preqc-lr Documentation

Release 0.1

Simpson Lab

Sep 10, 2018

Contents

preqc-lr is a software tool that performs quality control checks on long read sequencing data.

CHAPTER 1

What is preqc-lr?

With the emergence of new long read sequencing technology such as Pacbio Single Molecule, Real-Time (SMRT) Sequencing technology and Oxford Nanopore Technologies (ONT), there is a need for a method that assesses sequencing quality prior to analyses. Prior to genome assembly, preqc-lr can be used to infer quality statistics without alignment to a reference genome.

There are two components to preqc-lr:

1. Calculate
2. Report

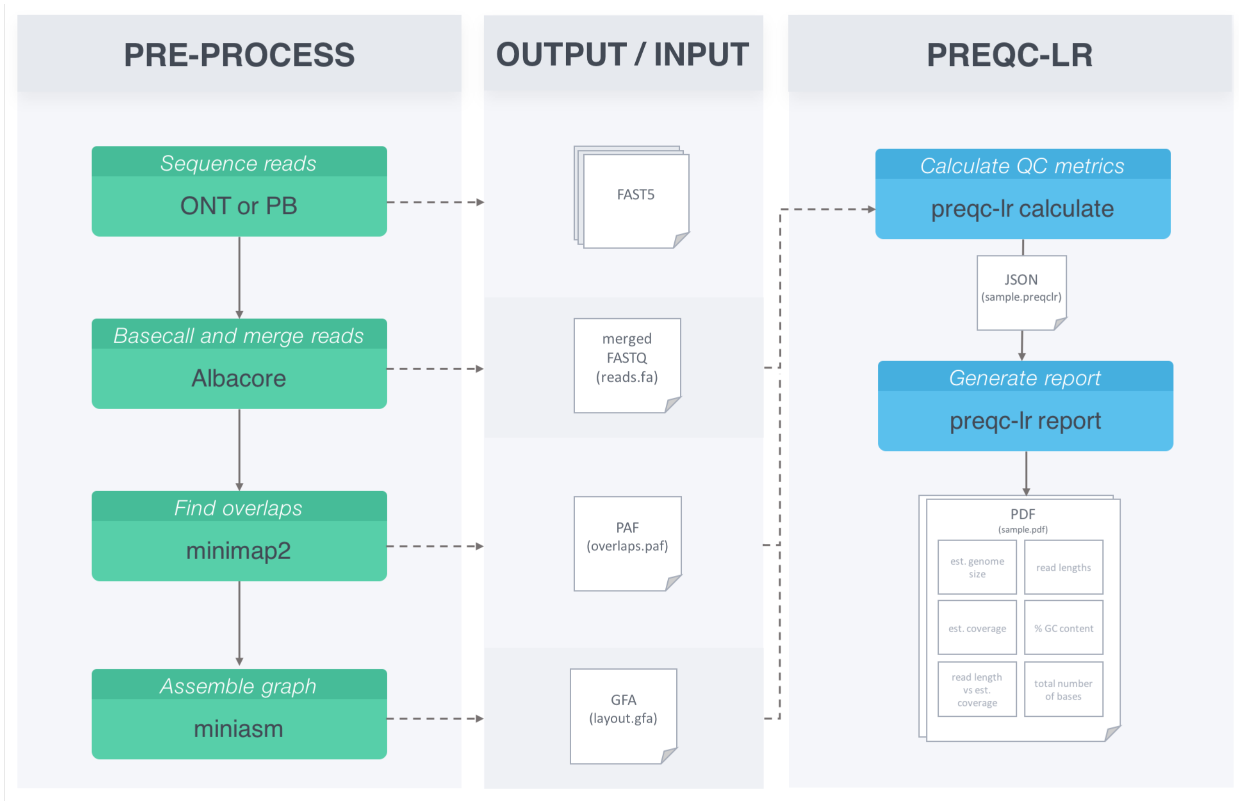
1.1 Calculate

The first tool will calculate all the datasets needed to create plots using overlap inform ation provided by minimap2.

1.2 Report

The second tool reads the calculated output and generates a pdf with the following plots:

1. Estimated genome size
2. Read length distribution
3. Estimated coverage distribution
4. Per read GC content distribution
5. Estimated coverage vs read length
6. Total number of bases as a function of minimum read length
7. NG(X)


PRE-PROCESS
OUTPUT / INPUT
PREQC-LR
Sequence reads
ONT or PB
Basecall and merge reads
Albacore
Find overlaps
minimap2
Assemble graph
miniasm
FAST5
merged FASTQ (reads.fa)
PAF (overlaps.paf)
GFA (layout.gfa)
Calculate QC metrics
preqc-lr calculate
JSON (sample.preqclr)
Generate report
preqc-lr report
PDF (sample.pdf)
est. genome size
read lengths
est. coverage
% GC content
read length vs est. coverage
total number of bases

CHAPTER 2

Installation

2.1 Requirements

To generate files needed:

- minimap2 (required to create required input PAF file)
- miniasm (required if NG(X) plots requested)

For the calculation step:

- C++ compiler with C++11 support

For the report generation step:

- Python 2.7.11
- setuptools - required for installation
- BioPython
- matplotlib v2.0.0

2.2 Installing the latest code from github

```
git clone --recursive  https://github.com/simpsonlab/preqc-lr.git
cd preqc-lr
make
```

2.3 Installing dependencies

First we need to make sure we have everything to properly use pip or the setup.py script.

```
# create virtual environment
virtualenv preqclr-venv
source preqclr-venv/bin/activate

# check that you are using correct environment
which pip

# check that setuptools is installed
pip freeze

# update setuptools if needed
python -m pip install --upgrade pip setuptools

# check that we are using the right version of python (2.7.11+)
python -V
```

Okay, we are ready to install dependencies.

```
# download report script dependencies
python setup.py install

# OR we can use pip
pip install preqc-lr
```

To check that we have installed all the packages and the right versions we run *pip freeze* and we should see the following:

```
biopython==1.70
cycler==0.10.0
functools32==3.2.3.post2
gevent==1.3a1
greenlet==0.4.13
matplotlib==2.0.0
numpy==1.14.0
preqc-lr==2.0
pyparsing==2.2.0
python-dateutil==2.6.1
pytz==2017.3
six==1.11.0
subprocess32==3.5.0rc1
```

CHAPTER 3

Quickstart

Time: 10 minutes

preqc-lr generates a PDF report containing several plots such as estimated genome size and coverage. This report can be used to evaluate the quality of your sequencing data. Here, we provide a step-by-step tutorial to get you started!

Requirements:

- preqc-lr v2.0
- minimap2 v2.6
- miniasm v0.2

3.1 Download example dataset

You can download the example dataset we will use here:

```
wget http://s3.climb.ac.uk/nanopolish_tutorial/preqclr_example_data.tar.gz
tar -xf preqclr_example_data.tar.gz
cd example_data/
```

Details:

This dataset from an *E. coli* sample were produced using Oxford Nanopore Technologies (ONT) MinION sequencer.

- Sample : E. coli str. K-12 substr. MG1655
- Instrument : ONT MinION sequencing R9.4 chemistry
- Basecaller : Albacore v2.0.1
- Number of reads: 63931

3.2 Generate overlap information with minimap2

We use minimap2 to find overlaps between our ONT long reads:

```
minimap2 -x ava-ont albacore_v2.0.1-merged.fasta albacore_v2.0.1-merged.fasta >␣
↪overlaps.paf
```

If we take a peek at the first few lines of the Pairwise mApping Format (PAF) file, we see the following:

```
7fd051aa-c88b-4cf7-8846-cc2117780be2_Basecall_1D_template  6605    118     6425    - ␣
↪     ae8fc44b-ee05-4c7a-a611-483bb408cb9e_Basecall_1D_template       7834    629    ␣
↪ 7230    24806671        0       tp:A:S  cm:i:387        s1:i:2413       dv:f:0.1144
7fd051aa-c88b-4cf7-8846-cc2117780be2_Basecall_1D_template  6605    343     6417    - ␣
↪     cecc6ee9-f1ec-4c82-915a-5312f39f7ec5_Basecall_1D_template       6762    421    ␣
↪ 6710    24286372        0       tp:A:S  cm:i:370        s1:i:2374       dv:f:0.1149
7fd051aa-c88b-4cf7-8846-cc2117780be2_Basecall_1D_template  6605    118     6377    - ␣
↪     c0d8087f-ad9f-430c-8094-24c6187bed6c_Basecall_1D_template       11415   3039   ␣
↪ 9493    22646559        0       tp:A:S  cm:i:346        s1:i:2209       dv:f:0.1214
7fd051aa-c88b-4cf7-8846-cc2117780be2_Basecall_1D_template  6605    738     6422    - ␣
↪     bbb93738-16ec-4bcd-86e5-31e852946a7d_Basecall_1D_template       6596    553    ␣
↪ 6498    20916000        0       tp:A:S  cm:i:302        s1:i:2031       dv:f:0.1242
7fd051aa-c88b-4cf7-8846-cc2117780be2_Basecall_1D_template  6605    212     6422    - ␣
↪     943b8d89-2ee5-4d67-91d1-a94772afed31_Basecall_1D_template       7324    807    ␣
↪ 7152    20676448        0       tp:A:S  cm:i:322        s1:i:2011       dv:f:0.1255
```

You can find more information about the format of the PAF file here.

3.3 Generate assembly graph with miniasm

We use miniasm to get an assembly graph in the Graphical Fragment Assembly format:

```
miniasm -f albacore_v2.0.1-merged.fasta overlaps.paf > layout.gfa
```

Note: Make sure `layout.gfa` and `overlaps.paf` are not empty before continuing.

3.4 Perform calculations

We now have the necessary files to run preqc-lr (`albacore_v2.0.1-merged.fasta`, `overlaps.paf`, and `layout.gfa`). To generate the data needed for the report we first run preqc-lr-calculate

```
./preqclr \
    --reads albacore_v2.0.1-merged.fasta \
    --sample_name ecoli.ONT \
    --paf overlaps.paf \
    --gfa layout.gfa \
    --verbose
```

This will produce a JSON formatted file (`ecoli.ONT.preqclr`) and a log of calculations that were performed (`ecoli.ONT_preqclr-calculate.log`).

3.5 Generate report

Now we are ready to run preqclr-report to generate a PDF file describing quality metrics of the sequencing data:

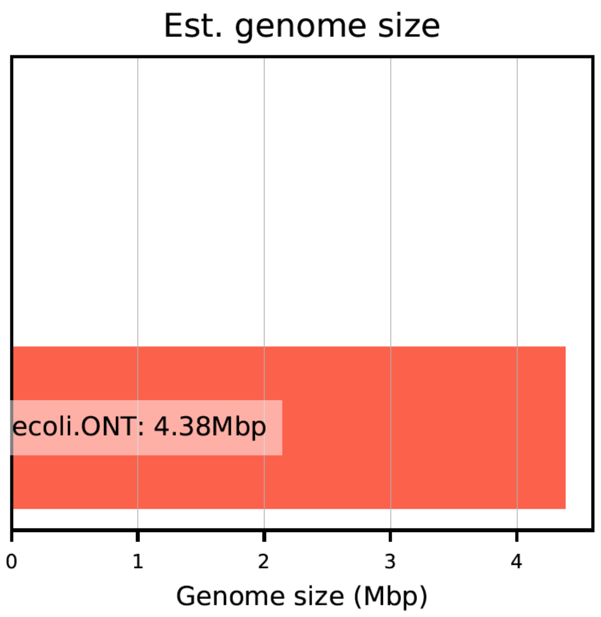
```
python preqclr-report.py \
    -i ecoli.ONT.preqclr --verbose
```

This will produce a PDF file: `ecoli.ONT.pdf`.

3.6 Example report

The report produces plots as seen below.

Plot 0:



Plot 1:

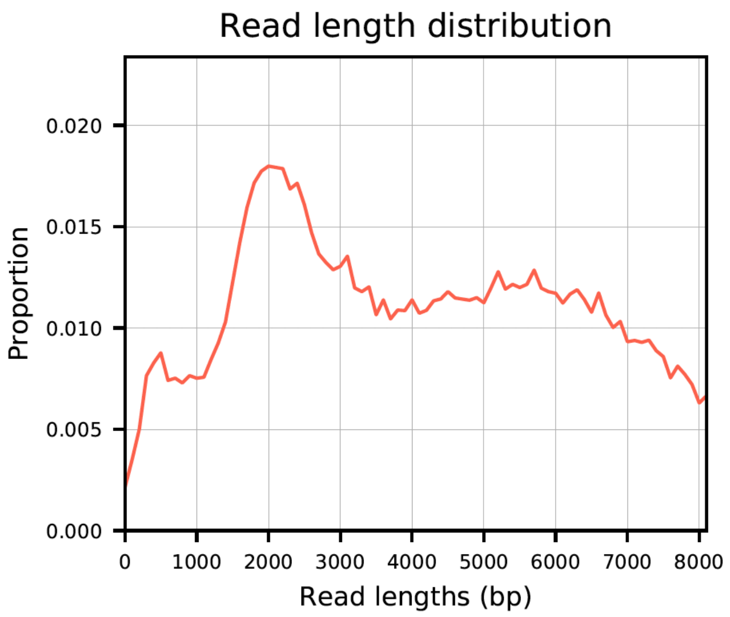
Plot 2:

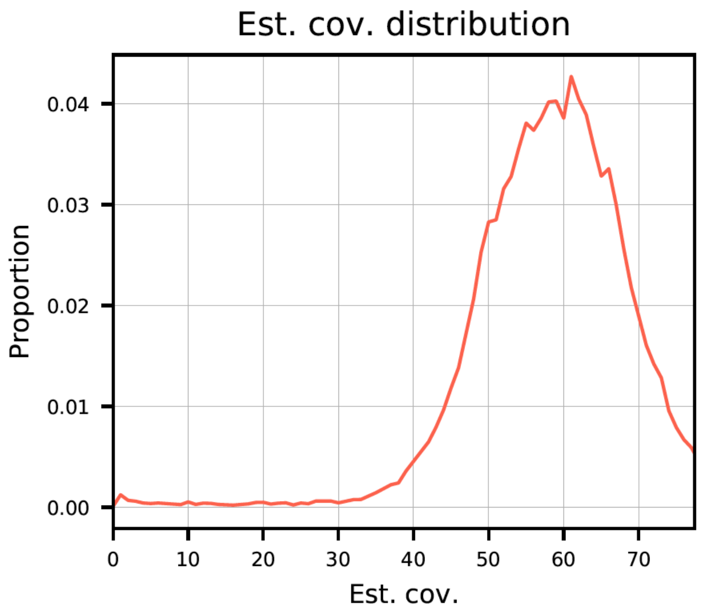
Plot 3:

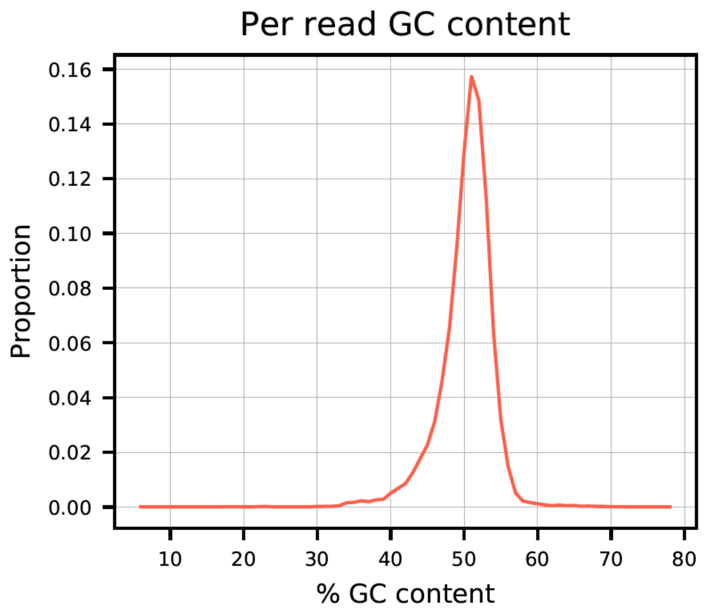
Plot 4:

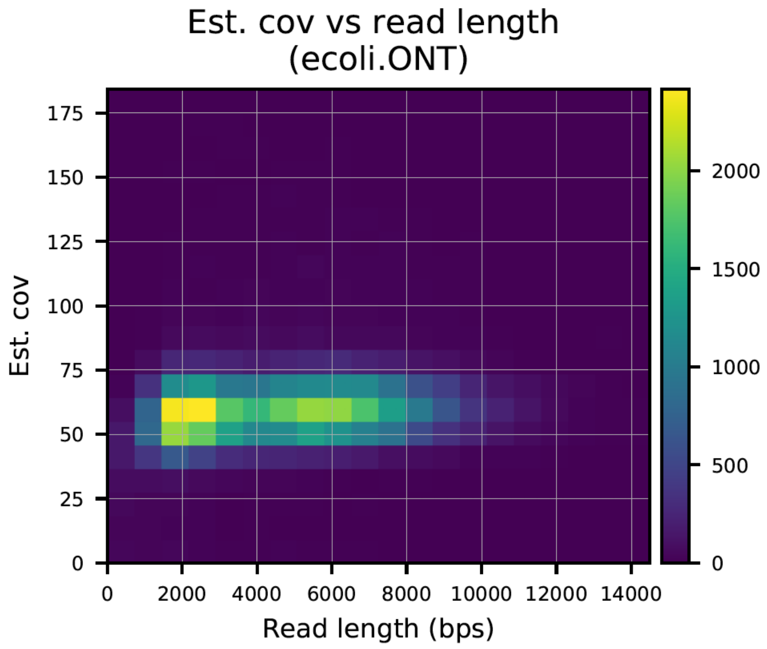
Plot 5:

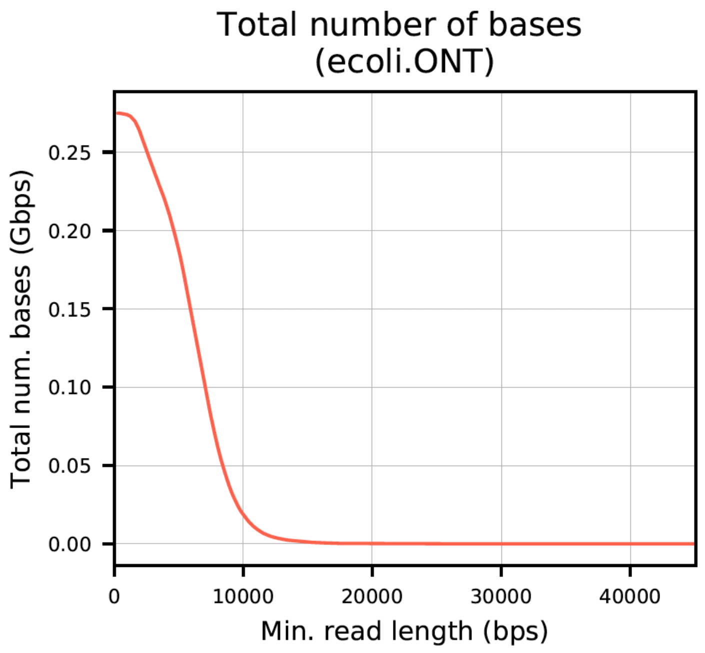
Plot 6:

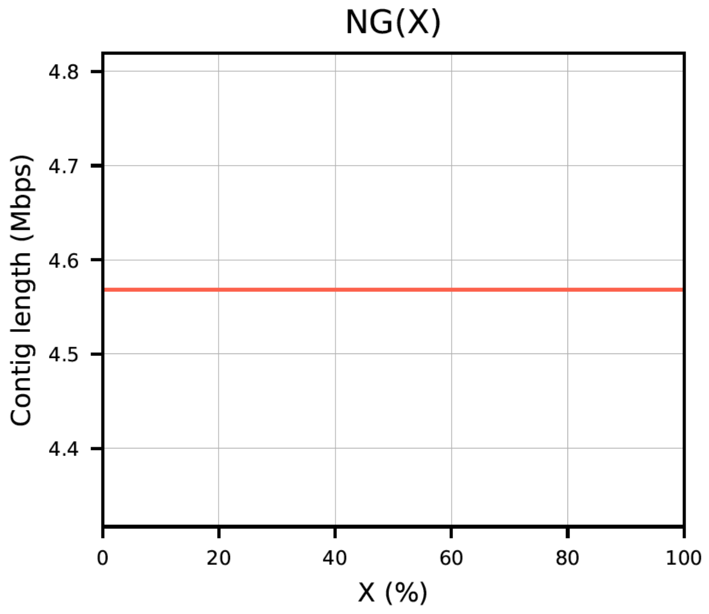

Read length distribution
Proportion
0.000
0.005
0.010
0.015
0.020
0
1000
2000
3000
4000
5000
6000
7000
8000
Read lengths (bp)


Est. cov. distribution
Proportion
0.00
0.01
0.02
0.03
0.04
0
10
20
30
40
50
60
70
Est. cov.


Per read GC content
Proportion
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
10
20
30
40
50
60
70
80
% GC content


Est. cov vs read length
(ecoli.ONT)
Est. cov
175
150
125
100
75
50
25
0
0
2000
4000
6000
8000
10000
12000
14000
Read length (bps)
2000
1500
1000
500
0


Total number of bases
(ecoli.ONT)
Total num. bases (Gbps)
0.25
0.20
0.15
0.10
0.05
0.00
0
10000
20000
30000
40000
Min. read length (bps)


NG(X)
Contig length (Mbps)
4.8
4.7
4.6
4.5
4.4
0
20
40
60
80
100
X (%)

CHAPTER 4

Manual

4.1 Calculate

4.1.1 Overview

Generates data needed to create plots in preqc-lr-report.

4.1.2 Input

- READS file: long read sequences in fasta or fastq format
- PAF file: information on overlaps between reads in READS file
- GFA file: graph assembly that contains contig information

4.1.3 Output

- JSON file containing data needed to generate plots in preqclr-report
- Log file summarizing statistics calculated, input, and output

4.1.4 Usage example

```
./preqclr [-h/--help] -r/--reads <fasta|fastq|fasta.gz|fastq.gz> \
        -n/--sample_name sample_name \
        -p/--paf <PAF> -g/--gfa <GFA> \
        --rlen_cutoff INT \
        --verbose -v/--version
```

Argument name(s)	Required	Default value	Description
`-r`, `--reads`	Y	NA	Fasta, fastq, fasta.gz, or fastq.gz files containing reads.
`-n`, `--sample_name`	Y	NA	Sample name; you can use the name of species for example. This will be used as output prefix.
`-p`, `--paf`	N	NA	Minimap2 Pairwise mApping Format (PAF) file. This is produced using `minimap2 -x ava-ont sample.fastq sample.fasta`.
`-g`, `--gfa`	N	NA	Miniasm Graph Fragment Assembly (GFA) file. This is produced using `miniasm -f reads.fasta overlaps.paf > layout.gfa`. This is required only if user wants to generate an NGX plot. If not given, it will **NOT CALCULATE NGX STATISTICS**.
`--verbose`	N	False	Use to output preqc-lr progress to stdout.

4.2 Report

4.2.1 Overview

Generates a report with plots describing QC metrics for long read data sets.

4.2.2 Input

- JSON file(s) containing data for sample(s) needed to generate plots created in preqclr calculate

4.2.3 Output

- PDF file report

Plots:

1. Estimated genome size This is a bar plot that shows the estimated genome size for one or more samples. As coverage was inferred from overlap information, we can use this to calculate genome size with Lander-Waterman statistics.
2. Read length distribution This is the distribution of read lengths calculated from the READS file. preqclr imposes an x-limit such that 90% of all of the read lengths falls under this limit. This was done to avoid extremely long tails.
3. Estimated coverage distribution This shows the distribution of coverage for each read inferred from the overlap information file (PAF).
4. Per read GC content distribution In this plot we show the distribution of GC content per read for a sample of 40% of reads. To calculate this for each read, we summed the number of C and G nucleotides then divided by the read length.
5. Total number of bases vs minimum read length We show the total number of bases with reads of a minimum length of x.
6. NGX This shows the contigiuity of the data. Miniasm produces contigs from your sequencing data. To interpret this let's look at x=50 and it's NG(50) value on the y-axis. The contig length on the y-axis describes the length at which 50% of the genome size estimate is capture in contigs with length greater than or equal to the NG(50) value.

4.2.4 Usage example

```
python preqclr-report.py [-h/--help] -i/--input <*.preqclr> \
     --save_png --list_plots -o/--output <output_prefix> --plot <list of user␣
↪specified plots> \
     --verbose
```

Argument name(s)	Required	Default value	Description
`-i, --input`	Y	NA	Output of preqclr calculate step. JSON formatted file with '.preqclr' extension.
`-o, --output`	N	If only one preqclr file given, it will infer from prefix. Else if multiple, prefix will be "preqc-lr-output".	Prefix for output PDF.
`--plot`	N	NA	Users can specify which plots they want. To do so, use `--list_plots` and use the names of plots.
`--list_plots`	N	NA	Use this to see which plots are available. Note that NGX plots are also dependent on whether or not it was calculated in preqc-lr-calculate step and this depends on whether or not miniasm's GFA file was passed as input.
`--save_png`	N	False	Use to save each subplot as a png.
`--verbose`	N	False	Use to print progress to stdout.