Fastqc Reporter Technical Documentation

By Oluwasusi David
Student Number: 447435

Table Of Contents

Introduction

Fastqc reporter is a Command Line Interface (CLI) tool built to parse fastqc files into the available sections and generate reports containing the section contents. It also generates a graphical representation of the section and a flag file containing the result of the QC test, which can be the values pass, fail, or warn.

Installation

To run this program, the following are required;

- From Python 3.9 upwards.
- Conda or venv (conda is used for the script in this documentation).

The next step is to create a new virtual environment and install the dependencies.

```
>>$ conda create -c conda-forge -n name_of_my_env seaborn pandas matplotlib
```

This will create a conda environment and install the seaborn, pandas, and matplotlib dependencies. Afterwards, activate the virtual environment by doing;

```
>>$ source activate name_of_my_env
```

Finally, run the program with its required parameters and optional parameters. See the Example Usage section for more info.

Program Design

Fastqc reporter takes an object-oriented approach to parsing fastqc files and generating appropriate reports and graphs. Two main classes exist;

- `FastQCParser`
- `Section`

The `FastQCParser` class encapsulates the functionalities of parsing the fastqc files into sections and defines the methods to handle the supported optional parameters when passed.

The `Section` class encapsulates the functionalities of writing each section's report and flag file to its appropriate folder depending on the optional parameters passed.

Each section has its class and inherits from the base `Section` class. All the properties of the section class are inherited and it now defines its implementation of the `plot_section()` method. This method plots the necessary graph that suits the type of data the section contains.

The data in each section is represented internally as a Python dictionary, where the keys represent the title of each section and the values are another dictionary having the section content and the fastqc test status.

A dictionary was used as the internal data structure because it enables us to parse the file once into memory and pick each section effortlessly.

Overview of Functionality

Fastqc reporter makes use of the `argparse` module from Python to add a description of what the tool is for. It also adds a list of parameters. Two parameters are required to run the script;

- The path to the fastqc file
- The folder to output the plots, reports, and flag

Other optional parameters are defined via the `add_argument` method of the `argparse` module. They are optional because the action parameter is set to `store_true`, which saves them if available.

The `FastQCParser` Class is instantiated with the two required parameters, and an empty dictionary is created. The `parse_fastqc_to_dictionary()` method of the class is called to read the file, and each section is saved into the dictionary with the section title as the key.

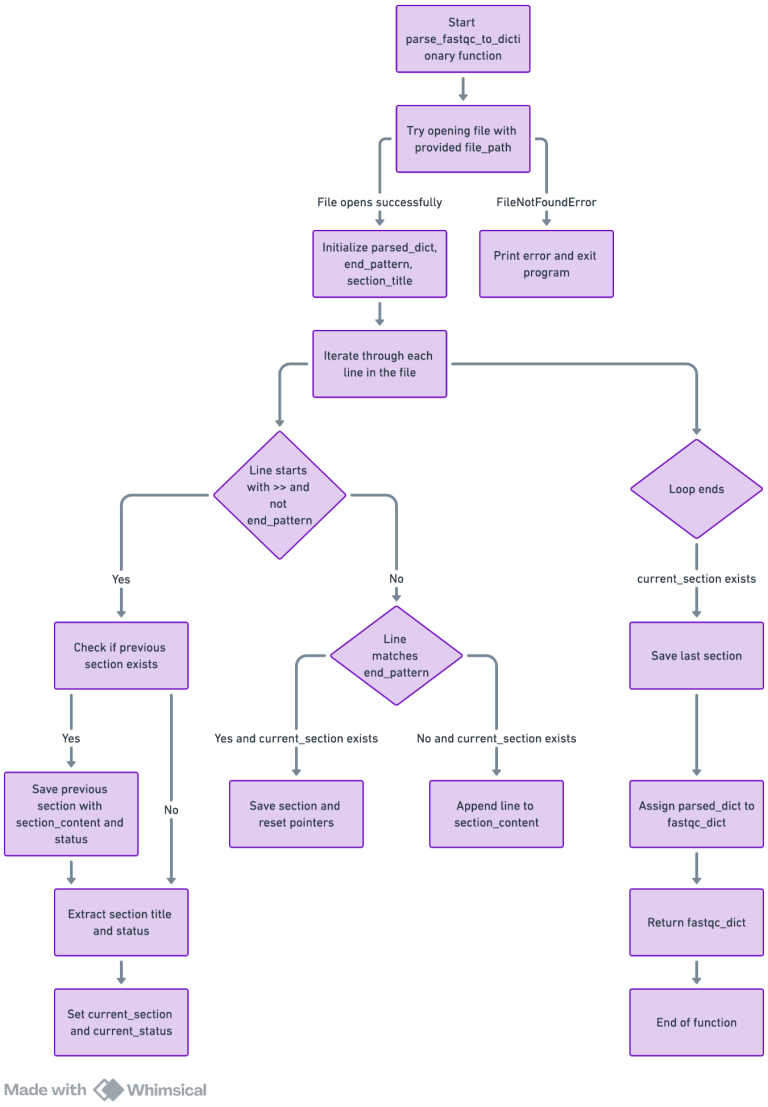

Figure 2: Flow chart diagram for the `parse_fastqc_to_dictionary()` method.

The program then checks to see if any optional arguments are passed in the command line and calls the appropriate method to handle it. The `FastQCParser` class defines methods to handle all the sections.

The concept diagram illustrating the structure of the program is shown below.

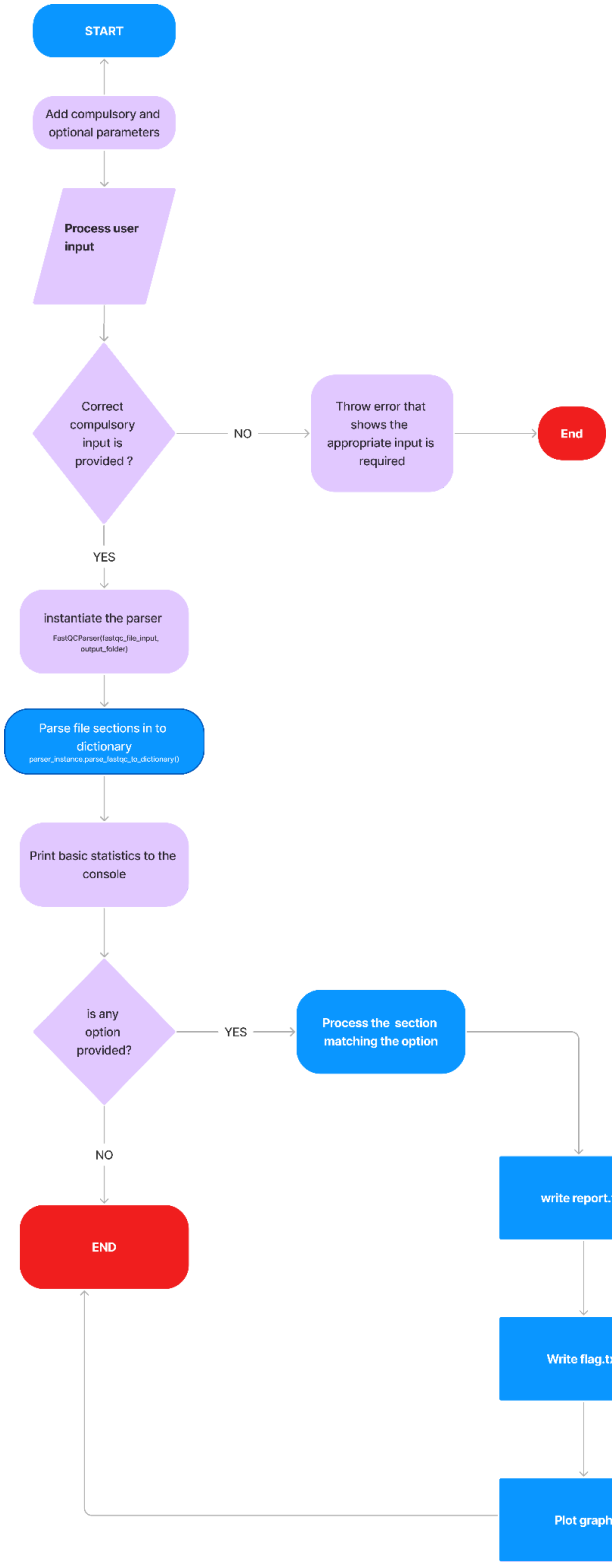

Figure 3: Flow chart diagram for Fastqc reporter program.

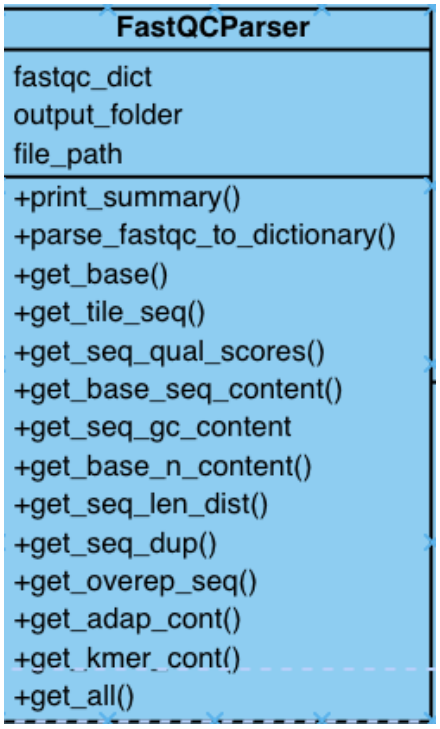

Figure 4: Class diagram for the `FastQCParser` class showing its attributes and methods to handle sections of the fastqc file.

All the methods for each option do the following;

- Create the appropriate `Section` class.
- Write the required outputs (report.txt, flag.txt, plot.png) to the folder specified.

The base statistics are always printed to the console, regardless of the section that is run. This is possible because of the static method `print_summary()`.

To see the list of options, pass the `-h` or `--help` option to the `fastqc_reporter` script.

Terminal

```
>>$ python3 fastqc_reporter.py ./data/fastqc_data1.txt ./solution1/ -h
```

Output

```
usage: fastqc_reporter.py [-h] [-b] [-t] [-s] [-c] [-g] [-n] [-l] [-d] [-o] [-p] [-k] [-a] FastQC file input path FastQC output folder path

This script parses FastQC text files and makes reports and plots based on the options passed to it.

positional arguments:
  FastQC file input path
                        FastQC input file path.
  FastQC output folder path
                        FastQC output folder path.

options:
  -h, --help            show this help message and exit
  -b, --per_base_seq_qual
                        Process the per base sequence quality section
  -t, --per_tile_seq_qual
                        Process the Process the per tile sequence quality section
  -s, --per_seq_qual_scores
                        Process the Per sequence quality scores section
  -c, --per_base_seq_content
                        Process the Per base sequence content section
  -g, --per_seq_GC_cont
                        Process the Per sequence GC content section
  -n, --per_base_N_cont
                        Process the Per base N content section
  -l, --seq_len_dist    Process the Sequence Length Distribution section
  -d, --seq_dup         Process the Sequence Duplication Levels section
  -o, --over_seq        Process the Overrepresented sequences section
  -p, --adap_cont       Process the Adapter Content section
  -k, --kmer_count      Process the K-mer Content section
  -a, --all             Process all the sections
```

Figure 5: The output of running the help option to fastqc_reporter

Each section is created from the base class `Section.` This class defines attributes and methods to create a report.txt file and flag.txt file.

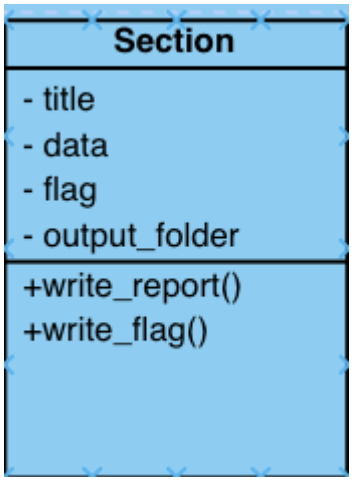

Figure 6: Class diagram for the `Section` class

Individual sections then extend the `Section` class and add a method to plot the graphs based on the section data.

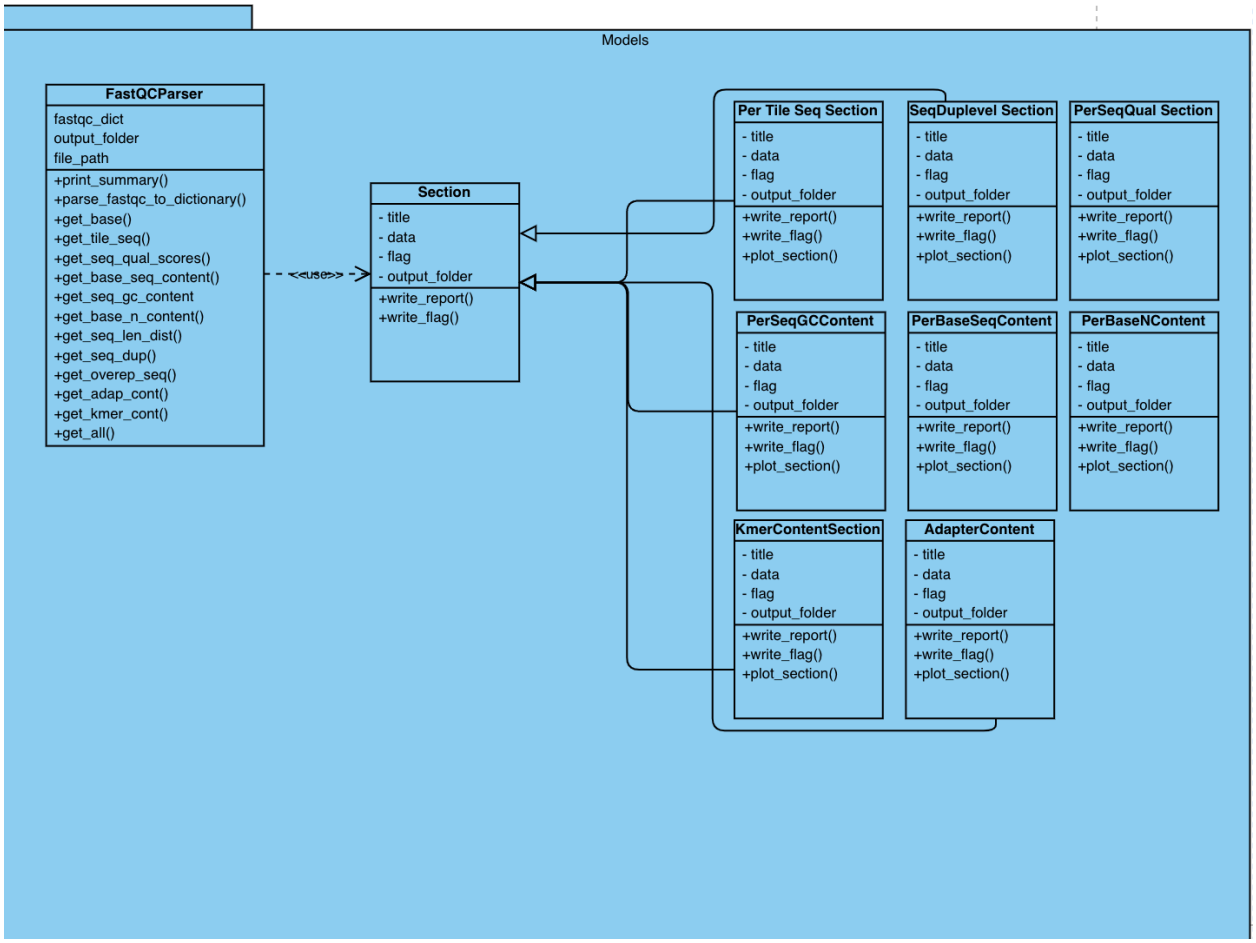

Figure 7: Class diagram for the `Models` package depicting inheritance per specific section.

Folder Structure

The fastqc reporter is structured into four main files and folders;

- The `model` folder.
- The `constants.py` file.
- The `fastqc_reporter.py` file.
- The `data` folder.

The entry point is the `fastqc_reporter.py` file, it defines all the parser options and calls the appropriate function to handle the options passed. It uses the `FastQCParser` class to create the parser instance to handle the options provided.

The `constants.py` file creates the titles of each section. This is necessary for maintainability as it lets us have a single point to change the titles if needed since they are used in several places across the program.

The `model` folder contains all the classes used in the script. It is packaged as a module with the `__init__.py` file, with this, the `fastqc_reporter.py` file can access all the classes under the `model` namespace.

The data folder contains the example test fastqc files used to test the functionality of the `fastqc_reporter.py` script.

The picture below shows the folder structure for the core functionality of the fastqc reporter.

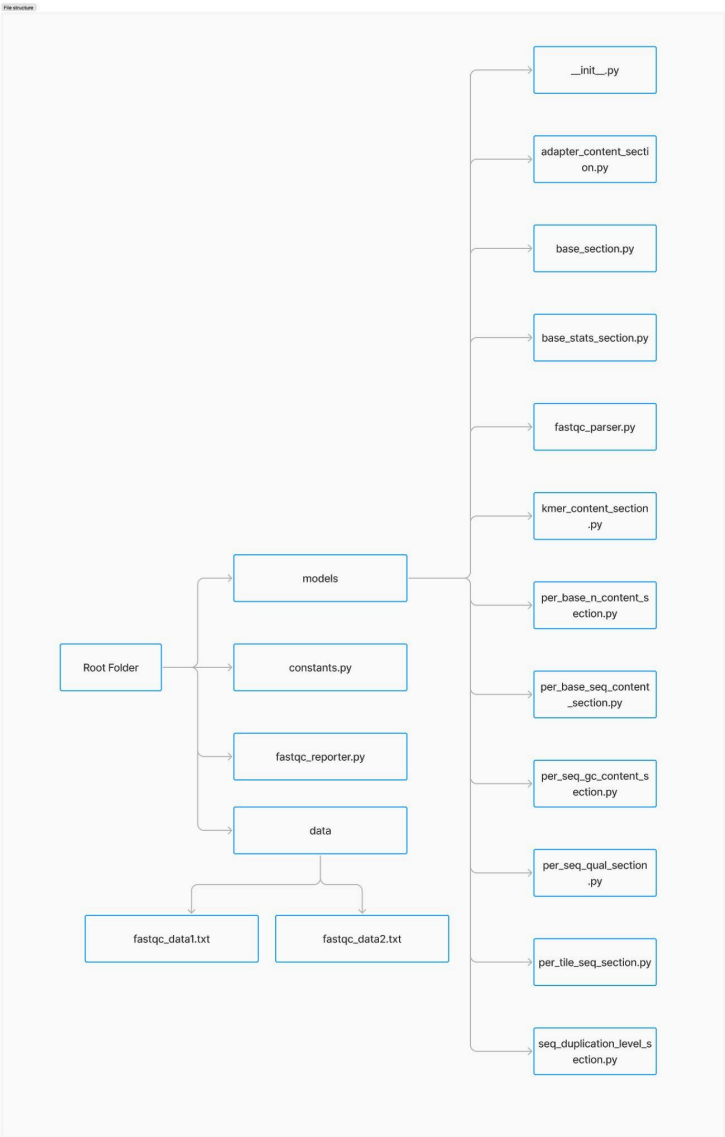

Figure 8: Image showing the folder structure for the fastqc reporter script.

Dependencies Used

Matplotlib was used together with Seaborn to plot the graphs for each section. Seaborn's plotting functions were used for different kinds of plots as appropriate (Seaborn Documentation, n.d.).

Pandas Library was used to extract the data in each section into a data frame. The `pandas.read_csv()` method was used to read the report.txt files into a data frame. (Pandas Documentation, n.d.).

Example Usage

After setting up the environment and installing the necessary dependencies, the program can be run in its default form like this;

```
>>$ python3 fastqc_reporter.py ./data/fastqc_data1.txt ./solution1/
```

The first parameter is the path to the fastqc file and the second is the output folder path. The result of the base statistics is printed to the console.

```
$>>Basic Statistics  pass

#Measure        Value
Filename        4_age21_S12_L001_R2_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
```

Options and Results

There are different options per section. The user can pass an option or multiple options to specify which section to run, also the user can pass the `-a` or `--all` option to process all the sections.

The results in this report were for the fastqc test file (`fastqc_data1.txt`) located in the `data` folder. The outputs were saved to the `solution1` folder.

For each run, the basic statistics are always printed to the console;

```
$>>Basic Statistics  pass

#Measure         Value
Filename         4_age21_S12_L001_R2_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 Tile Sequence Quality Section
Option: `-t` or `--per_tile_seq_qual`

Plot Output:

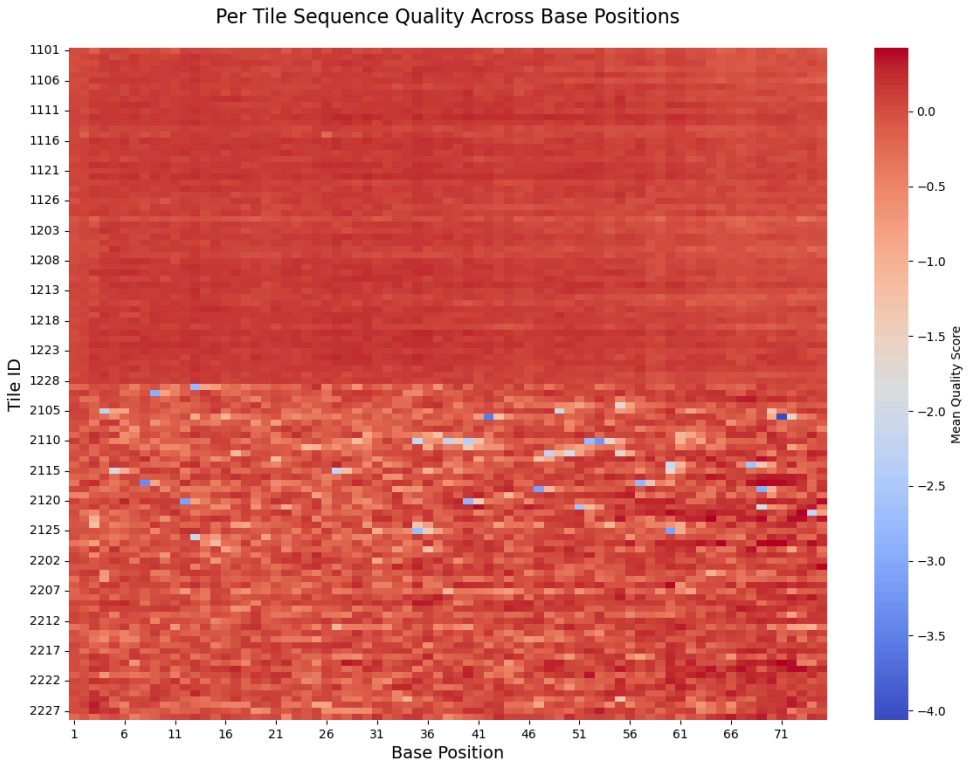

Figure 9: Heatmap showing the per-tile sequence quality across base positions.

Interpretation:

The plot shows that the mean quality scores across the tiles are high and consistent. However, there are patches of blue which indicate the lower quality regions. This can result from systemic issues with those tiles or errors introduced during sequencing (Babraham Institute, n.d.).

Per Sequence Quality Scores Section
Option: `-s` or `--per_seq_qual_scores`

Plot output

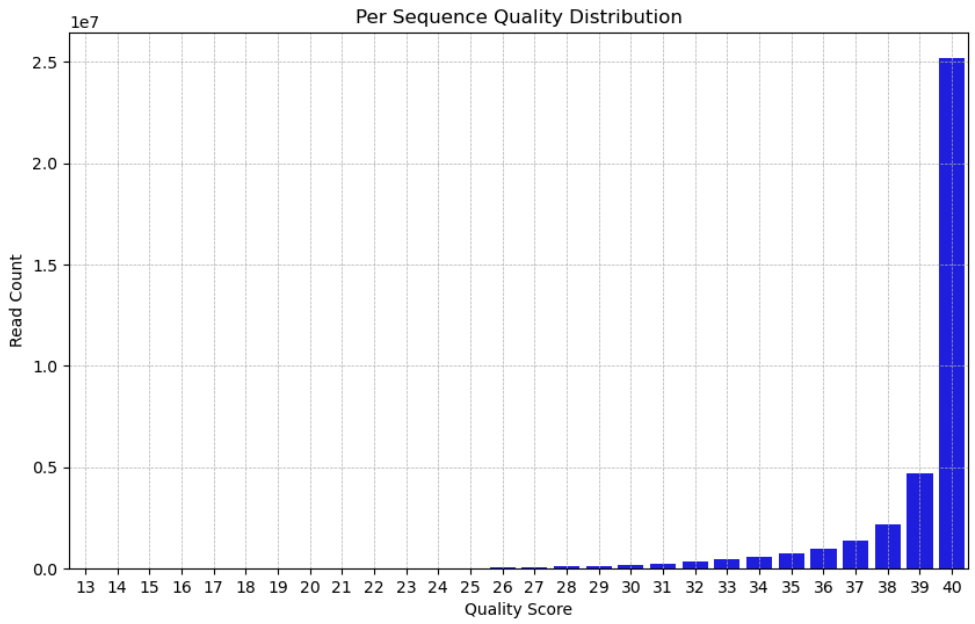

Figure 10: Barplot showing the per-tile sequence quality across base positions.

Interpretation:

The distribution of the quality scores is highly skewed to the right, indicating that most reads are high quality. A quality score of 40 indicates very confident base calls with a very low probability of error. Phreds score of 40 corresponds to an error probability of 0.0001 (O'Rawe et al., 2015).

Per base sequence content Section
Option: `-c` or `--per_base_seq_content`

Plot output:

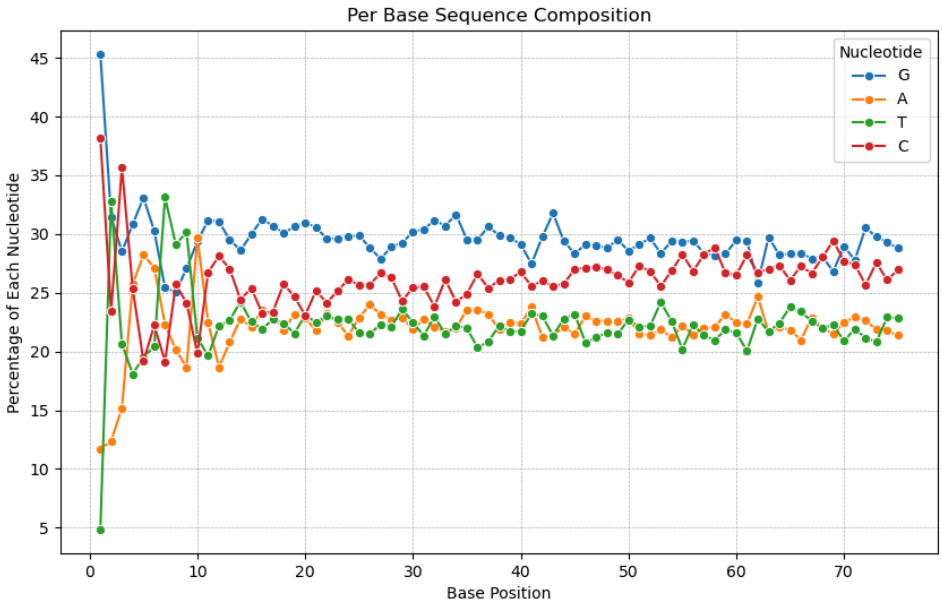

Figure 11: Line Plot showing the per-base sequence content across base positions.

Interpretation:

According to the plot above, there is a slightly higher percentage of G and C compared to A and T throughout the read length. Depending on the expected GC content of the target genome or transcriptome, this could indicate a GC bias in the library preparation or sequencing. (Illumina, 2018)

Per sequence GC content Section

Option: `-g` or `--per_seq_GC_cont`

Interpretation:

According to the plot below, The GC content distribution is highest around 55-60%, which may indicate that most reads have a GC content in this range. The GC content distribution is relatively symmetric, indicating that the library preparation and sequencing processes were successful, with minimal GC-related bias. (Illumina, 2018)

Plot output

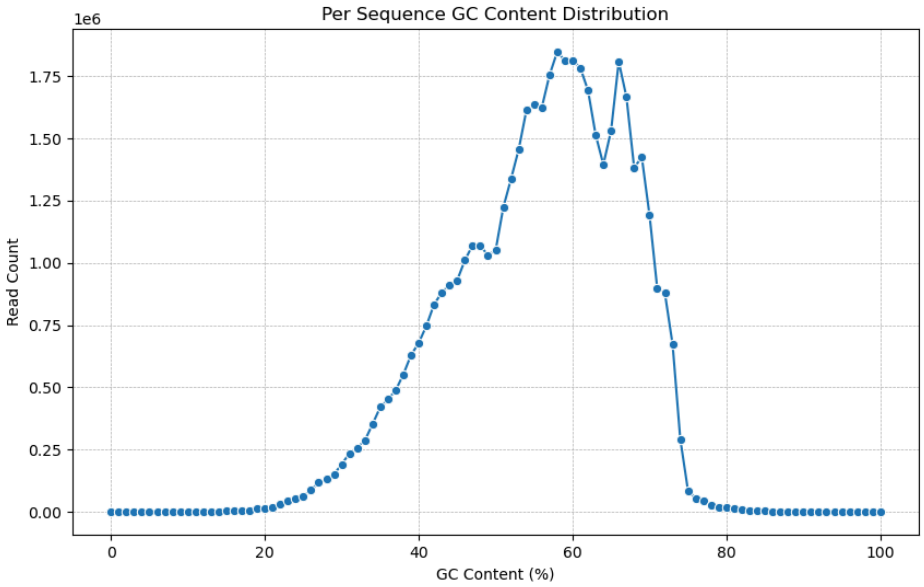

Figure 12: Line Plot showing the per-sequence GC content distribution across base positions.

Per base N content Section
Option: `-n` or `--per_base_N_cont`

Interpretation:

The line plot starts with a spike at base position 1. After the initial base, the N content drops to almost 0% and remains consistently low, indicating good sequencing quality for the rest of the read.

Trimming can be employed for the first base from each read to improve the overall quality of the data and ensure optimal performance in downstream analyses.

Plot output

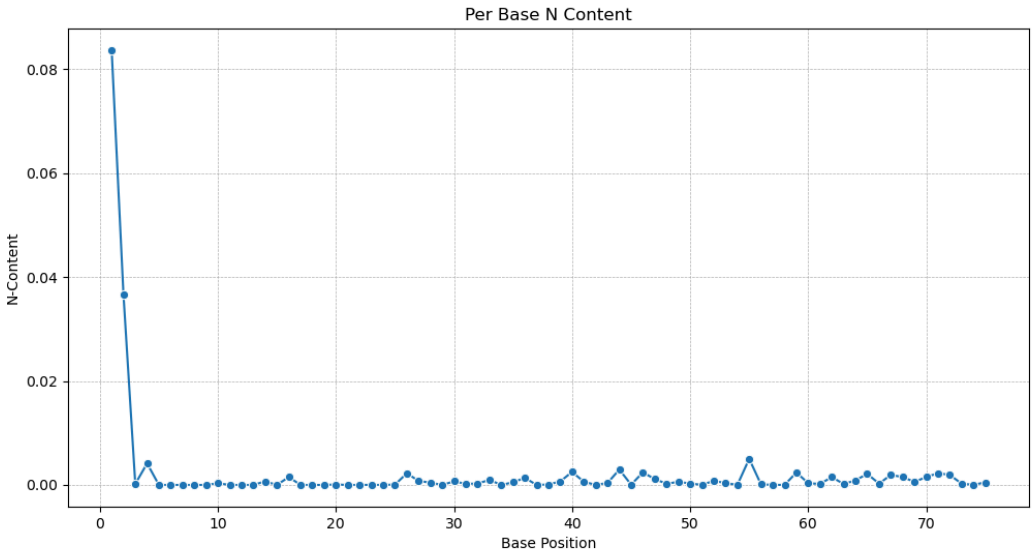

Figure 13: Line Plot showing the per base N content distribution across base positions.

Sequence Length Distribution Section
Option: `-l` or `--seq_len_dist`
Plot output: Not Applicable

Sequence Duplication Levels Section
Option: `-d` or `--seq_dup`

Interpretation:

In the plot below around 70% of the reads are unique, which is good in terms of library diversity. There is a noticeable level of duplication, with a spike in the >10 duplication category, suggesting potential PCR amplification bias. (Akalin, 2020).

Plot output:

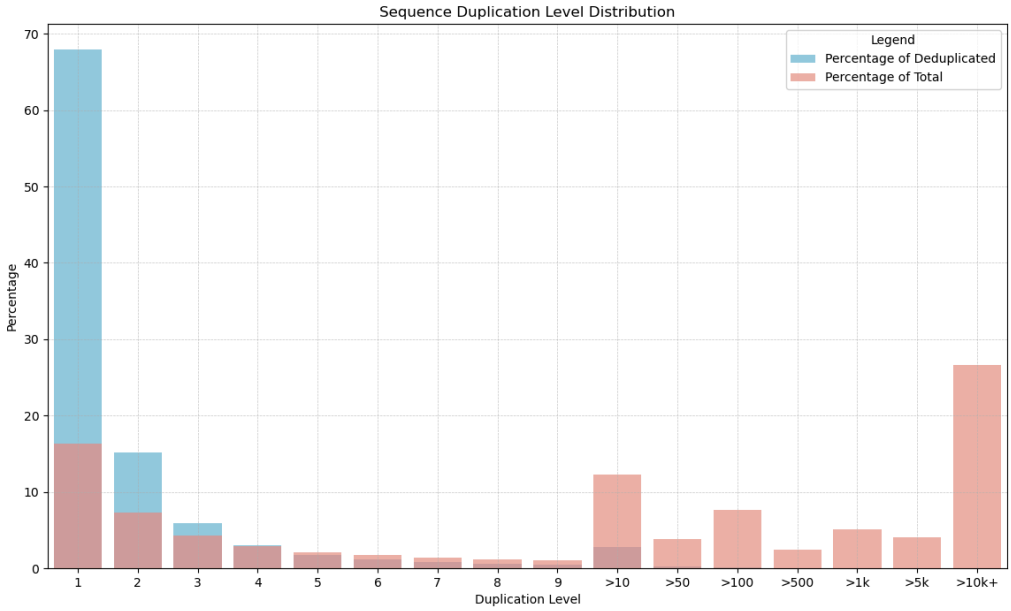

Figure 14: Bar Plot showing the sequence duplication level distribution.

Overrepresented sequences section
Option: `-o` or `--over_seq`
Plot output: Not Applicable

Adapter Content Section
Option: `-p` or `--adap_cont`

Interpretation:

The plot below shows that the Illumina Universal Adapter is present in a small but increasing percentage of reads toward the end of the sequences.

The remaining adapter types (Illumina Small RNA Adapter, Nextera Transposase Sequence, and SOLID Small RNA Adapter) have little to no presence, which indicates that there is effective trimming or that the adapters are totally absent.

Using adapter trimming tools (e.g. Btrim) is recommended to improve the quality of the data and ensure that the dataset is ready for further processing without bias or artifacts. (Kong, 2011).

Plot output:

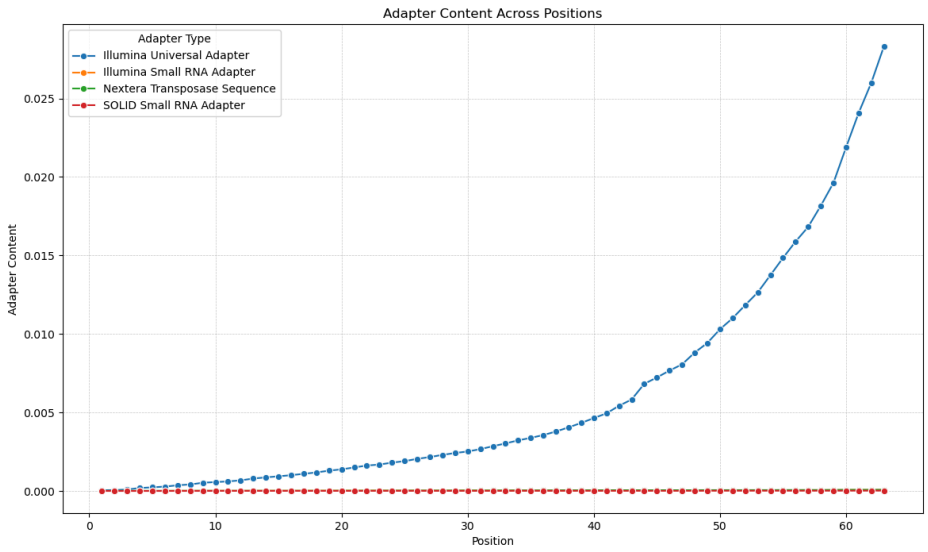

Figure 15: Line plot showing the Adapter content across positions.

K-mer Content Section
Option: `-k` or `--kmer_count`

Interpretation:

The plot indicates that certain k-mers, such as CCCACGT and CGGGCAT, are highly over-represented, with counts exceeding 10,000 occurrences.

This over-representation could indicate adapter contamination, PCR bias, or low library complexity. (Akalin, 2020).

Plot output:

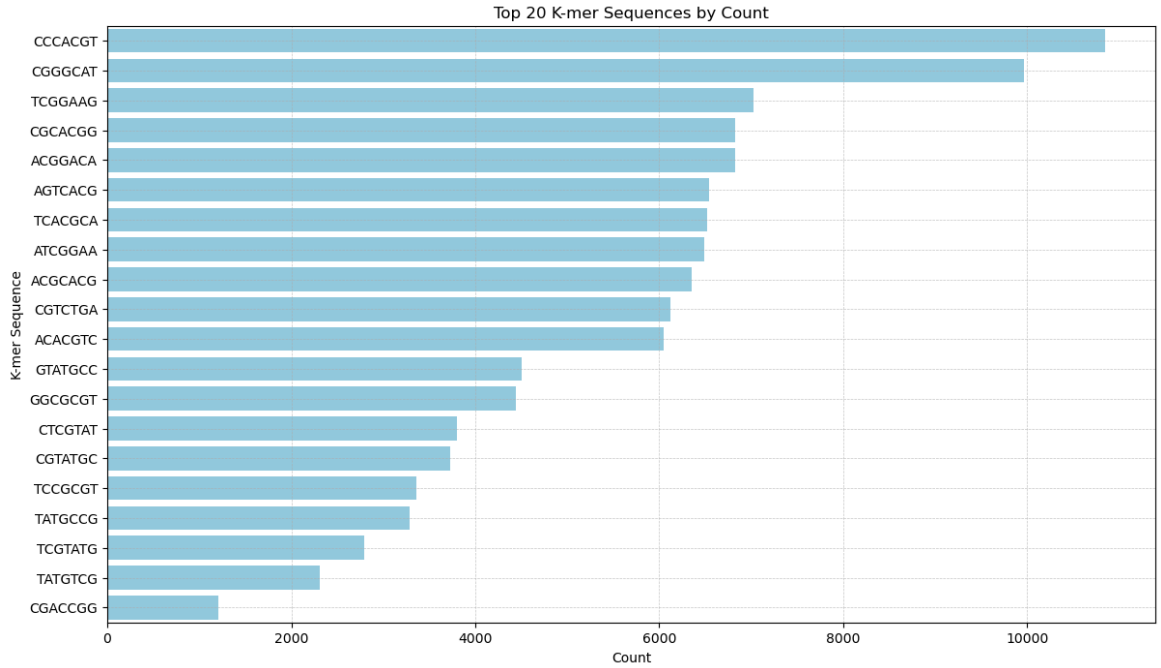

Figure 16: Line plot showing the Kmer content across positions.

All Sections
Option: `-a` or `--a`
Plot output: runs all the plots for each section where applicable, as described above.

Error Handling

Errors were handled with the native Python `try`: `except` construct. Errors that may arise from bad user input, malformed fastqc file data, file writing permissions, and file parsing are handled appropriately.

The program will exit with a non-zero exit code if it encounters an error and will print the error to the console.

References

Akalin, A. (2020). *Computational genomics with R*
(Chapter 7: Quality check, processing and alignment of high-throughput sequencing reads). Bookdown.
https://compgenomr.github.io/book/quality-check-on-sequencing-reads.html

Babraham Institute. (n.d.).
FastQC per tile sequence quality analysis.
https://www.bioinformatics.babraham.ac.uk/projects/fastqc/Help/3%20Analysis%20Modules/12%20Per%20Tile%20Sequence%20Quality.html

Kong, Y. (2011).
Btrim: A fast, lightweight adapter and quality trimming program for next-generation sequencing technologies. *Genomics, 98*(2), 152-153.
https://doi.org/10.1016/j.ygeno.2011.05.009

Illumina. (2018, September 26).
Ask a scientist - What is GC-Bias? [Video]. YouTube.
https://www.youtube.com/watch?v=wdEb3chYFOw

O'Rawe, J. F., Ferson, S., & Lyon, G. (2015, February 1).
Accounting for uncertainty in DNA sequencing data. *Trends in Genetics, 31*(2), 61–66.
https://doi.org/10.1016/j.tig.2014.12.002

Pandas Documentation. (n.d.).
pandas.read_csv. In *Pandas documentation*.
https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html#pandas.read_csv

Seaborn Documentation. (n.d.).
Overview of seaborn plotting functions. In *Seaborn documentation*.
https://seaborn.pydata.org/tutorial/function_overview.html