Assignment Instructions

2024-2025 academic year

Contents

1 Aim of the assignment

This assignment tests your ability to use Python for parsing text files to extract data and make plots.

2 Introduction

The data files used for this assignment were generated by a bioinformatics tool called FastQC:

https://www.bioinformatics.babraham.ac.uk/projects/fastqc/

FastQC is a very popular tool for assessment of raw sequencing data. FastQC accepts data in FASTQ, SAM or BAM formats. Then, FastQC calculates a range of metrics, and generates a number of plots, that allow to evaluate the quality of sequencing data. You may see more information about the FastQC in its web page, and in the provided PDF file with FastQC_description, which explain the metrics and plots generated by FastQC. Along with the plots, FastQC produces a text file with the QC results used to generate the plots.

The QC information in the text file is divided in the following sections:

- Basic statistics
- Per base sequence quality
- Per tile sequence quality
- Per sequence quality scores
- Per base sequence content
- Per sequence GC content
- Per base N content
- Sequence Length Distribution
- Sequence Duplication Levels
- Overrepresented sequences
- Adapter Content
- K-mer Content

Each section has a header, staring with ">>" symbols, followed by the section name and the ***flag***: *fail*, *pass*, or *warn* (=warning), representing the high-level conclusion suggested by FastQC for this section. Each section ends with ">>END_MODULE" line. A fragment of FastQC text file is illustrated on Figure 1.

```
##FastQC	0.11.3
>>Basic Statistics	pass
#Measure	Value
Filename	4_age21_S12_L001_R1_001_concat.fastq.gz
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	37287903
Sequences flagged as poor quality	0
Sequence length	75
%GC	55
>>END_MODULE
>>Per base sequence quality	pass
#Base	Mean	Median	Lower Quartile	Upper Quartile	10th Percentile	90th Percentile
1	31.639491526246463	32.0	32.0	32.0	32.0	32.0
2	31.618002787660117	32.0	32.0	32.0	32.0	32.0
3	35.66454209023232	37.0	37.0	37.0	32.0	37.0
4	36.135634015138905	37.0	37.0	37.0	37.0	37.0
5	36.33817981129162	37.0	37.0	37.0	37.0	37.0
6	39.89045790534265	41.0	41.0	41.0	37.0	41.0
7	39.98228999362072	41.0	41.0	41.0	37.0	41.0
8	40.09800371450226	41.0	41.0	41.0	37.0	41.0
9	40.09410346835541	41.0	41.0	41.0	37.0	41.0
10	40.1102169515942	41.0	41.0	41.0	37.0	41.0
11	40.14826572575025	41.0	41.0	41.0	37.0	41.0
12	40.12512352866827	41.0	41.0	41.0	37.0	41.0
13	40.069397493337185	41.0	41.0	41.0	37.0	41.0
14	40.08390292154536	41.0	41.0	41.0	37.0	41.0
15	40.085222357502914	41.0	41.0	41.0	37.0	41.0
16	40.087172453757994	41.0	41.0	41.0	37.0	41.0
17	40.05952587357889	41.0	41.0	41.0	37.0	41.0
18	40.08766706457051	41.0	41.0	41.0	37.0	41.0
19	40.077069123463446	41.0	41.0	41.0	37.0	41.0
20	40.09115953771924	41.0	41.0	41.0	37.0	41.0
21	40.04739781156371	41.0	41.0	41.0	37.0	41.0
22	40.02398643871177	41.0	41.0	41.0	37.0	41.0
23	40.03484234551887	41.0	41.0	41.0	37.0	41.0
```

Figure 1: Fragment of the text file generated by FastQC

For this assignment you will be given two text files generated by FastQC. Explore their content to familiarize yourself with the format and content of different sections.

See more information about the FastQC sections and plots in the FastQC documentation web page:

https://www.bioinformatics.babraham.ac.uk/projects/fastqc/Help/3%20Analysis%20Modules/

For instance, “Per base sequence quality” section, illustrated in Figure 1, is used to generate the plot shown in Figure 2.

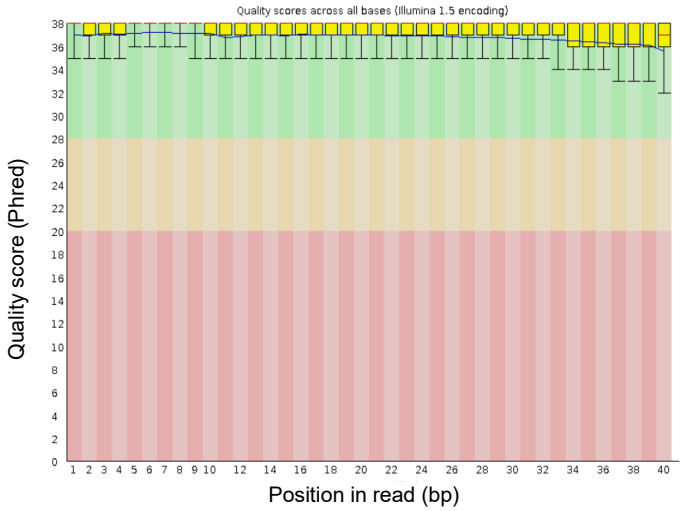

Figure2: Example of FastQC plot for “Per base sequence quality” section

3 Assignment tasks

Your task will be to write a Python program to parse FastQC text files, and make reports and plots required by the user.

Briefly, your program should: (i) run in the terminal, accepting command line arguments, and produce (ii) command-line output, (iii) summary text files and (iv) plots as descried below. In addition to the program, you should prepare (v) a concept diagram, illustrating the structure of your program, (vi) a report, with technical documentation and description of your program, and (vii) a short user-manual for your program. Also, you should test your script using test data files given to you and (viii) provide the results of testing.

The detailed program specifications, and some examples of input and output are given below.

3.1 Program specifications

Your program should run in the **command line / terminal**. You should not expect any IDE (like VS Code or PyCharm or Spyder) on the user’s computer.

3.1.1 Command line arguments

Your program should accept the following command line arguments:

- **the input file name** : the text file generated by FastQC
- **the output folder name** : the folder where the output files should be placed

These two arguments are mandatory, without default values, and cannot be omitted.

In addition to the input file and output folder names, your script should be able to accept the **optional arguments** specifying which section(s) of the input file should be processed, as shown in this table:

Short argument	Long argument	File section	Required outputs
-b	--per_base_seq_qual	Per base sequence quality	R, P, F
-t	--per_tile_seq_qual	Per tile sequence quality	R, P, F
-s	--per_seq_qual_scores	Per sequence quality scores	R, P, F
-c	--per_base_seq_content	Per base sequence content	R, P, F
-g	--per_seq_GC_cont	Per sequence GC content	R, P, F
-n	--per_base_N_cont	Per base N content	R, P, F
-l	--seq_len_dist	Sequence Length Distribution	R, F
-d	--seq_dup	Sequence Duplication Levels	R, P, F
-o	--over_seq	Overrepresented sequences	R, F
-p	--adap_cont	Adapter Content	R, P, F
-k	--kmer_cont	K-mer Content	R, P, F
-a	--all	All the above	As above

Table 1: Optional arguments

The required outputs include the Report (R), Plot (P), and a file with Flag (F) as explained later.

The user should be allowed to provide any combination of the optional arguments, or none of them. Independently of the provided optional arguments, the program should always print the content of the "Basic Statistics" section to the terminal.

In addition to the command line arguments described above, the script may provide option(s) to get help or show the version of the script. Command line arguments for help and version are not mandatory. However, they may bring additional marks.

Examples of command line arguments:

- User provides **-c** argument:
  `python your_script.py input_file output_folder -c`
  The content of the "Basic Statistics" section is printed to the terminal. The Report, Plot and file with Flag are generated for the "Per base sequence content" section. No other outputs are generated.
- User provides **-c** and **-o** arguments:
  `python your_script.py input_file output_folder -c -o`
  The content of the "Basic Statistics" section is printed to the terminal. The Report, Plot and file with Flag are generated for the "Per base sequence content" section. The Report and file with Flag are generated for the "Overrepresented sequences" section. No other outputs are generated.
- User provides argument --**all**:
  `python your_script.py input_file output_folder -all`
  The content of the "Basic Statistics" section is printed to the terminal. The required outputs are generated for all sections listed in Table 1.

3.1.2 Output files and folders

The following types of output should be provided, depending on the ***optional*** argument(s):

Report: the specified section should be extracted from the input file and saved into a separate text file. The report doesn't have to include ">>MODULE NAME" and ">>END_MODULE" lines.

Plot: the plot to illustrate section's content (except for the "Sequence Length Distribution" and "Overrepresented sequences" sections). The graphs do not need to be exactly like the ones generated by the FastQC tool. However, producing graphs of similar manner or even better will bring higher marks. The graphs may be generated in any image format (though PNG or PDF are preferred).

File with Flag: a text file containing a single word: *pass*, *fail* or *warn*, extracted from the first line of the section.

Outputs for each section should be placed into ***a separate sub-folder*** within the output folder specified by the user. The sub-folder names should be informative and must NOT contain any spaces or special characters (you may use underscore symbol instead of spaces). The output files should be named **report.txt**, **plot.png**/pdf and **flag.txt**.

3.2 Code features

In addition to accepting the specified inputs and generating the required outputs, your script(s) should have the following features:

3.2.1 Clear, structured, and understandable code

The code structure should be logical. The code should be modularised using functions/methods and/or classes; use of *main()* function is not obligatory but will bring higher marks. The code must include informative comments (in addition to the Docstrings). You should use meaningful names for variables, functions, modules and classes (if classes are used). Classes are not mandatory, but may bring higher marks, if appropriately used.

3.2.2 Error and exception handling

Use of proper error / exception handling is required. You should envisage as many run-time exceptions as possible and address them. Implementing ***if..,elif...else*** for error handling is discouraged. Dedicated Python methods for exception handling are recommended and will carry higher marks. Your code will be tested against a number of exceptions (e.g. missed input files, wrong arguments etc).

3.2.3 Proper use of Docstring

Use of Python Docstrings is highly recommended. Following Python conventions for Docstrings will bring additional marks (see: https://www.python.org/dev/peps/pep-0257)

3.3 Program testing

You are given two text files generated by FastQC. You shall test your program with the "--**all**" option using both provided test files. The testing results should be provided for assessment, including the output files within the individual sub-folders with proper naming. Also, you should include in your assignment submission the data that you used for testing.

3.4 Concept diagram

You should submit a **Concept diagram** to illustrate the structure of your script(s). The diagram may illustrate the general structure of your code, inputs/outputs, the used functions or classes etc. You may use computer aided method or free-hand drawing for this. Using standard methods to generate the diagram (such as: flow chart, spider diagram etc.) is desirable but NOT compulsory. The main goal is that the diagram should be informative and clear. You may refer to the diagram in your **technical documentation**.

3.5 Technical documentation

You should provide a written technical documentation (**Max 2000 words** excluding illustrations and tables) explaining the design of our application. If you grossly exceed the word limit, the marker may stop marking at 2000 words. The technical documentation may include:

- Description of overall design and structure of the program
- Description of main variables, methods, or classes (if classes are used)
- References to any imported classes/methods and other dependencies
- Examples and interpretations of the plots generated by your program
- etc.

The documentation may refer to the **concept diagram**. In short, the technical documentation should be written for *a professional bioinformatician* who needs to understand how your program works.

3.6 User manual

You should provide a **one-page** user manual for a non-bioinformatician. If you grossly exceed the size limit, the marker may stop marking at 1 page. The user manual should briefly outline the purpose of the tool, and how to use the tool. This may include installation (including OS and key dependencies), source data, command line options, expected outputs etc. Importantly, your instructions should explain how to run the software from the ***command line***. Do not assume that users are familiar with VS Code, Spyder, Jupyter etc. In short, the manual should be user-friendly and written in a way to enable a user *without an IT background* to run your script.

4 Marking criteria

Task	Maximal Mark
Code (70%)	
Passing command line arguments	5
Required output to the command line is generated	5
The required text files are generated in the required folders	15
The required plots are generated in the required folders	15
Clean readable/understandable code (with comments)	10
Error and exception handling	5
Proper use of Docstring	5
Professional looking graphs	5
Test results (including the input files that you used for testing)	5
Documentation (30%)	
Technical documentation	20
Concept diagram	5
User Manual	5
Total	**100**

5 Submission

Assignment must be submitted via CANVAS using the appropriate submission point and following the provided instructions.

You should submit all files in a single Zip archive containing:

- The Python **script(s)**
- The **input** files provided to you for testing (as well as any additional files, if you used additional files for testing your program)
- The **output** generated during your testing in the designated folders
- The **documentation**: Technical documentation + Concept diagram + User manual

The file name should include your student number and name like the following:

StudentName_StudentNumber_IBIX_PYT_24_assignment.zip

Example of the file name:

Larionov_s123456_IBIX_PYT_24_assignment.zip

Please replace "s123456" with *your* student number and "Larionov" with *your* name.

The submission deadline is provided in CANVAS in the Assignments section.

Questions?

E-mail to Dr. Alexey Larionov: alexey.larionov@cranfield.ac.uk