                    tan_ATG_count += 1
                if codon == 'GTG':
                    tan_GTG_count += 1
                if codon == 'TTG':
                    tan_TTG_count += 1
            total_start_codons += 1
    if initialStartCodon in start_codons:
        initialStartCodon_count += 1
        total_start_codons_dict[initialStartCodon] = (
            total_start_codons_dict[initialStartCodon][0] + initialStartCodon_count,
            total_start_codons_dict[initialStartCodon][1] + tan_ATG_count,
            total_start_codons_dict[initialStartCodon][2] + tan_GTG_count,
            total_start_codons_dict[initialStartCodon][3] + tan_TTG_count
        )
    tandemStartCountData = [freq, indexes, total_start_codons, total_start_codons_dict]
    return tandemStartCountData
```

3.3.11. displayTandemStopCodon

This function is for displaying the stop codon data into the text file output, it takes in the file and also the stop codon data like the frequency, indexes the current gene and the initial stop codon. As you can see there are print statements here, which almost mirror copies of the file writing statements, these are for displaying the information to the terminal. A for loop is used to loop through the count of stop codon frequencies.

```
def displayTandemStopCodon(stopFreq, stopIndexes, gene, initialStopCodon, f):
    # print("Stop codon frequencies:", gene + ":")
    # print("The initial stop codon was: " + initialStopCodon)
    f.write("Stop codon frequencies: " + gene + ":\n")
    f.write("The initial stop codon was: " + initialStopCodon + "\n")
    for codon, count in stopFreq.items():
        # print(codon, ":", count, "at indexes", stopIndexes[codon])
        f.write(codon + " : " + str(count) + " at indexes " + str(stopIndexes[codon]) + "\n\n")
    return f
```

3.3.12. displayTandemStartCodon

Similarly, this function is the same as the function above, accept this is for start codons, it would be possible to have both functions in one to potentially save repeated code use. However, for the purpose of good coding practice and presentation this is better.

```
def displayTandemStartCodon(startFreq, startIndexes, gene, initialStartCodon, f):
    # print("Start codon frequencies:", gene + ":")
    # print("The initial start codon was: " + initialStartCodon)
    f.write("Start codon frequencies: " + gene + ":\n")
    f.write("The initial start codon was: " + initialStartCodon + "\n")
    for codon, count in startFreq.items():
        #print(codon, ":", count, "at indexes", startIndexes[codon])
        f.write(codon + " : " + str(count) + " at indexes " + str(startIndexes[codon]) + "\n\n")
    return f
```

3.3.13. displayFinalinfo

Likewise, to the previous functions this is the final display info function, it displays all of the totalled up information at the end of the text file. The total tandem stop codons, total tandem start codons and the gc content for the genome. The output text file is saved to the output directory C:\Users\Tommy\PycharmProjects\test1\Output\graphs.

```
def displayFinalinfo(total_stop_codons, total_start_codons, GcResults, element, f):
    # print("Total tandem stop codons found: ", total_stop_codons, "\n")
    # print("Total tandem start codons found: ", total_start_codons, "\n")
    # print("Total gc content for " + element + " = " + (str(sum(GcResults.values()) / len(GcResults))) + "\n\n")
    f.write("Total tandem stop codons found: " + str(total_stop_codons))
    f.write("Total tandem start codons found: " + str(total_start_codons))
    f.write("Total gc content for " + element + " = " + (str(sum(GcResults.values()) / len(GcResults))) + "\n\n")
    return f
```

3.3.14. barchartCreate

This function here is for creating the bar chart output graph, it takes in the dictionary of the stop codon data, and the dictionary of the start codon data and indexes them with there position and the specific codon as the key. To fill up the data lists that will be used to plot the graph. The graph is created using matplotlib which is an imported python library for plotting graphs. Matplotlib sets up the figure and makes 2 subplots, then populates the bars on these subplots the create the bar chart, I then add labels to the figure and subplots. Then save the graph to the output directory

C:\Users\Tommy\PycharmProjects\test1\Output\mainTextOutput

```

def barchartCreate(element, total_stop_codons_dict, total_start_codons_dict):
    initial_stop_codons = ['TAA', 'TAG', 'TGA']
    #actual data ??
    initial_counts = [total_stop_codons_dict['TAA'][0], total_stop_codons_dict['TAG'][0], total_stop_codons_dict['TGA'][0]]
    tandem_TAA_counts = [total_stop_codons_dict['TAA'][1], total_stop_codons_dict['TAG'][1], total_stop_codons_dict['TGA'][1]]
    tandem_TAG_counts = [total_stop_codons_dict['TAA'][2], total_stop_codons_dict['TAG'][2], total_stop_codons_dict['TGA'][2]]
    tandem_TGA_counts = [total_stop_codons_dict['TAA'][3], total_stop_codons_dict['TAG'][3], total_stop_codons_dict['TGA'][3]]

    initial_start_codons = ['ATG', 'GTG', 'TTG']
    initial_start_counts = [total_start_codons_dict['ATG'][0], total_start_codons_dict['GTG'][0], total_start_codons_dict['TTG'][0]]
    tandem_ATG_counts = [total_start_codons_dict['ATG'][1], total_start_codons_dict['GTG'][1], total_start_codons_dict['TTG'][1]]
    tandem_GTG_counts = [total_start_codons_dict['ATG'][2], total_start_codons_dict['GTG'][2], total_start_codons_dict['TTG'][2]]
    tandem_TTG_counts = [total_start_codons_dict['ATG'][3], total_start_codons_dict['GTG'][3], total_start_codons_dict['TTG'][3]]

    # Bar chart setup
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5)) # Create subplots for stop codon and start codon bar charts
    width = 0.15 # the width of the bars
    x = [i for i in range(len(initial_stop_codons))] # the x locations for the groups of bars

    # Stop codon bar chart
    # Plot the initial stop codon counts TAA
    bar1 = ax1.bar([i - 1.5 * width for i in x], initial_counts, width, label='Initial Stop')
    # Plot the tandem stop codon counts TAG
    bar2 = ax1.bar([i - 0.5 * width for i in x], tandem_TAA_counts, width, label='TAA')
    # Plot the tandem stop codon counts TGA
    bar3 = ax1.bar([i + 0.5 * width for i in x], tandem_TAG_counts, width, label='TAG')
    # Plot the tandem stop codon counts TGA
    bar4 = ax1.bar([i + 1.5 * width for i in x], tandem_TGA_counts, width, label='TGA')
    ax1.set_xticks(x)
    ax1.set_xticklabels(initial_stop_codons)
    ax1.legend()

    ax1.set_ylabel('Frequency')
    ax1.set_title('Initial Stop Codon vs Tandem Stop Codons')

    # Start codon bar chart
    # Plot the initial start codon counts ATG
    bar1 = ax2.bar([i - 1.5 * width for i in x], initial_start_counts, width, label='Initial Start')
    # Plot the tandem start codon counts ATG
    bar2 = ax2.bar([i - 0.5 * width for i in x], tandem_ATG_counts, width, label='ATG')
    # Plot the tandem start codon counts GTG
    bar3 = ax2.bar([i + 0.5 * width for i in x], tandem_GTG_counts, width, label='GTG')
    # Plot the tandem start codon counts TTG
    bar4 = ax2.bar([i + 1.5 * width for i in x], tandem_TTG_counts, width, label='TTG')
    ax2.set_xticks(x)
    ax2.set_xticklabels(initial_start_codons)
    ax2.legend()
    ax2.set_ylabel('Frequency')
    ax2.set_title('Initial Start Codon vs Tandem Start Codons')

    # name formatting
    element = element.strip().split('.')
    element = element[0] + '.' + element[1]

    ax1.text(0.5, 1.15, element, transform=ax1.transAxes,
            ha='center', va='top', fontsize=12)

    # save the barchart
    plt.tight_layout()
    outfileName = ('bar_chart_' + element + '.png')
    directory_path = "C:/Users/Tommy/PycharmProjects/test1/Output/graphs/"
    out_filepath = os.path.join(directory_path, outfileName)
    plt.savefig(out_filepath)

    plt.savefig(out_filepath)
    plt.cla()
    plt.close(fig)
    print(element)

    # Show the chart

```

3.3.15. stopVariableInitialisation

This function is used to initialise the variables used for counting the stop codon data. This was created because it means that I don't have to have all of this code written in the already long

main section and makes the code appear much neater. The variables need to be initialised to zero at the beginning of the program because this project is meant to count frequencies, thus there wont be an initial number to add to if its not initialised to zero.

```
def stopVariableInitialisation():
    # total stop codon counting dict
    total_stop_codons_dict = {}
    initialStopCodon_count = 0
    tan_TAA_count = 0
    tan_TAG_count = 0
    tan_TGA_count = 0
    total_stop_codons_dict["TAA"] = (initialStopCodon_count, tan_TAA_count, tan_TAG_count, tan_TGA_count)
    total_stop_codons_dict["TAG"] = (initialStopCodon_count, tan_TAA_count, tan_TAG_count, tan_TGA_count)
    total_stop_codons_dict["TGA"] = (initialStopCodon_count, tan_TAA_count, tan_TAG_count, tan_TGA_count)
    return total_stop_codons_dict
```

3.3.16. startVariableInitialisation

This function is the same as the previous one except it is used for the start codon usage. Its implementation makes the code appear much neater.

```
def startVariableInitialisation():
    # total start codon counting dict
    total_start_codons_dict = {}
    initialStartCodon_count = 0
    tan_ATG_count = 0
    tan_GTG_count = 0
    tan_TTG_count = 0
    total_start_codons_dict["ATG"] = (initialStartCodon_count, tan_ATG_count, tan_GTG_count, tan_TTG_count)
    total_start_codons_dict["GTG"] = (initialStartCodon_count, tan_ATG_count, tan_GTG_count, tan_TTG_count)
    total_start_codons_dict["TTG"] = (initialStartCodon_count, tan_ATG_count, tan_GTG_count, tan_TTG_count)
    return total_start_codons_dict
```

3.3.17. Main

This is the main function; I will begin to explain what is occurring here so that the reader can better understand why this code is written.

```
if __name__ == '__main__':
    # main start
    #FFiles = globFileMatchUp()
    FFiles = glob('C:/Users/Tommy/PycharmProjects/ensembl_fungi_genomes/ensembl_fungi_genomes/' + '*.fa.gz')
    #FFiles = ["Candida dubliniensis_cd36_qcga_000026945.ASM2694v1.dna.toplevel.fa.gz"] # the fasta files provided in a list
    for element in FFiles:
        fastaName = os.path.basename(element)
        #print(element)
        #FDict = parse_fasta(os.path.join("C:/Users/Tommy/PycharmProjects/ensembl_fungi_genomes/ensembl_fungi_genomes/", element))
        FDict = parse_fasta(element)

        gffFilename = fastaName
        gffFilename = gffFilename.strip().split('.')
        gffFilename = gffFilename[0] + "." + gffFilename[1] + ".52.gff3.gz"
        #print(gffFilename)
        #gff_dict = parse_gff(os.path.join("C:/Users/Tommy/PycharmProjects/ensembl_fungi_gff/ensembl_fungi_gff/", gffFilename))

        gffPath = os.path.join('C:/Users/Tommy/PycharmProjects/ensembl_fungi_gff/ensembl_fungi_gff/', gffFilename)

        if os.path.exists(gffPath):
            gff_dict = parse_gff(gffPath)
            #print(element)
            #print(gffPath)
        else:
            print("match not found")
            print(gffPath)
            continue

    #gff_dict = parse_gff("Candida dubliniensis_cd36_qcga_000026945.ASM2694v1.52.gff3.gz")
```

Firstly, I use glob, to import all of the top down dna fasta file paths into a list called FFfiles. I then Start a for loop for each element in that list FFfiles, iterating through each file path. I create a string called fastaName and use os.basename(element) to get the file name of the fasta file path being currently being accessed. I then Call the parse_fasta function to create a dictionary containing the information from the parsed in fasta file path.

I then do some string manipulation to create the matching gff file Path, and use os.path.exists to check that the match exists in the gff file data directory. The code prints out “match not found” when it can’t find the corresponding gff to the fasta file and then continues through the for loop.

```
# gc content calculation
GcResults = {key: GcContent(value) for (key, value) in FDict.items()}

# variable initialisation
# Total start/stop codons
total_stop_codons = 0
total_start_codons = 0
# total stop codon counting dict
total_stop_codons_dict = stopVariableInitialisation()
total_start_codons_dict = startVariableInitialisation()

# file writing
filename = fastaName
if filename.endswith(".gz"):
    filename = filename[:-3]
outfileName = ("outfile_" + filename + ".txt")
directory_path = "C:/Users/Tommy/PycharmProjects/test1/Output/mainTextOutput/"
out_filepath = os.path.join(directory_path, outfileName)

with open(out_filepath, "w") as f:
    for gene in gff_dict:
        endSeq = get_End_Seq(gff_dict, FDict, gene)
        #print(endSeq)
        initialStopCodon = get_Initial_StopCodon(gff_dict, FDict, gene)
        #print(initialStopCodon)
        startSeq = get_start_Seq(gff_dict, FDict, gene)
        # print(startSeq)
        initialStartCodon = get_Initial_StartCodon(gff_dict, FDict, gene)
        #print(initialStartCodon)
        tandemStopCountData = tandemStopCount(endSeq, initialStopCodon, total_stop_codons, total_stop_codons_dict)
        # tandemstopcount == [0] = freq, [1] = indexes, [2] = total_stop_codons, [3] = total_stop_codons_dict
        tandemStartCountData = tandemStartCount(startSeq, initialStartCodon, total_start_codons, total_start_codons_dict)
```

The gc content is calculated using the GcContent function, and the total stop codon count, total start codon count, total stop codon dictionary and total start codon dictionary are initialised to zero using the stop variable initialisation function and start variable initialisation function.

This section is for preparing to write to the file. A string variable filename is initialised to the fastaName which is the basename of the of the current file path being iterated through in the for loop. This file name is then checked to see if it’s a .gz zipped and if it is it removes the .gz extension from the end of the filename. It then creates a new file name “outfileName” that has “outfile_” at the start, the genome name in the middle and “.txt” This outfileName is then added to the output file directory using os.path.join so that when the file is opened the it has the correct file name as in the correct location.

This section is where the file writing occurs, and where the start and stop codon data is retrieved using the functions. The file is opened using open in which is a file opening tool from python, the correct file name path is given and set in write mode “w”. A for loop is used to gather the information for each gene in the genome. The function endseq is called to obtain the end sequence where the tandem stop codons might be found, it takes in the gff dictionary, fasta file, dictionary and the current gene. The next function initial stop codon is then called to obtain the initial stop codon, the next function is called to obtain the start sequence where the start codons might be found as well as the initial start codon function these functions all

take in the same arguments the gff file dictionary, fasta file dictionary and the current gene from the for loop. The tandem Stop Codon count and tandem start codon count functions are called so that they can search the start sequence for start codons and the end sequence for stop codons and output the data to a dictionary that can be accessed later for creating the graph and the output text file with their frequencies initial codons and indexes. The comments in the lines below show what each index of each list is for reference.

```
tandemStartCountData = tandemStartCount(startSeq, initialStartCodon, total_start_codons, total_start_codons_dict)
# tandemstartcount == [0] = freq, [1] = indexes, [2] = total_start_codons, [3] = total_start_codons_dict

f = displayTandemStopCodon(tandemStopCountData[0], tandemStopCountData[1], gene, initialStopCodon, f)
f = displayTandemStartCodon(tandemStartCountData[0], tandemStartCountData[1], gene, initialStartCodon, f)

total_stop_codons = tandemStopCountData[2]
total_start_codons = tandemStartCountData[2]
total_stop_codons_dict = tandemStopCountData[3]
# total_stop_codons_dict == [0] = initialStopCodon_count, [1] = tan_TAA_count, [2] = tan_TAG_count, [3] = tan_TGA_count
total_start_codons_dict = tandemStartCountData[3]
# total_start_codons_dict == [0] = initialStartCodon_count, [1] = tan_ATG_count, [2] = tan_GTG_count, [3] = tan_TTG_count

f = displayFinalInfo(total_stop_codons, total_start_codons, GcResults, fastaName, f)
barchartCreate(filename, total_stop_codons_dict, total_start_codons_dict)
```

In this section the data is then written to the file by calling the display tandem stop codon function and the display start codon function and parsing in the variables. The total stop codon and total start codon counts are updated with the current count. Also the total stop codons dictionary and the total start codon dictionary is updated before the for loop ends. Once the for loop ends the final information is written to the file with the display final info function that parses is in the recently updated variables. Finally, The bar chart is then created with the bar chart create function parsing in the total start and stop codon data dictionaries. This ends the file genome loop.

3.4. Input

There are 4 input directory's which are located in my pycharm projects folder not in the actual projects folder this is so when I was uploading my project to the git repository my compressing the file I could cut down on a large amount of the time it would take to upload also I could keep the directory location the same for all other project files no matter what happened to those project files.

Tommy > PycharmProjects >

	Name	Date modified	Type
★	CompleteFinalProject	04/05/2023 15:32	File folder
★	Ensembl_bacteria_genomes	04/05/2023 15:46	File folder
★	Ensembl_bacteria_gff	04/05/2023 15:46	File folder
★	ensembl_fungi_genomes	09/04/2023 00:35	File folder
★	ensembl_fungi_gff	09/04/2023 00:08	File folder
★	FinalProject	21/03/2023 18:19	File folder
★	test1	02/05/2023 19:09	File folder

3.5. Output

graphs	04/05/2023 15:34	File folder
mainTextOutput	04/05/2023 15:34	File folder
pro_graphs	04/05/2023 15:34	File folder
pro_mainTextOutput	04/05/2023 15:35	File folder

The output directory looks like this with 4 files. Graphs and main text output are for the eukaryotic genome output and pro graphs and pro main text output are for the prokaryotic genome output.

Below is an extract of the output from this program, The program outputs 2 files, firstly a text file that displays the tandem stop codons and tandem start codons for each gene in the genome with their initial stop and start codon as well as their indexes in the sequence. Secondly a .png file is output which is a bar chart graph containing the frequencies of each of the initial stop and start codons and their tandem stop and start codons with the genome as the bar chart graphs title.

Text File output:

```
outfile_candida_arabinofermentans_nr1_yb_2248_gca_001661425.Canar1.dna
File Edit Format View Help

Stop codon frequencies: ID=gene:CANARDRAFT_4957:
The initial stop codon was: TTC
TAA : 0 at indexes []

TAG : 0 at indexes []

TGA : 0 at indexes []

Start codon frequencies: ID=gene:CANARDRAFT_4957:
The initial start codon was: TTA
ATG : 0 at indexes []

GTG : 0 at indexes []

TTG : 0 at indexes []

Stop codon frequencies: ID=gene:CANARDRAFT_173570:
The initial stop codon was: TTG
TAA : 0 at indexes []

TAG : 0 at indexes []

TGA : 1 at indexes [0]

Start codon frequencies: ID=gene:CANARDRAFT_173570:
The initial start codon was: TTA
ATG : 0 at indexes []

GTG : 0 at indexes []

TTG : 0 at indexes []

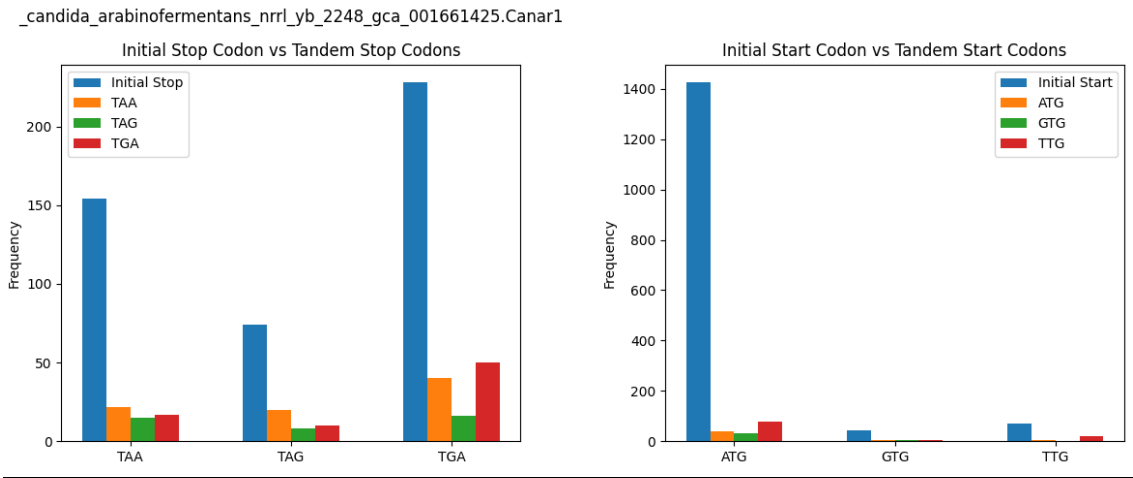
Stop codon frequencies: ID=gene:CANARDRAFT_25884:
The initial stop codon was: CAA
TAA : 0 at indexes []

TAG : 0 at indexes []

TGA : 0 at indexes []

Start codon frequencies: ID=gene:CANARDRAFT_25884:
```

Bar chart graph:



4. Implementation

4.1. Introduction

In this Section I will discussing the implementation process of my project and the issues I encountered and technical issues encountered during the project as well as the sections of the project that I developed at various times.

4.2. Issues encountered.

4.2.1. Initial Changes of requirements in this Project

Initially my project was “To use Machine learning to predict the coding and non-coding regions in DNA” However Wayne wanted to me to instead Look at tandem stop codons thus my project changed to “Investigating the frequency of tandem stop codons in eukaryotic & prokaryotic genomes” My requirements also changed later on after the mid project demo to include the genomes start codons and look for their tandem start codons, this wasn’t much of an issue however it was a change of requirements that did stagger my development so I feel it is worth mentioning.

4.2.2. Opening the files

Initially I had trouble opening the files provided to me from Wayne, this is because the files were .gz zipped and windows can not open these. So, I had to install software to unzip these .gz zipped “WINZIP” files so that they could be viewed. However, Wayne provided me with a code sample about a line long which opens .gz files in python, this was helpful for this project. Towards the end of the project I found a way of using my git terminal which I had used for my 2nd year group project and the code uploads for this project to un zip the files on that command line.

4.2.3. Wrong Fasta File provided.

Around the mid project, it was discovered that Wayne had provided me with CDS fasta files on the shared OneDrive. CDS fasta files don’t not contain any non coding sequences of nucleotides thus would not contain any tandem stop codons or even start codons for this matter. Wayne however, was able to quickly provide me with the correct fasta files “Top down dna fasta files” and I quickly got to work changing my parsing fasta file into a python dictionary function to work on these new files, so the tandem stop codons and later tandem start codons could be indexed and obtained.

4.2.4. Problems with Synder python ide

Early in the project I decided to use the synder python integrated development environment since I used it for the cs31820 computation bioinformatics module. However, during development of this project I ran into multiple problems using the ide, It would frequently crash, freeze up and in general was slow to run the code I had developed. This was Wayne Prompted be to switch to Pycharm a JetBrains python ide, as soon as I used this ide my issues went away.

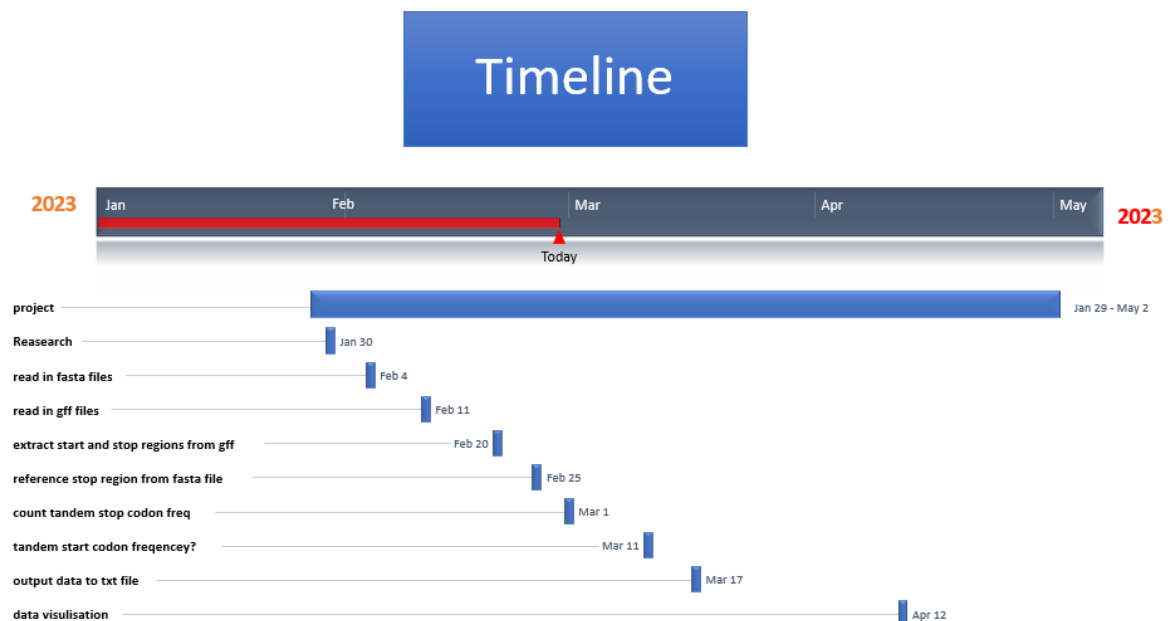
4.2.5. Prokarotyoc genomes?

My project is to “Investigating the frequency of tandem stop codons in eukaryotic & prokaryotic genomes”. However, the file that Wayne proved to me are ensembl Fungi

Genomes and fungi is Eukaryotic. These means that the data I was provided with is only Eukaryotic and there isn't any Prokaryotic genomes to run my code on. As soon as I'm provided with these files I should be able to run the code on them and write about the results. However, Wayne was able to provide me with a shared one drive directory of bacteria files from ensemble Unfortunatly because of the directory format being different to that of the fungi shared one drive I would have to download each individual top down dna fasta file and matching gff file by hand, thus I wouldn't be able to create a large output to analyse.

4.3. Time specific project developments

Early Timeline for this project during mid project demo:



30th jan - 6th of feb : Research and discussion

During this period of time I spent most of my time researching machine learning techniques and uses for computational biology and bioinformatics, As well as standard bioinformatics theory.

6th of feb – 13th of feb : Reading in Fasta files and gc content

During this period of time I spent my time developing code to read in Fasta file into a dictionary and using a gc content function I developed for the cs31820 computational bioinformatics module to calculate the gc content of the genome. This is also when my project changed to not using machine learning anymore and to look at tandem stop codons.

13th of feb – 20th of feb : Gff files

During this period of time I worked on developing a method of reading the gff files data line by line into a dictionary that I could access later.

20th of feb – 27th of feb : indexing the fasta files

During this period I developed a method of indexing the fasta files, from the gff file stop region data to obtain the end region where the stop codons might appear

27th of feb – 6th of mar : analysing the stop region

At this point I need a way to find the analyse a string containing the tandem stop codons, so I created a function to do so that took in the string end sequence and find the in frame tandem stop codons TAA, TAG, TGA.

6th of mar – 13th of mar : text file out put

At this point I realised that I should probably out put the data I've gathered from these fasta files and gff files, so I decided to write the data to a text file that I could would have for each gene the found tandem stop codons, their indexes, their initial stop codon, the gc content and the total tandem stop codons found.

13th of mar – 20th of mar : mid project demo and some new developments?

At this point in the project I found out that I had been given the wrong fasta files they were cds fasta file not top down dna fasta files thus I need to remodel my code. I also created slides for a presentation the mid project demonstration.

20th of mar – 27th of mar : Remodelling

During this period I remodelled the project into functions as previously it was one continuous main method, and fixed the fasta file reading problems for the new top down dna fasta files.

27th of mar – 3rd of April : Start codons?

I was then tasked by Wayne not only look for tandem stop codons but to also look for tandem start codons, so I developed functions to look for the tandem start codons within a start sequence and their initial start codon, and out put that data to the text file.

3rd of April – 10th of April : bar char graphing

During this period, I finalised the creation of the Bar chart graph that displays the tandem stop codons and their initial stop codon, as well as the tandem stop codons and their initial start codon.

10th of April – 17th of April : file matching and glob

During this period I developed and tested the method of using glob to import the file paths and match them up correctly to the gff files, without processing any files that didn't have any matching files.

17th of April – 5th of May : report

During this period of time I spent most of my time developing and finishing up the report for this project as well as some minor tweaks to the technical code.

4th of May: some prokaryotic genomes received

During the last day of my project, I spent my time finishing up the project report and updating the code on my project to run on the few prokaryotic genomes I was able to obtain that day.

4.4. Minor last-minute Changes

I moved the parse_fasta until after the potential gff file is confirmed to exist in the gff file directory, this is so the program doesn't spend time writing the fasta file into a dictionary when it might not actually use it.


```
FFiles, gffFiles = globFileMatch(nupl)
FFiles = glob('C:/Users/Tommy/PycharmProjects/ensembl_fungi_genomes/ensembl_fungi_genomes/' + '*.fa.gz')
#FFiles = ["Candida_dubliniensis_cd36_qca_000026945.ASM2694v1.dna.toplevel.fa.gz"] # the fasta files provided in a list
for element in FFiles:
    fastaName = os.path.basename(element)
    #FDict = parse_fasta(element) moved to after gff is checked to increased speed

    gffFilename = os.path.basename(element)
    gffFilename = gffFilename.strip().split('.')
    gffFilename = gffFilename[0] + "." + gffFilename[1] + ".52.gff3.gz"

    gffPath = os.path.join('C:/Users/Tommy/PycharmProjects/ensembl_fungi_gff/ensembl_fungi_gff/', gffFilename)

    if os.path.exists(gffPath):
        FDict = parse_fasta(element) # put fasta file into a dictionary
        gff_dict = parse_gff(gffPath) # puts gff into a dictionary
        #print(element)
        #print(gffPath)
    else:
        print("match not found")
        print(gffPath)
        continue
```

5. Testing

5.1. Introduction and Strategy

This is where I will discuss testing what strategy I used for this project, and some examples of the types of testing that was done throughout the process so it could potentially be replicated by the reader. This is a bioinformatics project written to run on one python file with many functions so the strategy was to use unit testing then integration testing and finally Systems testing.

5.2. Overall Approach to Testing

When testing for my project I would mainly use unit testing, when I needed to write a section of code usually a function, I would create a new python file and write, develop and test the function in that file then add it to a copy of my working main python project file where I would use integration/system testing to test that the new function is working and correctly integrated.

As this is a bioinformatics research project I didn't utilise, Security testing or user acceptance testing, since the data in this project is not sensitive and doesn't have any users.

5.3. Unit Testing

These unit tests are written in separate python files within the project, it's important to note that I used print statements to test the output as well as the debug tool, these are some examples of the files I wrote. It is also important to note that these are the files are unchanged from the moment I finished them and their actual counterparts in the main code may be different due to integration testing and perhaps changes in requirements later down the line.

5.3.1. Unit Tests table

Test	Expected	Pass/fail
Bar chart display	Bar chart created with test data for stop	pass
Bar chart display	Bar chart created with test data for start	pass
Bar chart display	Bar chart created with test data for both stop and start	pass
Parse gff	The elements of the gff dictionary will be displayed to the terminal screen	pass
Parse fasta	The elements of the top down dna fasta file dictionary will be displayed to the terminal screen	pass

Slice notation test	The string will be sliced together with the selected indexed sections	pass
File writing from within in function	Out put a text file containing the information written to the file object by functions	pass
Codon counting test	The in frame stop codons from the fabricated input string will be detected and out put to the screen along with their indexes in the string	pass
Using glob to match up files test	The list of all fasta files and gff files that corresponding matches will be printed to the display terminal. And the Glob module will be used to import the paths for these files from their directory	pass

5.3.2. Screenshots

Bar-chart test: start and stop

This test was created so that I could see how the matplotlib bar chart might look when used with the real data later. You can see I used fake data there in comparison to its main code counterpart simply for testing.

```

en.py × Defensive.py × globfilematchup.py × slicenotationtest.py × StartcodonTest.py × StartStopBarChart.py × fileWritingtest.py ×
import matplotlib.pyplot as plt

# Data for stop codon bar chart
initial_stop_codons = ['TAA', 'TAG', 'TGA']
initial_counts = [100, 75, 50] # to be replaced with data
tandem_TAA_counts = [80, 60, 40] # to be replaced with data
tandem_TAG_counts = [80, 60, 40] # to be replaced with data
tandem_TGA_counts = [80, 60, 40] # to be replaced with data

# Data for start codon bar chart
initial_start_codons = ['ATG', 'GTG', 'TTG']
initial_start_counts = [120, 90, 60] # to be replaced with data
tandem_ATG_counts = [100, 70, 50] # to be replaced with data
tandem_GTG_counts = [100, 70, 50] # to be replaced with data
tandem_TTG_counts = [100, 70, 50] # to be replaced with data

# Bar chart setup
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5)) # Create subplots for stop codon and start codon bar charts
width = 0.15 # the width of the bars
x = [i for i in range(len(initial_stop_codons))] # the x locations for the groups of bars

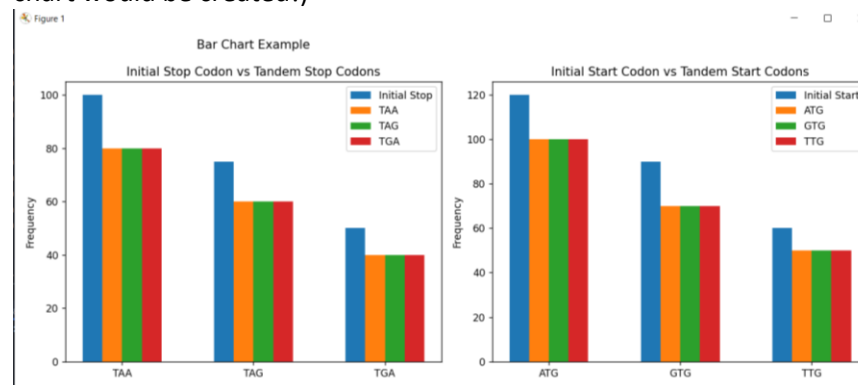
# Stop codon bar chart
# Plot the initial stop codon counts TAA
bar1 = ax1.bar([i - 1.5*width for i in x], initial_counts, width, label='Initial Stop')

# Plot the tandem stop codon counts TAG
bar2 = ax1.bar([i - 0.5*width for i in x], tandem_TAA_counts, width, label='TAA')

# Plot the tandem stop codon counts TGA
bar3 = ax1.bar([i + 0.5*width for i in x], tandem_TAG_counts, width, label='TAG')

```

This is the output: (Note, this is not the actual data from the files but test data to see if the chart would be created!)



Gff file reading and dictionary creation testing.

This was used to test what the gff file reading and dictionary would output, as well me using this file to develop a working test.

```
def parse_gff(gff_file):
    gff_dict = {}
    f = openfile(gff_file)
    for line in f:
        if line.startswith('#'):
            continue
        fields = line.strip().split('\t')
        if fields[2] == 'gene':
            gene_id = fields[8].split(';')[0]
            start = int(fields[3])
            stop = int(fields[4])
            chromosome = fields[0]
            gff_dict[gene_id] = (start, stop, chromosome)
    return gff_dict # Press Ctrl+F8 to toggle the breakpoint.
```

Main:

```
if __name__ == '__main__':
    gff_dict = parse_gff("Candida_dubliniensis_cd36_gca_000026945.ASM2694v1.52.gff3.gz")
    print(gff_dict)
```

Output:

```
'ID=gene:CD36_35285': (2184102, 2184953, 'R'), 'ID=gene:CD36_35290': (2185073, 2186086, 'R'), 'ID=gene:CD36_35295': (2187930, 2187930, 'R'), 'ID=gene:CD36_35296': (2187969, 2188799, 'R'), 'ID=gene:CD36_35300': (2189723, 2191573, 'R'), 'ID=gene:CD36_35310': (2192812, 2195499, 'R'), 'ID=gene:CD36_35320': (2195506, 2196492, 'R'), 'ID=gene:CD36_35330': (2199599, 2199599, 'R'), 'ID=gene:CD36_35334': (2200034, 2200663, 'R'), 'ID=gene:CD36_35340': (2200746, 2201636, 'R'), 'ID=gene:CD36_35355': (2205186, 2205896, 'R'), 'ID=gene:CD36_35360': (2206123, 2207613, 'R'), 'ID=gene:CD36_35370': (2212082, 2212082, 'R'), 'ID=gene:CD36_35390': (2212454, 2213716, 'R'), 'ID=gene:CD36_35400': (2213793, 2216186, 'R'), 'ID=gene:CD36_35415': (2217290, 2217697, 'R'), 'ID=gene:CD36_35420': (2217760, 2219757, 'R'), 'ID=gene:CD36_35430': (2222335, 2222335, 'R'), 'ID=gene:CD36_35450': (2222417, 2223553, 'R'), 'ID=gene:CD36_35460': (2223866, 2224882, 'R'), 'ID=gene:CD36_35480': (2226758, 2231965, 'R'), 'ID=gene:CD36_35485': (2232048, 2232779, 'R'), 'ID=gene:CD36_35490': (2235415, 2235415, 'R'), 'ID=gene:CD36_35510': (2235535, 2237448, 'R'), 'ID=gene:CD36_35520': (2237432, 2238478, 'R'), 'ID=gene:CD36_35524': (2239299, 2239703, 'R'), 'ID=gene:CD36_35526': (2239831, 2240217, 'R'), 'ID=gene:CD36_35530': (2240217, 2240217, 'R')
```

Fasta file reading and dictionary creation testing.

This was used to test what the fasta file reading and dictionary would output, as well me using this file to develop a working test.

```
import gzip

def openfile(filePath): # opening and reading the fastafile
    if filePath.endswith(".gz"):
        with gzip.open(filePath, 'rt') as f:
            return [l.strip() for l in f.readlines()]
    else:
        with open(filePath, 'r') as f:
            return [l.strip() for l in f.readlines()]

def parse_fasta(filename):
    FastaFile = openfile(filename)
    FDict = {}
    chromosomeID = ""
    nucleotide_sequence = ""
    for line in FastaFile:
        line = line.strip()
        if line.startswith(">"):
            if chromosomeID != "":
                FDict[chromosomeID] = nucleotide_sequence
                nucleotide_sequence = ""
            chromosomeID = line.strip().split(' ')[2].split(':')[2]
        else:
            nucleotide_sequence += line
    # adds the last sequence to the dictionary because there isn't a > after the last line
    FDict[chromosomeID] = nucleotide_sequence
    return FDict
```

Main:

```
if __name__ == '__main__':
    gff_dict = parse_gff("Candida_dubliniensis_cd36_gca_000026945.ASM2694v1.52.gff3.gz")
    print(gff_dict)
    FDict = parse_fasta("Candida_dubliniensis_cd36_gca_000026945.ASM2694v1.dna.toplevel.fa.gz")
    print(FDict)
```

Output:

```
ATTAAATCTAAGGCAAGTGACAAAGACGGGACATAAACAAGTTGATCAATTTAACACAAATGGCAAGAACAAAGTAATCTAAG
TTTAGGACAAGTTGGCTAGCCTGGGACATGTCATGACAAATATGGGACAAAACGTGACAAAAACAAATTTATCAATCAAGA
CTGGGTAACACAGACATGTCACGACAAGGATTAGACAAGGACATGTGAATGGGCCTGGGGGTTGTTAGGACGTACGGATGC
CTATAGGTAATCGAAGGTCAAGAATTGGGAGAAGGAGTAGTAGGGGAATGGCGACACGTACGGATGCATTAGTGGTGGCAA
TGAAAAATTTTGAAGTGGCGAAAAAGAGAGGTGAAGATATTGTATGTGCGTGTAGTTAGGTGACACAAGTTTGAGTTGGGA
GACACAAGTTTTTTTTTATGACTTTTTTGGTGC GCGACAGAAAGATAGTTGAAACAGAGTTTTTTTTTCTATGGTTTTTTT
CTATGGTTTTTCTTAGTGC GCGGAAAAAGTGGTGAATTGATAGTTTTTTTCTATGACTTTTTGTTGCGCGCGGAAAAAGT
CGTCGAAAAGTGGTGAATTGAAGTTTTTTTTTCTATGGTTTTTTTGTGTGCGAGGAAAAGTGGTGAACAGAGTTTTTTT
TAGAAATTTTTTCTATGACTTTTTGTCAGCGCGCAGAAAAGTGGTGATTGCAAGTTTTTTTCTATGGTTTTTTTTGTC
ATTTTTTAAAGTGC GTGAAAAAGTGGTGAACGAAGTTTTTTTTTCTATGACTTTTTCTTAGTGC GCGGAAAAAGTGGTGAG
AAAGTGGTGAACGAAGTTTTTTTTTCTATGGCTTTTTGTTGTGCGCGGAAAAAGTGGTGAAGAGAAAGATTTTTTCTAT
```

Slice notation test

This file was created so I could test how to use slice notation in python. Slice notation very simple to use in python however, since using slice notation resulted in an continuous error at the beginning of the project, it needed further work to make sure it could fully work without an mistakes later on.

```

if __name__ == '__main__':
    stopRegion = 15
    fullseq = "ATGCTACGTACTGAATAGTAGTAATAA"
    gene = "stuff"
    FDict = {gene: (fullseq)}
    #endseq = (fullseq[stopRegion+3:stopRegion+12])
    #print("endseq = " + endseq)
    endseq_dict = (FDict[gene][stopRegion+3:stopRegion+12])
    print("endseq_dict = " + endseq_dict)

```

File writing test

This file was created so I could test how to write to a text file and output it using functions in python.

```

def display1(f):
    print("hi")
    f.write("hi\n")
    return f

def display2(f):
    print("bye")
    f.write("bye\n")
    return f

if __name__ == '__main__':
    outfileName = ("outfile_")
    list = [1,2]
    with open(outfileName, "w") as f:
        for element in list:
            f = display1(f)
            f = display2(f)

```

Output

The output was a text file with “hi hi bye” and this was displayed on the output terminal as well. But clearly showed that it was possible to be to write to a text file whilst using separate functions.

Codon counting test

This unit test was created so that I could test how to count the codons and their indexes then output them (only to the terminal screen). Given my known input “my_list”

```
my_list = "AAATTGTATAGG"
stop_codons = ["TAA", "TAG", "TGA"]
freq = {codon: 0 for codon in stop_codons}
indexes = {codon: [] for codon in stop_codons}
total_codons = 0

for i in range(0, len(my_list)-2, 3):
    codon = my_list[i:i+3]
    if codon in stop_codons:
        if (i+3) % 3 == 0:
            freq[codon] += 1
            indexes[codon].append(i)
        total_codons += 1

print("Stop codon frequencies:")
for codon, count in freq.items():
    print(codon, ":", count, "at indexes", indexes[codon])
print("Total codons found at indexes that are not 0:", total_codons - freq["TAA"] - freq["TAG"] - freq["TGA"])

total_codons = 0
```

Output

This output was what was expected however this format would later be changed for the main project file.

```
C:\Users\Tommy\PycharmProjects\FinalProject\venv\
Stop codon frequencies:
TAA : 0 at indexes []
TAG : 1 at indexes [9]
TGA : 0 at indexes []
Total codons found at indexes that are not 0: 0
```

Glob file match up test

This unit test was for a function that could take the files and match the fasta-gff so that there weren't any unmatched files. This function was later made redundant, and changes were made to my main code so that, the file match up system was better refined, however these tests were a necessary part of the development process.

```

n.py x  Defensive.py x  globfilematchup.py x  slicenotationtest.py x  St
from glob import glob

globFastaFiles = glob('C:/Users/Tommy/PycharmProjects/ensembl_fa
globGffFiles = glob('C:/Users/Tommy/PycharmProjects/ensembl_fun
FastaFiles = []
GffFiles = []

for element in globFastaFiles:
    fastaFilename = element.strip().split('\\')[1].split('.')
    fastaFilename = fastaFilename[0] + fastaFilename[1]
    FastaFiles.append(fastaFilename)
for element in globGffFiles:
    gffFilename = element.strip().split('\\')[1].split('.')
    gffFilename = gffFilename[0] + gffFilename[1]
    GffFiles.append(gffFilename)

FFiles = []
GFiles = []
for element in FastaFiles:
    if GffFiles.count(element):
        FFiles.append(element + ".dna.toplevel.fa.gz")
        GFiles.append(element + ".52.gff3.gz")

for element in FFiles:
    print(element)
for element in GFiles:
    print(element)

```

Output

This output was the list of files that had gff-fasta file matches so that there weren't any unmatched files.

```

_candida_glabrata_gca_001466565ASM146656v1.52.gff3.gz
_candida_glabrata_gca_001466575ASM146657v1.52.gff3.gz
_candida_glabrata_gca_001466635ASM146663v1.52.gff3.gz
_candida_glabrata_gca_001466685ASM146668v1.52.gff3.gz
_candida_glabrata_gca_002219185ASM221918v1.52.gff3.gz
_candida_glabrata_gca_002219195ASM221919v1.52.gff3.gz
_candida_glabrata_gca_010111755ASM1011175v1.52.gff3.gz
_candida_glabrata_gca_014217725ASM1421772v1.52.gff3.gz
_candida_inconspicua_gca_004931855ASM493185v1.52.gff3.gz
_candida_intermedia_gca_900106115CBS_141442_assembly.52.gff3.gz
_candida_intermedia_gca_900106125PYCC_4715_assembly.52.gff3.gz

```

5.4. Integration and System Testing

5.4.1. Integration and System Tests table

Start/stop region tests

Codons to look for	Region looking for	Are there many of these codons found (yes/no)
Initial start codon	Start region -3 : start region	no
Initial start codon	Start region : start region +3	no
Initial start codon	Start region -1 : start region +2	yes
Initial stop codon	Stop region -2 : stop region +1	no

Initial stop codon	Stop region -3 : stop region	no
Initial stop codon	Stop region : stop region +3	no
Initial stop codon	Stop region -1 : stop region +2	yes

System test table

test	Expected result	Pass or fail
Single Text file output to text file output directory	The file is there with the correct name and data	Pass
Single Graph output to output graph directory	The file is there with the correct name and data	pass
All the files are output to the correct output directory using function glob file match up	All the file matches are output to the correct directory and contain the correct data	fail
All the files are output to the correct output directory using removed function glob file match up. And used os module within main	All the file matches are output to the correct directory and contain the correct data	fail
All the files are output to the correct output directory using removed function glob file match up. And used os module within main. Changed os.path manipulation	All the file matches are output to the correct directory and contain the correct data	pass
Non file matches are detected and dealt with accordingly	Non file matches between fasta and gff files are not read through into the program and are skipped in the for loop. There should also not be any false output files. The non matches should be printed to the display terminal with "no match"	pass
Test program on a small section of prokaryotic genomes	All the small section of files selected matches are output to the correct directory and contain the correct data	pass
All the prokaryotic genome files are output to the correct output directory using removed function glob file match up. And used os module within main. Changed os.path manipulation	For all prokaryotic genomes Non file matches between fasta and gff files are not read through into the program and are skipped in the for loop. There should also not be any false output files. The non matches should be printed to the	Fail, Prokarotic gemomes shared directory format meant I couldn't download all of the data

	display terminal with “no match”	
--	----------------------------------	--

5.4.2. Integration and System Testing Screenshots

Start region test

This python file was created so I code test obtaining the initial start codon, and the sequence where the tandem start codons might appear. It must be noted that I look for the tandem starts before the initial start codon. I am not looking after the initial start codon. This was created so that I could confirm I was obtaining the correct region.

```
en.py x Defensive.py x globfilematchup.py x slicenotationtest.py x StartcodonTest.py x StartStopBarChart.py x
return gff_dict # Press CTRL+F8 to toggle the breakpoint.

def get_start_seq(gff_dict, FDict, gene):
    startRegion = gff_dict[gene][0]
    chromosome = gff_dict[gene][2]
    startSeq = (FDict[chromosome][startRegion - 13:startRegion - 1])_# front end
    # startSeq = (FDict[chromosome][startRegion + 2:startRegion + 14]) # back end
    return startSeq

def get_initial_startcodon(gff_dict, FDict, gene):
    startRegion = gff_dict[gene][0]
    chromosome = gff_dict[gene][2]
    initialStartCodon = (FDict[chromosome][startRegion:startRegion + 3]) # stop codons ??
    #initialStartCodon = (FDict[chromosome][startRegion - 6:startRegion + 6]) # big search
    initialStartCodon = (FDict[chromosome][startRegion - 1:startRegion + 2])_# start codons ???
    return initialStartCodon

if __name__ == '__main__':
    gff_dict = parse_gff("Candida_dubliniensis_cd36_gca_000026945.ASM2694v1.52.gff3.gz")
    #print(gff_dict)
    FDict = parse_fasta("Candida_dubliniensis_cd36_gca_000026945.ASM2694v1.dna.toplevel.fa.gz")
    #print(FDict)
    for gene in gff_dict:
        startSeq = get_start_seq(gff_dict, FDict, gene)
        #print(startSeq)
        initialStartCodon = get_initial_startcodon(gff_dict, FDict, gene)
        print(initialStartCodon)
        # print(startSeq + " " + initialStartCodon)
        # print(get_fullseq(gff_dict, FDict, gene))
```

I would print the results of the functions. Firstly, I used the initial start codon function to look for these, then I could use the get_start_seq function to look for the start region. This is an example of integration testing because implement these functions together on top of the, parse fasta file and gff parse file.

Output example

Initial start codons

ATG	CTA
ATG	ATG
TTA	TTA
TCA	TTA
TTA	ATG
ATG	TTA
ATG	ATG
ATG	CTA
ATG	TTA
ATG	ATG
ATG	TTA
ATG	TTA
ATG	TTA
TTA	TCA
ATG	CTA

The start sequence

CGCTCAAGTTCA	GTATATTAATAA
ACCTACTAACCA	AAGAATAACGTT
ACAAGAAAACTT	ATAGTGCGTGTA
TTTTTTTTTTTT	ACACACACATAC
TTTTTGACGCGA	TGTAAGTCAAAA
AAAAATTTTGGG	ACAAGGTGGTGT
CCAACCTCAACC	TAACCTTAGAAG
CTCAATTCAACC	AAACGATACAAA
AGCAATAACACG	AGTACCAGCTTC
TCTTTGCTAACC	CTTTACACAACA
TAACCATTTGCC	CTGTTATATACG
TCTAGAGAAATA	CAACTACATAAC
TAGTTACAGACC	AGTACCCTAACC
GATATGTGACCT	CACACTACCCTA
	TGTTTCATGTCTA

Stop region test

Similarly, to the start and region test, this python file was created so that I could test the end regions obtained from the gff and where this matched up to in the fasta file. I used print statements to display this information and adjusted the regions using slice notation ever so slightly to find these regions based of where many stop codons where found.

```

def get_End_Seq(gff_dict, FDict, gene):
    stopRegion = gff_dict[gene][1]
    chromosome = gff_dict[gene][2]
    endSeq = (FDict[chromosome][stopRegion + 2:stopRegion + 14])
    return endSeq

def get_Initial_StopCodon(gff_dict, FDict, gene):
    stopRegion = gff_dict[gene][1]
    chromosome = gff_dict[gene][2]
    initialStopCodon = (FDict[chromosome][stopRegion - 1:stopRegion+2])
    return initialStopCodon

if __name__ == '__main__':
    gff_dict = parse_gff("Candida_dubliniensis_cd36_gca_000026945.ASM2694v1.52.gff3.gz")
    FDict = parse_fasta("Candida_dubliniensis_cd36_gca_000026945.ASM2694v1.dna.toplevel.fa.gz")
    for gene in gff_dict:
        endSeq = get_End_Seq(gff_dict, FDict, gene)
        initialStopCodon = get_Initial_StopCodon(gff_dict, FDict, gene)
        print(endSeq)
        print(initialStopCodon)

```

Initial Stop codons

TGG	TAT
TAT	TTG
GTA	AAA
GAG	TAA
AGT	GTA
GTT	TTG
AAG	TGG
GGC	ATA
GGT	TGG
ATG	TGA
TAG	TGG
GAT	TGG
TTG	TGG
GTG	TTT

End Sequence

CATGTAGGCGCT	TTAAGATAGAGT
TTTCAATTGTTC	AGAAAAAAAAT
GTTTTGTATAGA	TATATTGATTGG
TTTAGTATATGC	AATGTACATTTT
TTAAGATTAAAGT	CTTTTATATTGA
AATAAAGACATG	ATCATCCTTCAT
CTGTGACAGGTT	TATAGATGTGTG
CGTGATTATGTA	ACAATATATAGT
TCTTGGTTATTG	AATCGTTTCGGG
CTTGGATGAATT	TTTGTTCGCAA
CAAGGTGGTGT	TTGGTGTGGAAT
TTGCTATGTAGT	TGCAATGCGCTT
GATTTAAATATA	TTTGTATTTT
CAGAGACTGGTA	ATATATAGGTCG
TGAAGAAAGTGG	CCTCTTTTTTTA

6. Results and Conclusions



6.1. Introduction

In this section I will discuss the results produced from by my project and the conclusion that can be drawn from these results. I will display the results output graphs and text files, the data collected and what they could possibly mean in response to by initial hypothesis for this project.

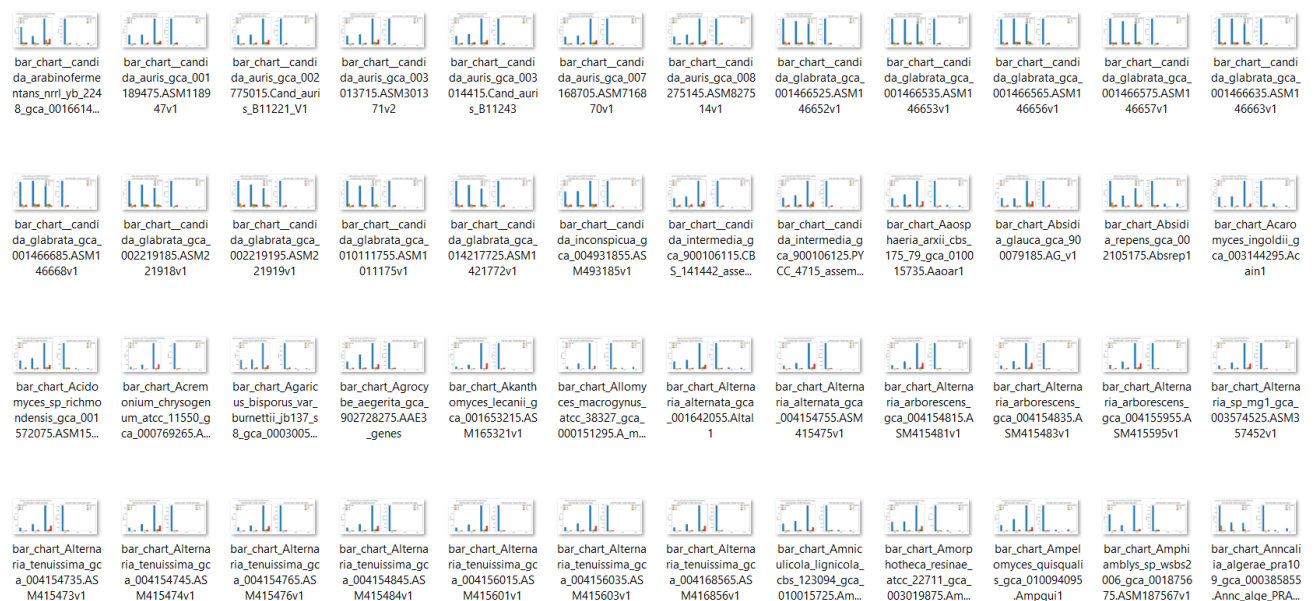
6.2. Eukaryotic genomes Results

The result data was output to an output directory splitting the graph files and text files up.

Tommy > PycharmProjects > test1 > Output

Name	Date modified	Type	Size
 graphs	28/04/2023 20:44	File folder	
 mainTextOutput	28/04/2023 20:44	File folder	

Graphs folder



Text file output

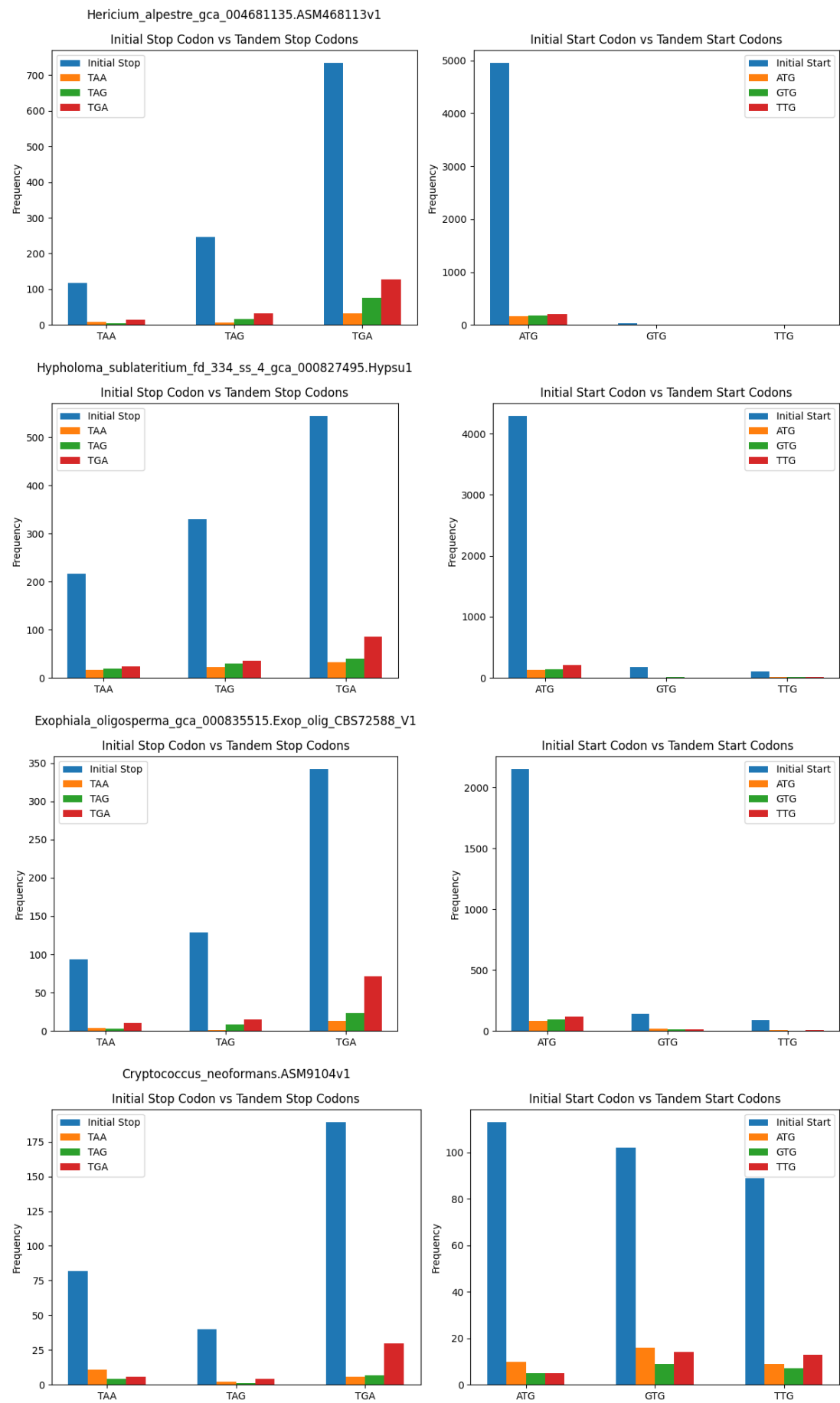
Name	Date modified	Type	Size
outfile_candida_arabinofermentans_nr1...	28/04/2023 20:43	Text Document	1,827 KB
outfile_candida_auris_gca_001189475.A...	28/04/2023 20:43	Text Document	2,248 KB
outfile_candida_auris_gca_002775015.Ca...	28/04/2023 20:43	Text Document	1,685 KB
outfile_candida_auris_gca_003013715.A...	28/04/2023 20:43	Text Document	1,637 KB
outfile_candida_auris_gca_003014415.Ca...	28/04/2023 20:43	Text Document	1,660 KB
outfile_candida_auris_gca_007168705.A...	28/04/2023 20:43	Text Document	1,635 KB
outfile_candida_auris_gca_008275145.A...	28/04/2023 20:43	Text Document	1,681 KB
outfile_candida_glabrata_gca_00146652...	28/04/2023 20:43	Text Document	1,592 KB
outfile_candida_glabrata_gca_00146653...	28/04/2023 20:44	Text Document	1,584 KB
outfile_candida_glabrata_gca_00146656...	28/04/2023 20:44	Text Document	1,627 KB
outfile_candida_glabrata_gca_00146657...	28/04/2023 20:44	Text Document	1,616 KB
outfile_candida_glabrata_gca_00146663...	28/04/2023 20:44	Text Document	1,827 KB
outfile_candida_glabrata_gca_00146668...	28/04/2023 20:44	Text Document	1,623 KB
outfile_candida_glabrata_gca_00221918...	28/04/2023 20:44	Text Document	1,625 KB
outfile_candida_glabrata_gca_00221919...	28/04/2023 20:44	Text Document	1,626 KB
outfile_candida_glabrata_gca_01011175...	28/04/2023 20:44	Text Document	1,605 KB
outfile_candida_glabrata_gca_01421772...	28/04/2023 20:44	Text Document	1,593 KB
outfile_candida_inconspicua_gca_00493...	28/04/2023 20:44	Text Document	1,552 KB
outfile_candida_intermedia_gca_900106...	28/04/2023 20:44	Text Document	2,019 KB
outfile_candida_intermedia_gca_900106...	28/04/2023 20:44	Text Document	2,060 KB
outfile_Aaosphaeria_anxii_cbs_175_79_gc...	02/05/2023 19:09	Text Document	4,442 KB
outfile_Absidia_glauca_gca_900079185.A...	02/05/2023 19:09	Text Document	4,429 KB
outfile_Absidia_repens_gca_002105175.A...	02/05/2023 19:09	Text Document	4,695 KB
outfile_Acaromyces_ingoldii_gca_003144...	02/05/2023 19:09	Text Document	2,510 KB
outfile_Acidomyces_sp_richmondensis_gc...	02/05/2023 19:09	Text Document	3,225 KB
outfile_Acremonium_chrysogenum_atcc_...	02/05/2023 19:09	Text Document	2,698 KB
outfile_Agaricus_bisporus_var_burnettii_j...	28/04/2023 19:38	Text Document	3,562 KB
outfile_Agrocybe_aegerita_gca_9027282...	28/04/2023 19:38	Text Document	4,130 KB
outfile_Akanthomyces_jeanii_gca_00165...	28/04/2023 19:38	Text Document	2,464 KB
outfile_Allomyces_macrozynus_atcc_383...	28/04/2023 19:39	Text Document	5,652 KB
outfile_Alternaria_alternata_gca_0016420...	28/04/2023 19:39	Text Document	4,227 KB

662 items

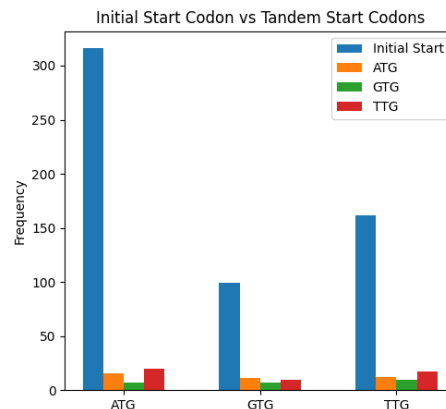
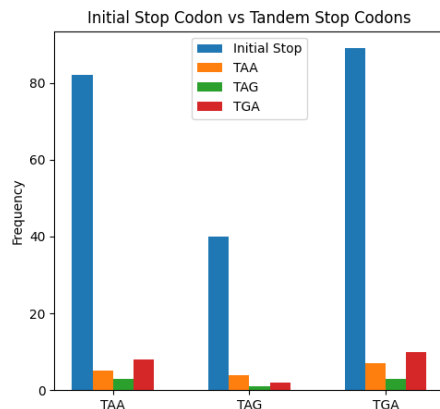
As you can see 662 of the genomes was analysed by this program. It would be impossible to go through each individual file. However, I will analyse a few which have some significance to my projects aims and hypnosis.

6.3. The Eukaryotic genomes graphs.

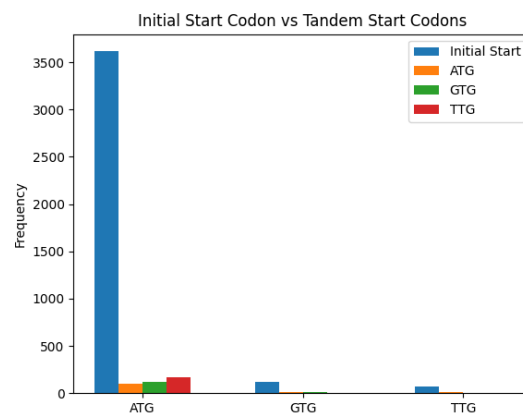
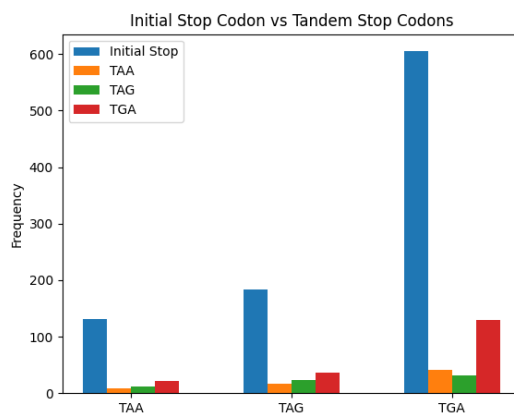
Below are a few graphs from the output directory that I thought were worth including to get a good representation of the types of results gathered from the project.



Cryptococcus_neoformans_var_grubii_gca_002222015.Cryp_neof_grubii_cng8_V1



Aspergillus_brasiliensis_cbs_101740_gca_001889945.Aspbr1



As we can see from these graphs TGA is by far the most common stop codon used on these eukaryotic genomes but not only that TGA is also by far the most common tandem stop codon used by the other initial stop codon.

As for start codons these graphs confirm their existence in the 12 bases before the initial start codon, ATG is by far the most common start codon used on these eukaryotic genomes and not only that, but the most common tandem start codon is TTG.

Looking at the Cryptococcus genome, this genome almost equally use all start codons for their genes in huge contrast to other eukaryotic genomes but not only that when looking at their tandem start codons used, the most common tandem start codon is the same as the initial codon for this genome.

```
Stop codon frequencies: ID=gene:CNA00430:
The initial stop codon was: ATG
TAA : 0 at indexes []
TAG : 1 at indexes [3]
TGA : 0 at indexes []

Start codon frequencies: ID=gene:CNA00080:
The initial start codon was: CTT
ATG : 1 at indexes [0]
GTG : 0 at indexes []
TTG : 0 at indexes []
```

```

Start codon frequencies: ID=gene:CNA00150:
The initial start codon was: CAA
ATG : 1 at indexes [0]

GTG : 0 at indexes []

TTG : 0 at indexes []

Stop codon frequencies: ID=gene:CNA00160:
The initial stop codon was: TTC
TAA : 1 at indexes [3]

TAG : 0 at indexes []

TGA : 0 at indexes []

Start codon frequencies: ID=gene:CNA00160:
The initial start codon was: TTT
ATG : 0 at indexes []

GTG : 0 at indexes []

TTG : 1 at indexes [6]

```

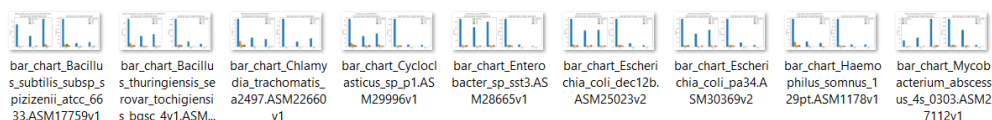
Looking at the main text file out put here we can see the index of these codons in the selected sequence. There are many tandem codons found where their wasn't an initial this could be that the tandem codons are doing their job here as readthrough backups but it is more likely however that they are just random occurrences.

6.4. Prokaryotic genomes?

The shared One drive directory from Wayne which provided the data for this project at the start of the project only contained Fungi genomes so Eukaryotic genomes thus u haven't been able to create a large results data output for this like I have for the Eukaryotic genome files.

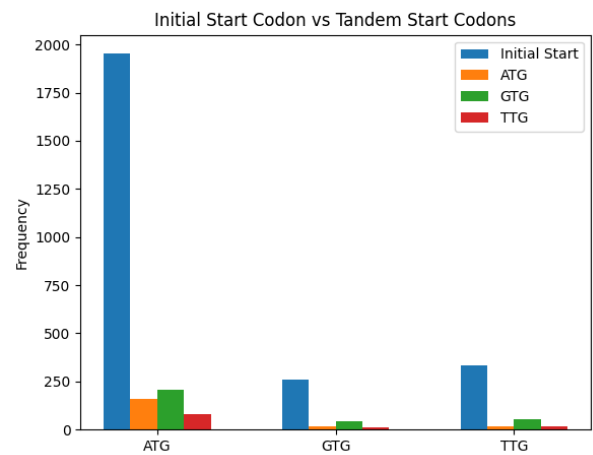
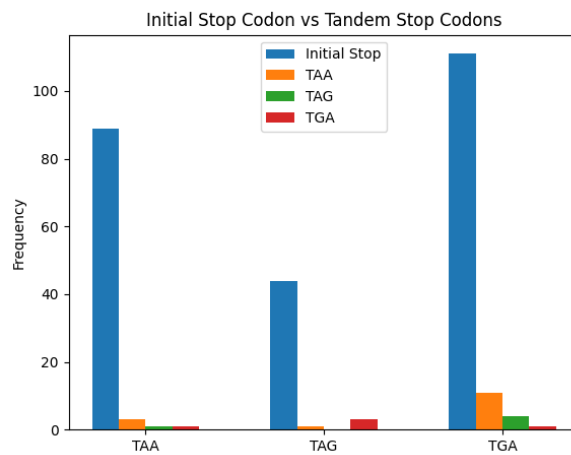
Name	Modified	Modified By	File size	Sharing
yersinia_pestis_py_101	4 minutes ago	Wayne Aubrey [wsa2]	Sta 2 items	[B] Shared
bacillus_subtilis_subsp_spizizenii_atcc_6633	4 minutes ago	Wayne Aubrey [wsa2]	Sta 2 items	[B] Shared
streptococcus_pneumoniae_taiwan19f_14	4 minutes ago	Wayne Aubrey [wsa2]	Sta 2 items	[B] Shared
escherichia_coli_w_gc_000258145	4 minutes ago	Wayne Aubrey [wsa2]	Sta 2 items	[B] Shared
mycobacterium_abscessus_6g_0125_s	4 minutes ago	Wayne Aubrey [wsa2]	Sta 2 items	[B] Shared
lactobacillus_coleohominis_101_4_chn	4 minutes ago	Wayne Aubrey [wsa2]	Sta 2 items	[B] Shared
bacillus_thuringiensis_serovar_tochigiensis...	4 minutes ago	Wayne Aubrey [wsa2]	Sta 2 items	[B] Shared
clostridium_saccharolyticum_wm1	4 minutes ago	Wayne Aubrey [wsa2]	Sta 2 items	[B] Shared
klebsiella_pneumoniae_subsp_pneumoniae...	4 minutes ago	Wayne Aubrey [wsa2]	Sta 2 items	[B] Shared
bacteroides_dorei_c03t12:c01	4 minutes ago	Wayne Aubrey [wsa2]	Sta 2 items	[B] Shared

Wayne was able to create a shared one drive directory with bacteria genomes late in to the project, However, the directory format meant that I couldn't just download 2 files, I would have to download each file individually by hand, and I wouldn't have time to do that the day before the project deadline, so I resorted to just running the program on a few of the files I hand picked.



Name	Date modified	Type	Size
outfile_Bacillus_subtilis_subsp_spizizenii_...	04/05/2023 20:28	Text Document	1,240 KB
outfile_Bacillus_thuringiensis_serovar_toc...	04/05/2023 20:28	Text Document	1,781 KB
outfile_Chlamydia_trachomatis_a2497.AS...	04/05/2023 20:28	Text Document	295 KB
outfile_Chlamydia_trachomatis_a2497.AS...	04/05/2023 20:28	Text Document	295 KB
outfile_Cycloclasticus_sp_p1.ASM29996v1...	04/05/2023 20:28	Text Document	669 KB
outfile_Enterobacter_sp_sst3.ASM28665v...	04/05/2023 20:28	Text Document	1,277 KB
outfile_Escherichia_coli_dec12b.ASM2502...	04/05/2023 20:28	Text Document	1,775 KB
outfile_Escherichia_coli_pa34.ASM30369v...	04/05/2023 20:28	Text Document	1,719 KB
outfile_Haemophilus_somnus_129pt.ASM...	04/05/2023 20:28	Text Document	532 KB
outfile_Mycobacterium_abscessus_4s_030...	04/05/2023 20:28	Text Document	1,489 KB
outfile_Salmonella_enterica_subsp_enteri...	04/05/2023 20:28	Text Document	1,294 KB
outfile_Salmonella_enterica_subsp_enteri...	04/05/2023 20:28	Text Document	1,415 KB
outfile_Streptococcus_pneumoniae_taiwa...	04/05/2023 20:28	Text Document	608 KB
outfile_Streptomyces_viridochromogenes...	04/05/2023 20:28	Text Document	2,322 KB
outfile_Yersinia_pestis_py_101.ASM26946...	04/05/2023 20:28	Text Document	1,469 KB

Bacillus_subtilis_subsp_spizizenii_atcc_6633.ASM17759v1



outfile_Bacillus_subtilis_subsp_spizizenii_atcc_6633.ASM17759v1.dna.toplevel.fa - Notepad

File Edit Format View Help

TGA : 0 at indexes []

Start codon frequencies: ID=gene:BSU6633_00535:

The initial start codon was: ATG

ATG : 1 at indexes [0]

GTG : 1 at indexes [3]

TTG : 0 at indexes []

Stop codon frequencies: ID=gene:BSU6633_00540:

The initial stop codon was: AAA

TAA : 0 at indexes []

TAG : 0 at indexes []

TGA : 1 at indexes [0]

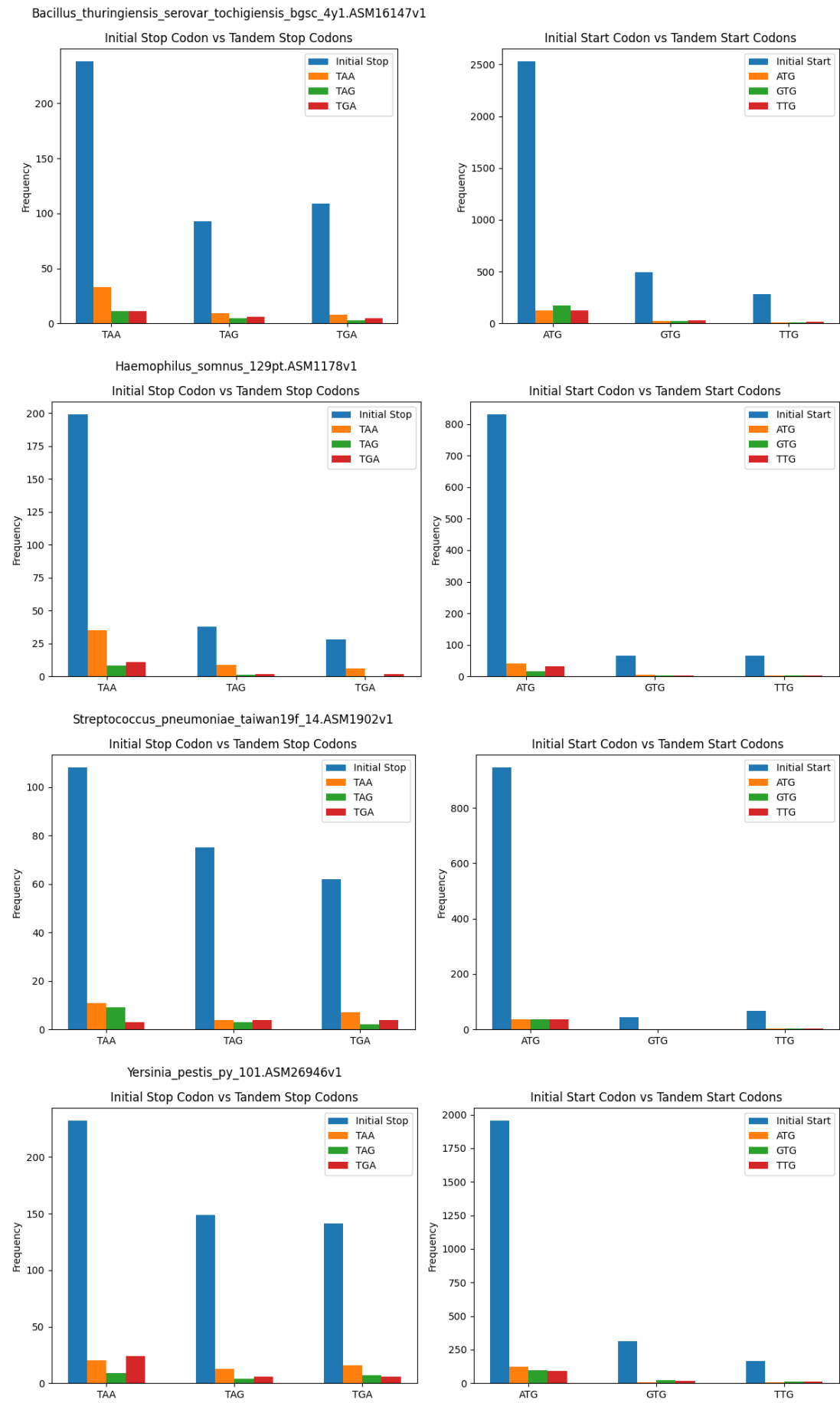
Start codon frequencies: ID=gene:BSU6633_00540:

The initial start codon was: ATG

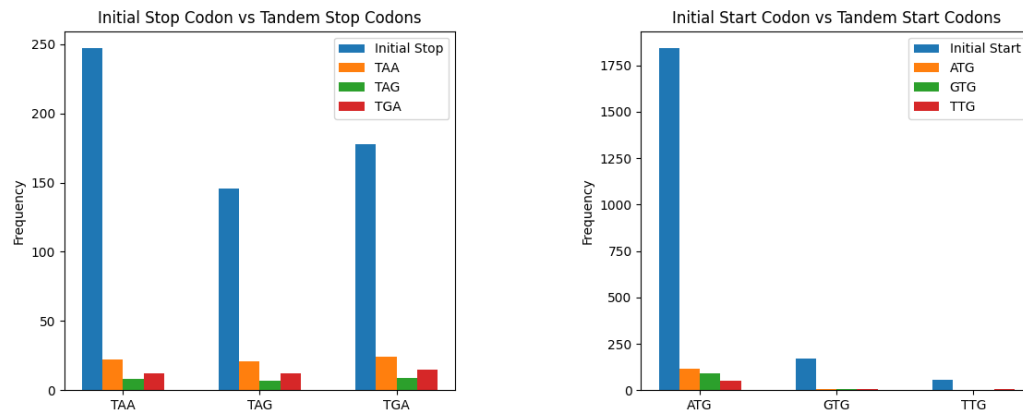
ATG : 0 at indexes []

GTG : 0 at indexes []

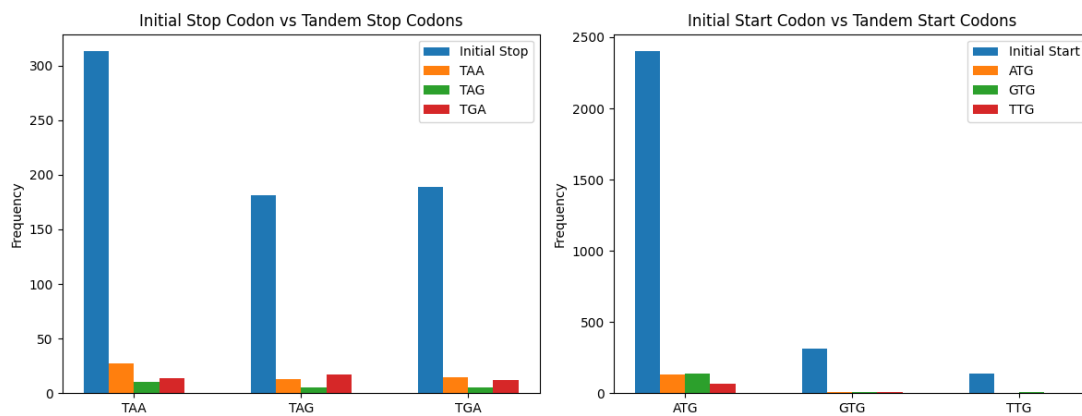
TTG : 0 at indexes []



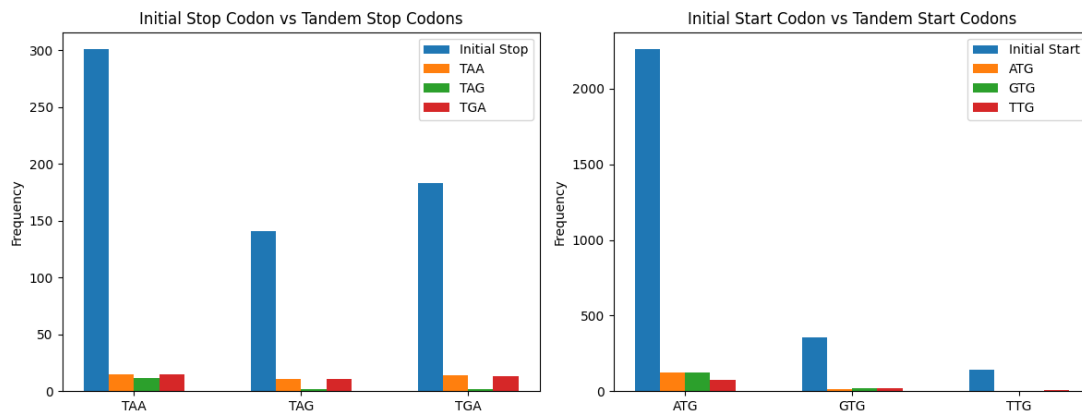
Salmonella_enterica_subsp_enterica_serovar_heidelberg_str_b182.ASM25836v1



Escherichia_coli_dec12b.ASM25023v2



Escherichia_coli_pa34.ASM30369v2



When Looking stop codons data from these files we can see that these prokaryotic genomes have a clear preference for the TAA stop codon but not only that when looking at the frequency of tandem stop codons we can also see a clear preference for the TAA codon as a tandem stop codon

When looking at the start codons data from these files we can see a similar preference of ATG as the main initial start codon. However, we can see that the preferred tandem start codon is also ATG the same as the initial this is difference from the eukaryotic genomes which had a preference for TTG as its tandem start codon. We can also variation based on genome to, when looking at bacillus subtilus there is a higher frequency of GTG start codons although it

isn't the most preferred initial start codon its certainly the most preference tandem for this genome, and the second most preferred initial start behind ATG.

6.5. Conclusions

The result from this project has the hypothesis "The type of tandem Start codon and stop codon used by be specific genes and their frequency could potentially be genome specific instead of random". And has confirmed the existence of tandem start codons in eukaryotic genomes and in Prokaryotic genomes. It was also interesting to see how different frequencies of stop and start codons and their tandem stop and start codons change based on the genome as well as the difference in preferred tandem stop codon between eukaryotic and prokaryotic. TGA being preferred for eukaryotic, and TAA being preferred for prokaryotic.

It should also be mentioned that these graphs show all of the initial codons found and only shows the tandem codons found in the sequences when the initial codon is a stop or start based on what its looking for, that means graph will show way more initial codons than their tandem counterparts and vice versa only display tandem stop codon counts when there was a correct initial codon.

However unfortunately I wasn't able to receive a large amount prokaryotic genomes files to run the program on, so this data is only relevant to eukaryotic genome's and a few of the Prokaryotic genomes I could obtain.

7. Critical Evaluation

7.1. Introduction

In this section I will critically evaluate my progress undertaking this project. If the requirements identified were correct. If early design for the project were correct and any changes made. If there are any more suitable tools to use to help improve the development process. And If I started the project again, what would you do differently as well as what would I do in the future if I was to continue working on the project?

7.2. Project Aim

The aim of this project was to produce some bioinformatics software that could analyse a large data set of genomes to look at their frequency of tandem stop codons and tandem start codons based on their initial codon to look for patterns between genomes and look for patterns of codon usage for eukaryotic genomes and potentially prokaryotic genomes.

The main aim of this project was achieved, the software I developed can analyse a large directory of fasta file and gff file genomes and output 2 files for each genome, first a text file with the tandem codons found for each gene as well as their initial codon and indexes also a count of the total tandem stop and start codons found with the genomes gc content. Second

a .png file of a graph for the genome with 2 bar charts one for the tandem stop codons and another for the tandem start codons.

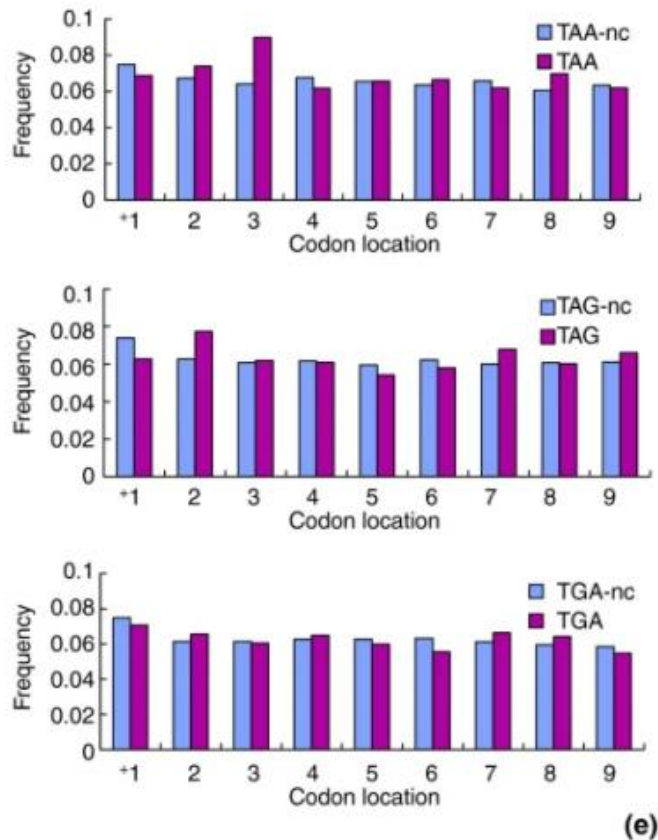
I was able to effectively use the output files too look at the patterns between the tandem codons and their initial codon and draw some informative conclusions for this data.

I would have like to implement a way of including the indexes in into the graph, currently you can only see the indexes for the codons for each gene in the text file.

I also would have like to run my project on some prokaryotic genomes similar to the ensemble fungi eukaryotic genomes, However I was not provided with the data to do so.

7.3. Requirements

- Produce a program that imports fasta files and gff files.
This system must be able to import large amounts of fasta and gff files from their directories and match them up correctly. Since each fasta has a matching gff file. The system is capable of easily doing so using the python glob module in the main method I wrote. This aspect of the system was test thoroughly.
- Produce a program that analyses the input data.
This program must be able to obtain the correct info from the gff and obtain the frequencies from the fasta file. This program can effectively obtain the correct information from the gff and index the fasta to obtain the frequencies of these codons and their indexes.
- Produce a program that output the data preferably a in a visual graph format.
This System must be able to output the data collected and not just output it to the terminal. This program is able to output 2 files containing the data collected about the frequency of tandem stop codons used and tandem start codons used. My program meets the requirements for this output and representation of the data, however I would have liked to add a representation of the indexes in the visual bar chart graph. Like this [1] from “the conservation of tandem stop codons in yeasts”



7.4. Design

7.4.1. Choice of Tools, IDE, Programming language

At the beginning of the project I decided to use python, since it was a bioinformatics project and python is often used for many bioinformatics projects around the world it seemed like the best fit. And for the most part I was correct using python for this project was most likely the best choice, as I was provided with many libraries that I could use to help develop solutions in my code. However, that being said python isn't the fastest language for running solutions with. My only concern is that given much larger data sets this program could potentially take a long time to finish processing. Running my project on the given dataset from Wayne already took a long time to finish running and the dataset wasn't that large.

My initial choice of IDE was bad, the Synder python gave me many problems when developing my code, however switching to Pycharm was easy and took no time at all seemingly solving all of my problems at once.

My first choice of unzipping method Winzip could have been better but at the time I was unaware of anyway to unzip .gz zipped files on my windows machine so the winzip free trial was best given my early options. Luckily, I found a much easier method of using a git terminal to unzip these .gz zipped files for visual inspection.

7.4.2. Single file python file design

Choosing to have my project as a single python file was a good decision based on the fact that is this a bioinformatics project, splitting up the project into multiple files is unnecessary, simply using a variety of functions for this project is more than enough.

7.5. Project management

My project management during this project was good, the use of the kanboard to set tasks to do very good as it gave a sense of urgency to complete these tasks and stay on top of things effectively. Likewise seeing all of the completed tasks is extremely motivating and allowed me to feel good about my project so I could potentially complete my best work.

This coupled with my weekly group meetings and weekly personal meeting with Wayne further boosted my performance since I could display and discuss my progress on project with others and gain their insight. Furthermore, I could hear their progress on their projects which motivated me to complete tasks.

7.6. Testing strategy

My test strategy throughout the process was overall successful, since much of my project is to write functions from use in main. I would create unit tests using other python files in my testing project where I could input fabricated data and compare the output with the expected values. This type of testing is more than enough for a bioinformatics project of this type.

Although, that being said I could have potentially done more integration testing to test other functions being used together. This wasn't a problem when developing the code for this project however I believe it would demonstrate better programming practice to do so.

7.7. Future Development

There are a few ways that I could improve this project if I was to start again or continue working on it in the future.

Firstly, the speed of this program is not the best that could be attributed to the use of python for this project, so to solve this problem I could rewrite much of what I have in c or potentially rust to massively increase processing speed.

Secondly, having a Graphical user interface would be ideal for biologist users who lack the technical experience to use this program, this gui could ask the users to provide the file paths to the fasta and gff files. Then the program could run and give the user the option to look at the graph produced or potentially each gene or all the genes with tandem codons. This gui could also potentially be developed as more of a multi tool to provide the user with more information.

Finally, running this program on a large number of prokaryotic genomes could potentially be beneficial, unfortunately I was unable to do so in this project however the code is written so program is ready to go all it needs is the data.

8. References

- [1] Liang, H., Cavalcanti, A.R. & Landweber, L.F. Conservation of tandem stop codons in yeasts. *Genome Biol* 6, R31 (2005). <https://doi.org/10.1186/gb-2005-6-4-r31>
- [2] Wang Liu-Wei, Wayne Aubrey, Amanda Clare, Robert Hoehndorf, Christopher J. Creevey, Nicholas J. Dimonaco
 bioRxiv 2022.09.16.508314; doi: <https://doi.org/10.1101/2022.09.16.508314>
- [3] Neil Taylor, "MMP Information", 2021 (Online) Available at:
<https://teaching.dcs.aber.ac.uk/docs/2022/MMP/information/> Accessed 28th
 February 2021.

9. Figures

Figure 1 – Oleg A. Shchelochkov, M.D. OPEN READING FRAME (2023).

<https://www.genome.gov/genetics-glossary/Open-Reading-Frame>

Figure 2- <https://ftp.ensembl.org/pub/>

Figure 3- https://prifysgolaber-my.sharepoint.com/personal/waa2_aber_ac_uk/_layouts/15/onedrive.aspx?ct=1678795759534&or=OWA%2DNT&cid=24d4d635%2D61c9%2D28c7%2Dbe6f%2Dd0fab4221d79&ga=1&isAscending=true&id=%2Fpersonal%2Fwaa2%5Faber%5Fac%5Fuk%2FDocuments%2Fensemble%5Ffungi&sortField=LinkFilename&view=0

Figure 4- https://prifysgolaber-my.sharepoint.com/personal/waa2_aber_ac_uk/_layouts/15/onedrive.aspx?ct=1683212560696&or=OWA%2DNT&cid=72172e3b%2D82f5%2Ddb284%2Ddaa1%2D7d9715d0d405&ga=1&id=%2Fpersonal%2Fwaa2%5Faber%5Fac%5Fuk%2FDocuments%2Fensemble%5Fbacteria&view=0

Figure 5- <https://genomebiology.biomedcentral.com/articles/10.1186/gb-2005-6-4-r31>

Figure 6- <https://www.winzip.com/en/>

Figure 7- <https://matplotlib.org/>

10. Appendices

In this section I will discuss any third part code that I have used, such as the imported libraries, and any tools that I used during the project and any tools that I used.

10.1. Tools used.

10.1.1. Winzip

Winzip is a file unzipping tool, used to allow users to easily unzip files on Microsoft windows operating systems, this software tool was initially needed for my project as I was developing my project on my Lenovo ideapad5 laptop that currently has the windows 10 operating system installed and windows does not have an inbuilt way for opening .gz zipped files to view / display on screen, this was useful initially at the start of the project before I found a way to use my git terminal to use gunzip to un zip the files that Wayne had provided in the shared One drive directory.

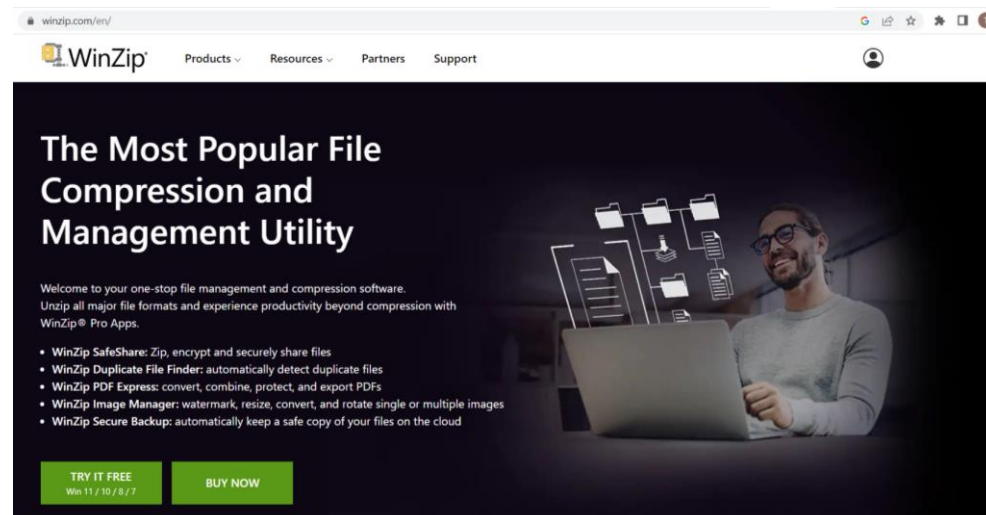


Figure 6

A. Third-Party Code and Libraries

These are the following Libraries that I implemented into this bioinformatics project;

Matplotlib – Matplotlib is a plotting library for use when programming in the python language and its numerical mathematics extension NumPy. Matplotlib provides an object oriented API for embedding plots into applications using general purpose Gui python toolkist for example, Tkinter and wxPython. It was initially released in 2003 by author John D. Hunter. Matplotlib was essential to producing the out put graphs that I needed for my projects results. It allowed be to easily create a bar chart and parse and populate it with the correct data collected.

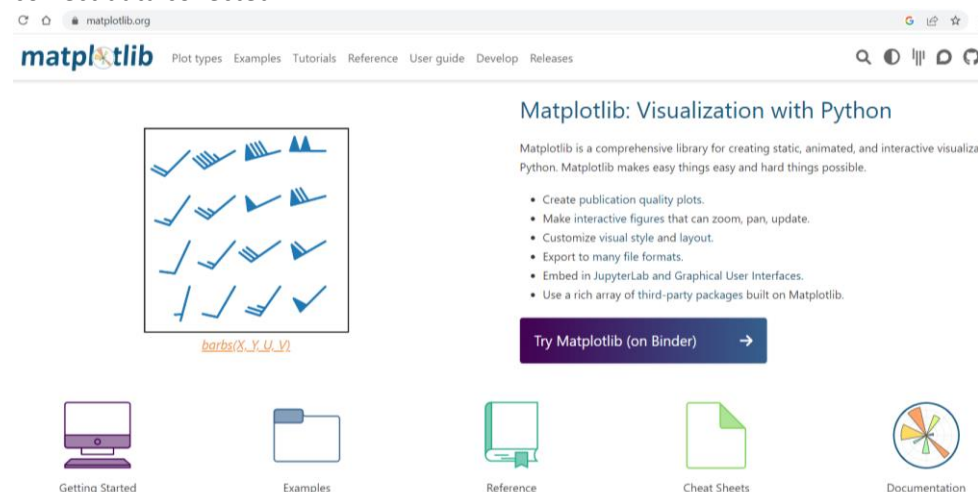


Figure 7

Glob – Glob is a Python module that allows the user to find and import all of the path names for a specified pattern according to the rules used by the unix shell. It allows the user to import all of the file path names from a specified directory into a list. The unix commands can be used to specify which files to import. For example in my project Imported the top down dna fasta file path names from a specified directory and used the “*.fa.gz” unix expression to only import the fasta files that where .gz zipped.

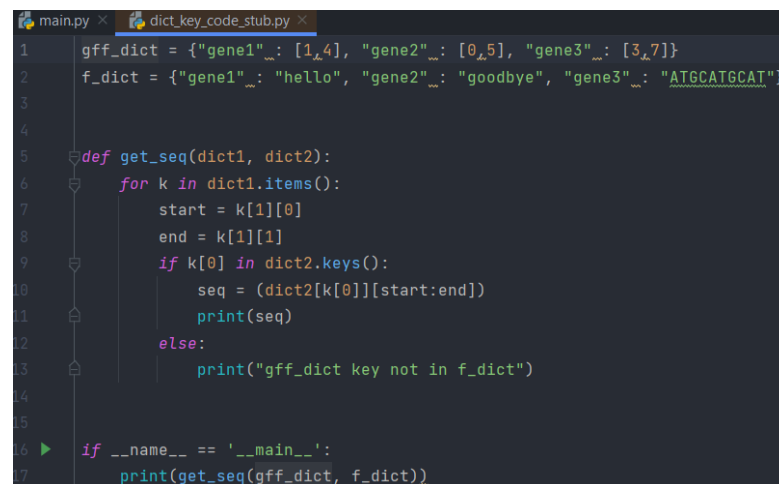
Gzip – Gzip is a python module that allows users to unzip gzipped / gunzipped files like gzip and gunzip would. It provides a simple interface to compress and decompress files. This module was very important for my project since the files provided to be by Wayne where .gz zipped and had to first be unzipped. I implemented the use of this module into my file opening function that checks to see if a file has the .gz extension and if it does it uses the gzip module to unzip the file and open it to be read.

OS – OS is a python module that provides the user with a portable way of using operating system dependent functionality. It allows uses to easily use and manipulate system file paths for use in python programs. For example I used OS in my project to create a gff file path string using os.path.join and then I used os to check that the file path I created existed in a specific gff file directory using os.path.exists.

B. Code Samples

10.1.2. Code examples provided

The following code extracts where piece of code that where provided to me by Wayne, Although these extracts might not have actually been used in my project or may have been changed it believe that it is important to include them since, they provided me with a direction to follow on from with my project and were extremely useful for mentally planning the design of certain functions that would be created later on in the project. The piece of code below for opening files was written by me however I was shown a similar chunk of code from Wayne which help me produce this, same with the parse gff dict.



```

1  gff_dict = {"gene1": [1,4], "gene2": [0,5], "gene3": [3,7]}
2  f_dict = {"gene1": "hello", "gene2": "goodbye", "gene3": "ATGCATGCAT"}
3
4
5  def get_seq(dict1, dict2):
6      for k in dict1.items():
7          start = k[1][0]
8          end = k[1][1]
9          if k[0] in dict2.keys():
10             seq = (dict2[k[0]][start:end])
11             print(seq)
12         else:
13             print("gff_dict key not in f_dict")
14
15
16 if __name__ == '__main__':
17     print(get_seq(gff_dict, f_dict))

```

```

my_list = "AAATTTGTAATGA"
stop_codons = ["TAA", "TAG", "TGA"]
for i in stop_codons:
    if i in my_list:
        if my_list.index(i) % 3 == 0:
            print(my_list.index(i))
        else:
            print("not in frame",
my_list.index(i))

```



```

def openfile(filePath): # opening and reading the fastafile and gfffile
    if filePath.endswith(".gz"):
        with gzip.open(filePath, 'rt') as f:
            return [l.strip() for l in f.readlines()]
    else:
        with open(filePath, 'r') as f:
            return [l.strip() for l in f.readlines()]

def parse_gff(gff_file):
    gff_dict = {}
    f = openfile(gff_file)
    for line in f:
        if line.startswith('#'):
            continue
        fields = line.strip().split('\t')
        if fields[2] == 'gene':
            gene_id = fields[8].split(':')[0]
            start = int(fields[3])
            stop = int(fields[4])
            chromosome = fields[0]
            gff_dict[gene_id] = (start, stop, chromosome) # creates gff dict
    return gff_dict # Press Ctrl+F8 to toggle the breakpoint.

```