Project Specification

1. **Background and context**

The project consists of a Python implementation of a dataset class that is capable of performing various operations on a GFF3 file format. GFF3 is a widely used format for representing genomic annotation data, containing information about genomic features such as genes, exons, etc. The Dataset class offers various functionalities that enable users to manipulate GFF3 files, such as basic information about the dataset, counting features by source and filtering datasets based on specific criteria. The code utilizes various Python libraries, including gffpandas, pandas and ABC. This implementation offers an easy-to-use interface for manipulating GFF3 files.

1. **Project Objective**

To develop a web-based application that can process genomic data obtained from a gff3 file using a dataset reader that relies on Pandas. The application should be capable of handling 11 operations that require the traversal, manipulation and reading of the gff3 file, and returning another dataset object as output.

1. **Expected Results**

The rough code for each element of the project is expected to be completed within 2.5 weeks of the project initiation. This includes time for analysing of what is expected of us for each project component, and for initial brainstorming sessions. The final 10 days will be used for developing the project documentation, rendering the rough code more user-friendly, and combining each member’s individual components into a final code. Several team meetings and check-ups will be planned to monitor progress and potential update the timeline if more time is necessary.

1. **Inputs**

We relied on Visual studio code and collab as interpreters for our codes. We used python and html as programming languages. Throughout the development of the project, we imported several modules including pandas, gffpandas, NumPy and flask.

1. **Roles and Responsibilities:**

* Reader – Chiara
* Operations development– Chiara and Laura
* CRC cards and UML diagram – Sedef
* Project Documentation – Laura and Sedef
* HTML web-based user interface - Natalija
* GitHub depository development – Natalija

1. **Project Management:**

Three zoom meetings were held over the course of this project. The first 2 were carried out within the first week of the project initiation, as this was a critical moment to lay the foundation for the project. The first was used for initial brainstorming and role-assignment. The second was used to monitor progress. The third meeting was held a 2 weeks later. Due to some miscommunication, the rough code was not completed in the expected timeline, so a meeting was held with all members that were responsible for the reader and its associated operations. General monitoring and management was carried out using a group chat and collaborative work spaces. A final in-person meeting was carried out a week prior to the deadline of the project.

Communication between group members was carried out in a WhatsApp group chat, where each member had the responsibility of reporting on their progress. We also made use of multiple shared OneNote and Word documents. We used Google Collab to write the code and updates and changes to the shared code were communicated via the WhatsApp group

1. **Project Activities:**

|  |  |  |
| --- | --- | --- |
| **Name of Activity** | **Completed by:** | **Completion date** |
| Reader | Chiara | **22/03** |
| Operation 1 – basic\_info | Chiara | **22/03** |
| Operation 2 – unique\_sequence\_ids | Chiara | **24/03** |
| Operation 3 – unique\_operation\_types | Chiara | **01/04** |
| Operation 4 – count\_features\_by\_source | Chiara | **01/04** |
| Operation 5 – count\_entries\_by\_operation\_type | Chiara | **24/04** |
| Operation 6 – chromosome\_dataset | Laura | **25/03** |
| Operation 7 – Fraction\_unassembled\_seq | Laura | **25/03** |
| Operation 8 – ensembl\_havana\_dataset | Laura | **04/04** |
| Operation 9 – count\_entries\_by\_type\_ensembl\_havana | Laura | **03/04** |
| Operation 10 – gene\_names\_ensembl\_havana | Laura | **06/04** |
| HTML page | Natalija | **22/03** |
| Implementation of operations in HTML | Natalija | **08/04** |
| CRC cards and UML diagram | Sedef | **29/03** |

1. **Execution**

A brief timeline describing how we completed each step of the project:

Sunday 12 March – First zoom meeting

Monday 13 March – Second zoom meeting

Wednesday 19 March - Rough draft of the algorithms for each operation

By 23 March we had a rough outline of the first 5 operations and the first draft of the reader

Saturday 25 March – Third zoom meeting, set deadline for 29 March for UML diagram, cards and last operations

Wednesday 29 March - first draft of the CRC and UML cards

Set another deadline of 3 April to finish the code

In person meeting on 2 April

Monday 3 April - final operations were completed and implemented

Tuesday 4 April – group starts working on the project documentation

1. **Website development**

Prior to having the reader or any operations set up, it was possible to create the html page and fine-tune it appearance-wise. We relied on tutorials found on W3Schools.com, as was recommended. In order to explore abilities of html language, different approaches for inserting images were used. Background was inserted using a link in img tag, while the CRC cards and UML diagram were inserted from a folder. This line:

<meta name="viewport" content="width=device-width, initial-scale=1.0">

allowed us to make the text inserted into page responsive (occupy the same percentage of the page), no matter how much we pull the page wider or narrower. In addition to this, scroll bar was also programmed.

The idea was to make a simple and accessible website which we think was successfully achieved. The homepage provides user with descriptions of all the operations. The navigation bar allows seamless transition between different pages. User can choose an operation to run from the list provided on active operations page.

Once the codes for reader and operations were available, jinja was applied to implement them into the html page. In some instances, html tables were used to display data in a presentable manner.

During implementation of operations in html page, we’ve faced difficulties when it comes to opening files, or more precisely, finding the accurate file path. We relied on os.path.abspath() to solve this. For reasons unknown to us, the website runs smoothly from Thonny, but still faces issues when run from VSCode.

Websites detailing Gff3 documentation, methods and attributes were relied on to develop a comprehensive understanding of the structure of Gff3 files (columns, their content etc,) and their associated methods and attributes. Reviewing the lessons on Pandas provided the necessary revision of Pandas-associated methods, which rendered the implementation and execution of the dataset operations more efficient.

1. **Description of operations**

**Operation 1: basic\_info**

This operation returns the column names and their respective data types to a list. the names of each column and the data they store is universal in all gff3 files.

**Operation 2: unique\_sequence\_ids**

This operation traverses column 1 (‘seqid’), storing in a separate list all unique sequence IDs that are encountered. Multiple features can have the same sequid but store different types of data (like exons or supercontigs).

**Operation 3: unique\_operation\_types**

This operation traverses column 3 (‘types’), storing in a separate list all the unique types of operations. Both operation 2 and 3 use the .unique() method provided by Pandas

**Operation 4: count\_features\_by\_source**

This operation returns a dictionary where the key-value pairs correspond to the sources from which the features have been obtained (column 2) and the total number of features that originate from that source. This operation uses the .groupby() and the .count() method provided by Pandas

**Operation 5: count\_entries\_by\_operation\_type**

This operation follows a similar structure to Operation 4, but the features are grouped and counted based on their operation types (column 3)

**Operation 6: chromosome\_dataset**

This operation returns a new data frame containing all features about entire chromosomes, deriving from source GRCh38. The data frame is created by filtering the data based on ‘source’ (column 2)

**Operation 7: fraction\_unassembled\_seq**

This operation creates a separate data frame containing all the data from source ‘GRCh38’ , and then counts the fraction of those filtered features that are unassembled (i.e. they are super contigs)

**Operation 8: ensemble\_havana\_dataset**

This operation creates a new dataset containing all data obtained from sources ‘havana’, ‘ensembl’ and ‘ensembl\_havana’

**Operation 9: count\_entries\_by\_type\_ensembl\_havana**

This operation creates a new dataset containing all data obtained from sources ‘havana’, ‘ensembl’ and ‘ensembl\_havana’, and the counts the number of entries of this new dataset by type, storing them in a dictionary. It combined features implemented in Operation 8 and Operation 5

**Operation 10: gene\_names\_ensembl\_havan**

This operation creates a new dataset containing only data of type ‘gene’ that derives from sources ‘havana’, ‘ensembl’ and ‘ensembl\_havana’, and then filters the attributes column (column 9) for the names of each gene, storing them in a list.

1. **Results**

We were successfully able to write a code that can perform functions on the GFF3 dataset. There is a single class called "Dataset" with a parent class called "DatasetInterface". "Dataset" has several attributes: "filepath", "data", and "\_\_active\_operations". "Dataset" also has several methods: "check\_df()", "activate\_operation()", "deactivate\_operation()", "is\_operation\_active()", "basic\_info()", "unique\_sequence\_ids()", "unique\_operation\_types()", "count\_features\_by\_source()", "count\_entries\_by\_operation\_type()", "chromosome\_dataset()", "fraction\_unassembled\_seq()", and "ensembl\_havana\_dataset()".”DatasetInterface" is an abstract base class that contains a single abstract method called "check\_df()"."Dataset" implements "check\_df()" and overrides it. "check\_df()" raises an exception if the file path does not end with ".gff3" and returns the GFF3 dataframe otherwise. "Dataset" also has a decorator called "operation" that wraps the methods that can be activated or deactivated. The decorator checks if the operation is active before executing it and raises an exception if it is not active. "Dataset" has three methods that activate or deactivate operations, which are "activate\_operation()", "deactivate\_operation()", and "is\_operation\_active()". These methods modify the "\_\_active\_operations" attribute. "Dataset" has ten methods that perform different operations on the data, which are "basic\_info()", "unique\_sequence\_ids()", "unique\_operation\_types()", "count\_features\_by\_source()", "count\_entries\_by\_operation\_type()", "chromosome\_dataset()", "fraction\_unassembled\_seq()", "ensembl\_havana\_dataset()". "Dataset" has two methods with the same name "chromosome\_dataset()". The first one returns a Pandas series with a boolean mask indicating whether the data source is "GRCh38". The second one returns a subset of the data where the source is "GRCh38".Finally, there is a line of code that creates an instance of "Dataset" and activates some operations.