My Project Documentation

API Reference

Technical Documentation.

It is recommended to open this via browser: https://ms2206.github.io/FastQCParser/api

---

Class Definitions

`ParsedArg`

Bases: `dict`

A class to represent parsed command-line arguments for sequence quality analysis.

This class maps command-line arguments to their corresponding search string and manages the required output types based on the argument.

Attributes:

Name	Type	Description
`cli_argument`	`str`	The command-line argument provided by the user.
`value`	`bool`	The bool command-line argument from 'store_true'
`required_outputs`	`list`	A list of required output types, defaulting to ['R', 'P', 'F'].
`search_string`	`str`	Search string to extract from FASTQC - retrieved from the arg_key_map.

▼ **Class Attributes**

arg_key_map (dict): A mapping of command-line argument keys to their descriptions.

Methods:

Name	Description
configure_outputs	Adjusts the required outputs based on the command-line argument. If the argument is 'seq_len_dist' or 'over_seq', 'P' is removed from the required outputs.

▼ **Example**

> > > parsed_arg = ParsedArg('per_base_seq_qual', 'some_value') parsed_arg.configure_outputs() ['R', 'P', 'F']

▶ **Source code in `src/class_definitions.py`**

`configure_outputs()`

Removes 'P' from ['seq_len_dist', 'over_seq'] :return:

▶ **Source code in `src/class_definitions.py`**

Parsing

`extract_raw(filepath, search_string)`

Extracts a section of a CSV file based on a search string and returns it as a DataFrame.

This function reads a CSV file into a DataFrame, searches for a specified string within the 'raw' column, and extracts rows from the first occurrence of the search string to the corresponding 'END_MODULE' marker. The function uses a predefined mapping to determine the correct 'END_MODULE' for each search string.

Parameters: filepath (str): The path to the CSV file to be read. search_string (str): The string to search for within the 'raw' column of the DataFrame.

Returns: DataFrame: A DataFrame containing the rows from the first occurrence of the search string to the corresponding 'END_MODULE'.

Raises: FileNotFoundError: If the specified file does not exist, the function prints an error message and exits the program.

Example:

> > > extract_raw('path/to/your/file.csv', 'Basic Statistics')

▶ **Source code in** `src/parser.py`

`handle_cli_args(args)`

This function further parses args from the ArgParse command line input. It unpacks all args with store_true. Then initiates and configures an instance of Class::ParsedArg for each optional arg that was supplied by the user.

:param args: cli arguments passed from user :return: a list of ParsedArg objects that have been configured to have their "Required outputs" adjusted based on expected output property.

▶ **Source code in** `src/parser.py`

`parse_raw(raw_data)`

Parses raw data from extract_raw().

This function further processes a DataFrame from extract_raw(). It gathers all lines that do not start with '#' and splits on tab characters. These are the values from the FASTQC file. It also extracts the headers as lines that start with '#' and uses these two lists to return a pandas DataFrame.

Parameters: raw_data (DataFrame): The output from extract_raw()

Returns: DataFrame: A formatted DataFrame with the extracted values and headers.

Example:

> > > raw_data = extract_raw('data/raw/fastqc_data1.txt', 'Sequence Duplication Levels') formatted_df = parse_raw(raw_data) print(formatted_df)

▶ **Source code in** `src/parser.py`

Plotting

`plot_adap_cont(cli_arg, output_dir)`

Plot and save plot_adap_cont to filepath :return: None

▶ **Source code in** `src/plotter.py`

`plot_kmer_cont(cli_arg, output_dir)`

Plot and save per_base_N_cont to filepath :return: None

▶ **Source code in** `src/plotter.py`

`plot_per_base_N_cont(cli_arg, output_dir)`

Plot and save per_base_N_cont to filepath :return: None

▶ **Source code in** `src/plotter.py`

`plot_per_base_seq_content(cli_arg, output_dir)`

Plot and save per_base_seq_content to filepath :return: None

▶ **Source code in** `src/plotter.py`

`plot_per_base_seq_qual(cli_arg, output_dir)`

Plot and save per_base_seq_qual to filepath :return: None

▶ **Source code in** `src/plotter.py`

`plot_per_seq_GC_cont(cli_arg, output_dir)`

Plot and save per_base_seq_content to filepath :return: None

▶ **Source code in** `src/plotter.py`

`plot_per_seq_qual_scores(cli_arg, output_dir)`

Plot and save per_seq_qual_scores to filepath :return: None

▶ **Source code in** `src/plotter.py`

`plot_per_tile_seq_qual(cli_arg, output_dir)`

Plot and save per_tile_seq_qual to filepath :return: None

▶ **Source code in** `src/plotter.py`

`plot_seq_dup(cli_arg, output_dir)`

Plot and save per_base_N_cont to filepath :return: None

▶ **Source code in** `src/plotter.py`

`plot_seq_len_dist(cli_arg, output_dir)`

Plot and save per_base_N_cont to filepath :return: None

▶ **Source code in** `src/plotter.py`

My Project Documentation

API Reference

Technical Documentation.

It is recommended to open this via browser: https://ms2206.github.io/FastQCParser/api

---

Class Definitions

`ParsedArg`

Bases: `dict`

A class to represent parsed command-line arguments for sequence quality analysis.

This class maps command-line arguments to their corresponding search string and manages the required output types based on the argument.

Attributes:

Name	Type	Description
`cli_argument`	`str`	The command-line argument provided by the user.
`value`	`bool`	The bool command-line argument from 'store_true'
`required_outputs`	`list`	A list of required output types, defaulting to ['R', 'P', 'F'].
`search_string`	`str`	Search string to extract from FASTQC - retrieved from the arg_key_map.

▼ **Class Attributes**
arg_key_map (dict): A mapping of command-line argument keys to their descriptions.

Methods:

Name	Description
configure_outputs	Adjusts the required outputs based on the command-line argument. If the argument is 'seq_len_dist' or 'over_seq', 'P' is removed from the required outputs.

▼ **Example**

> > > parsed_arg = ParsedArg('per_base_seq_qual', 'some_value') parsed_arg.configure_outputs() ['R', 'P', 'F']

▶ **Source code in `src/class_definitions.py`**

`configure_outputs()`

Removes 'P' from ['seq_len_dist', 'over_seq'] :return:

▶ **Source code in `src/class_definitions.py`**

Parsing

`extract_raw(filepath, search_string)`

Extracts a section of a CSV file based on a search string and returns it as a DataFrame.

This function reads a CSV file into a DataFrame, searches for a specified string within the 'raw' column, and extracts rows from the first occurrence of the search string to the corresponding 'END_MODULE' marker. The function uses a predefined mapping to determine the correct 'END_MODULE' for each search string.

Parameters: filepath (str): The path to the CSV file to be read. search_string (str): The string to search for within the 'raw' column of the DataFrame.

Returns: DataFrame: A DataFrame containing the rows from the first occurrence of the search string to the corresponding 'END_MODULE'.

Raises: FileNotFoundError: If the specified file does not exist, the function prints an error message and exits the program.

Example:

>>> extract_raw('path/to/your/file.csv', 'Basic Statistics')

▶ **Source code in** `src/parser.py`

`handle_cli_args(args)`

This function further parses args from the ArgParse command line input. It unpacks all args with store_true. Then initiates and configures an instance of Class::ParsedArg for each optional arg that was supplied by the user.

:param args: cli arguments passed from user :return: a list of ParsedArg objects that have been configured to have their "Required outputs" adjusted based on expected output property.

▶ **Source code in** `src/parser.py`

`parse_raw(raw_data)`

Parses raw data from extract_raw().

This function further processes a DataFrame from extract_raw(). It gathers all lines that do not start with '#' and splits on tab characters. These are the values from the FASTQC file. It also extracts the headers as lines that start with '#' and uses these two lists to return a pandas DataFrame.

Parameters: raw_data (DataFrame): The output from extract_raw()

Returns: DataFrame: A formatted DataFrame with the extracted values and headers.

Example:

>>> raw_data = extract_raw('data/raw/fastqc_data1.txt', 'Sequence Duplication Levels') formatted_df = parse_raw(raw_data) print(formatted_df)

▶ **Source code in** `src/parser.py`

Plotting

`plot_adap_cont(cli_arg, output_dir)`

Plot and save plot_adap_cont to filepath :return: None

▶ **Source code in** `src/plotter.py`

`plot_kmer_cont(cli_arg, output_dir)`

Plot and save per_base_N_cont to filepath :return: None

▶ **Source code in** `src/plotter.py`

`plot_per_base_N_cont(cli_arg, output_dir)`

Plot and save per_base_N_cont to filepath :return: None

▶ **Source code in** `src/plotter.py`

`plot_per_base_seq_content(cli_arg, output_dir)`

Plot and save per_base_seq_content to filepath :return: None

▶ **Source code in** `src/plotter.py`

`plot_per_base_seq_qual(cli_arg, output_dir)`

Plot and save per_base_seq_qual to filepath :return: None

▶ **Source code in** `src/plotter.py`

`plot_per_seq_GC_cont(cli_arg, output_dir)`

Plot and save per_base_seq_content to filepath :return: None

▶ **Source code in** `src/plotter.py`

`plot_per_seq_qual_scores(cli_arg, output_dir)`

Plot and save per_seq_qual_scores to filepath :return: None

▶ **Source code in** `src/plotter.py`

`plot_per_tile_seq_qual(cli_arg, output_dir)`

Plot and save per_tile_seq_qual to filepath :return: None

▶ **Source code in** `src/plotter.py`

`plot_seq_dup(cli_arg, output_dir)`

Plot and save per_base_N_cont to filepath :return: None

▶ **Source code in** `src/plotter.py`

`plot_seq_len_dist(cli_arg, output_dir)`

Plot and save per_base_N_cont to filepath :return: None

▶ **Source code in** `src/plotter.py`
