ATACseq_Tutorial1_Preprocessing

September 10, 2023

1 ATAC-seq Module1: Preprocessing and Quality Control

1.1 Overview & Purpose

This short tutorial demonstrates the intial processing steps for ATAC-seq analysis. In this module we focus on generating quality reports of the fastq files, adapter trimming, mapping, and removal of PCR duplicates.

In this tutorial we will process a randomly chosen published dataset. This is available from GEO: GSE67382 Bao X, Rubin AJ, Qu K, Zhang J et al. A novel ATAC-seq approach reveals lineage-specific reinforcement of the open chromatin landscape via cooperation between BAF and p63. Genome Biol 2015 Dec 18;16:284. PMID: 26683334

This dataset is paired-end 50 bp sequencing. We will analyze two samples representing NHEK cells with BAF depletion compared to a control. Note that to allow faster processing we have limited the reads to that of chromosome 4.

1.1.1 Required Files

In this stage of the module you will use the fastq files that have been prepared. In step1 we will copy these files over to your instance. You can also use this module on your own data or any published ATAC-seq dataset.

STEP1: Setup Environment

Initial items to configure your google cloud environment. In this step we will use conda to install the following packages:

Quality Reporting: fastqc, multiqc

Read Trimming: trimmomatic

Mapping: bowtie2

Deduplication: samtools, picard

[ ]:
```
#!python -m ipykernel install --user --name ATACtraining
numthreads=!lscpu | grep '^CPU(s)'| awk '{print $2-1}'
numthreadsint = int(numthreads[0])
!conda config --prepend channels bioconda
!conda install -y -c bioconda fastqc bowtie2 picard multiqc samtools trimmomatic
!pip install jupyterquiz jupytercards
from jupyterquiz import display_quiz
```

```python
from IPython.display import IFrame
from IPython.display import display
from jupytercards import display_flashcards
import pandas as pd
```

1.2 Setup FileSystem

Now lets create some folders to stay organized and copy over our prepared fastq files. We're going to create a directory called "Tutorial1" which we'll use for this module. We'll then create subfolders for our InputFiles and for the files that we'll be creating during this module. We'll also copy over the fasta file for chromosome 4 as well as some bowtie2 index files (don't worry we'll teach you how to create these index files).

[2]:
```python
#These commands create our directory structure.
#!cd $HOMEDIR
#!mkdir -p Tutorial1
#!mkdir -p Tutorial1/InputFiles
!mkdir -p Tutorial1/QC
!mkdir -p Tutorial1/Trimmed
!mkdir -p Tutorial1/Mapped
#!mkdir -p Tutorial1/RefGenome
#!mkdir -p Tutorial1/LessonImages
#!echo $PWD

#### the google bucket is not working
#These commands help identify the google cloud storage bucket where the example␣
↪files are held.
# project_id = "nosi-unmc-seq"
# original_bucket = "gs://unmc_atac_data_examples/Tutorial1"
#!gsutil -m cp $original_bucket/images/* Tutorial1/LessonImages
#This command copies our example files to the Tutorial1/Inputfiles folder that␣
↪we created above.
# ! gsutil -m cp $original_bucket/InputFiles/*fastq.gz Tutorial1/InputFiles
#! gsutil -m cp $original_bucket/InputFiles/hg38* Tutorial1/RefGenome
```

1.2.1 OK

Let's make sure that the files copied correctly. You should see 4 files after running the following command:

[3]:
```python
!ls Tutorial1/InputFiles
```

```
CTL_R1.fastq.gz  CTL_R2.fastq.gz  Mutant_R1.fastq.gz  Mutant_R2.fastq.gz
```

STEP2: QC

Sequences are typically provided as files in fastq format. This format includes 4 lines per sequence.

[6]:
```
## This file dne
# display_flashcards('Tutorial1/LessonImages/FastqFlashCard.json')
```

1.2.2 Click on the above image to see what each line represents.

Next, let's take a look at the sequence quality of the raw reads usinq fastqc:

[8]:
```
### check num threads
numthreadsint
```

[8]: 7

[9]:
```
#This command runs fastqc on each fastq.gz file inside our InputFiles directory␣
↪and stores the ouput reports in our QC directory.
!fastqc -t $numthreadsint -q -o Tutorial1/QC Tutorial1/InputFiles/*fastq.gz

#We then use multiqc to summarize the report.
!multiqc -o Tutorial1/QC -f Tutorial1/QC 2> Tutorial1/QC/multiqc_log.txt

#We'll load this into a pandas table to work in this context, but fastqc also␣
↪produces an html report that you can browse.
dframe = pd.read_csv("Tutorial1/QC/multiqc_data/multiqc_fastqc.txt", sep='\t')
display(dframe)
```

```
application/gzip
application/gzip
application/gzip
application/gzip

      Sample            Filename                File type  \
0     CTL_R1     CTL_R1.fastq.gz  Conventional base calls
1     CTL_R2     CTL_R2.fastq.gz  Conventional base calls
2  Mutant_R1  Mutant_R1.fastq.gz  Conventional base calls
3  Mutant_R2  Mutant_R2.fastq.gz  Conventional base calls

                Encoding  Total Sequences Total Bases  \
0  Sanger / Illumina 1.9         250000.0    12.5 Mbp
1  Sanger / Illumina 1.9         250000.0    12.5 Mbp
2  Sanger / Illumina 1.9         250000.0    12.5 Mbp
3  Sanger / Illumina 1.9         250000.0    12.5 Mbp

   Sequences flagged as poor quality  Sequence length   %GC  \
0                                0.0             50.0  43.0
1                                0.0             50.0  42.0
2                                0.0             50.0  42.0
3                                0.0             50.0  42.0

   total_deduplicated_percentage  …  basic_statistics  \
0                      52.289537  …              pass
```

```
1                        51.099795  ...              pass
2                        63.378570  ...              pass
3                        61.583462  ...              pass

   per_base_sequence_quality per_sequence_quality_scores  \
0                       pass                        pass
1                       pass                        pass
2                       pass                        pass
3                       pass                        pass

  per_base_sequence_content per_sequence_gc_content per_base_n_content  \
0                      fail                    pass               pass
1                      fail                    warn               pass
2                      fail                    pass               pass
3                      fail                    warn               pass

  sequence_length_distribution sequence_duplication_levels  \
0                         pass                        warn
1                         pass                        warn
2                         pass                        warn
3                         pass                        warn

  overrepresented_sequences adapter_content
0                      warn            pass
1                      fail            pass
2                      warn            pass
3                      fail            pass

[4 rows x 22 columns]
```

Alternatively, we can view the fastqc html files:

[11]:
```
#We can display the resulting fastqc results.
IFrame(src='Tutorial1/QC/CTL_R1_fastqc.html', width=1080, height=800)
```

[11]: <IPython.lib.display.IFrame at 0x7fa8189fe3e0>

Look at the the "Per base seqeuence content" in the above FastQC report. We'll trim the reads to remove some of this effect. For now, think about possible explanations for this result.

Also look at the "Sequence Duplication Levels". Sometimes duplicates appear due to the PCR amplification step of library preparation. We'll remove duplicates in a later step.

Lastly, look at the report at the "Overrepresented sequences". What are some possible explanations for this result?

Trimming

Next let's trim our sequences.

Why is it particularly important to trim the reads in ATAC-seq? To understand let's review how

ATAC-seq works. Tn5 inserts adapter sequences into accessible regions.

Image source: Grandi et al., Nature Protocols 2022

What would happen if the distance between inserted sites is short? For example our sequencing lenghth in the example dataset is 50 bp, so what would the sequence look like if our fragment (insert size) is only 30 bp long?

Interactive Quiz Question 1: Click on the correct answer in following cell.

[10]:
```
# display_quiz("Tutorial1/LessonImages/adapterQuiz.json")
```

1.3 Let's use trimmomatic to prepare the sequeces before mapping.

[12]:
```
#This will trim off N's as well as nextera adapters present in ATAC-seq library␣
↪preparation. placing the trimmed reads in our Trimmed folder.
!trimmomatic PE -threads $numthreadsint Tutorial1/InputFiles/CTL_R1.fastq.gz␣
↪Tutorial1/InputFiles/CTL_R2.fastq.gz Tutorial1/Trimmed/CTLtrimmed_R1.fastq.
↪gz Tutorial1/Trimmed/CTLunpaired_R1.fastq.gz Tutorial1/Trimmed/CTLtrimmed_R2.
↪fastq.gz Tutorial1/Trimmed/CTLunpaired_R2.fastq.gz ILLUMINACLIP:Tutorial1/
↪RefGenome/NexteraPE.fa:2:30:10 LEADING:3 TRAILING:3
```

```
TrimmomaticPE: Started with arguments:
 -threads 7 Tutorial1/InputFiles/CTL_R1.fastq.gz
Tutorial1/InputFiles/CTL_R2.fastq.gz Tutorial1/Trimmed/CTLtrimmed_R1.fastq.gz
Tutorial1/Trimmed/CTLunpaired_R1.fastq.gz
Tutorial1/Trimmed/CTLtrimmed_R2.fastq.gz
Tutorial1/Trimmed/CTLunpaired_R2.fastq.gz
ILLUMINACLIP:Tutorial1/RefGenome/NexteraPE.fa:2:30:10 LEADING:3 TRAILING:3
Using PrefixPair: 'AGATGTGTATAAGAGACAG' and 'AGATGTGTATAAGAGACAG'
Using Long Clipping Sequence: 'GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAG'
Using Long Clipping Sequence: 'TCGTCGGCAGCGTCAGATGTGTATAAGAGACAG'
Using Long Clipping Sequence: 'CTGTCTCTTATACACATCTCCGAGCCCACGAGAC'
Using Long Clipping Sequence: 'CTGTCTCTTATACACATCTGACGCTGCCGACGA'
ILLUMINACLIP: Using 1 prefix pairs, 4 forward/reverse sequences, 0 forward only
sequences, 0 reverse only sequences
Quality encoding detected as phred33
Input Read Pairs: 250000 Both Surviving: 249999 (100.00%) Forward Only
Surviving: 0 (0.00%) Reverse Only Surviving: 1 (0.00%) Dropped: 0 (0.00%)
TrimmomaticPE: Completed successfully
```

1.4 Let's do this for the other sample as well.

[13]:
```
#This will trim off N's as well as nextera adapters present in ATAC-seq library␣
↪preparation. placing the trimmed reads in our Trimmed folder.
```

```
!trimmomatic PE -threads $numthreadsint Tutorial1/InputFiles/Mutant_R1.fastq.gz␣
↪Tutorial1/InputFiles/Mutant_R2.fastq.gz Tutorial1/Trimmed/Mutanttrimmed_R1.
↪fastq.gz Tutorial1/Trimmed/Mutantunpaired_R1.fastq.gz Tutorial1/Trimmed/
↪Mutanttrimmed_R2.fastq.gz Tutorial1/Trimmed/Mutantunpaired_R2.fastq.gz␣
↪ILLUMINACLIP:Tutorial1/RefGenome/NexteraPE.fa:2:30:10 LEADING:3 TRAILING:3
```

```
TrimmomaticPE: Started with arguments:
 -threads 7 Tutorial1/InputFiles/Mutant_R1.fastq.gz
Tutorial1/InputFiles/Mutant_R2.fastq.gz
Tutorial1/Trimmed/Mutanttrimmed_R1.fastq.gz
Tutorial1/Trimmed/Mutantunpaired_R1.fastq.gz
Tutorial1/Trimmed/Mutanttrimmed_R2.fastq.gz
Tutorial1/Trimmed/Mutantunpaired_R2.fastq.gz
ILLUMINACLIP:Tutorial1/RefGenome/NexteraPE.fa:2:30:10 LEADING:3 TRAILING:3
Using PrefixPair: 'AGATGTGTATAAGAGACAG' and 'AGATGTGTATAAGAGACAG'
Using Long Clipping Sequence: 'GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAG'
Using Long Clipping Sequence: 'TCGTCGGCAGCGTCAGATGTGTATAAGAGACAG'
Using Long Clipping Sequence: 'CTGTCTCTTATACACATCTCCGAGCCCACGAGAC'
Using Long Clipping Sequence: 'CTGTCTCTTATACACATCTGACGCTGCCGACGA'
ILLUMINACLIP: Using 1 prefix pairs, 4 forward/reverse sequences, 0 forward only
sequences, 0 reverse only sequences
Quality encoding detected as phred33
Input Read Pairs: 250000 Both Surviving: 249998 (100.00%) Forward Only
Surviving: 0 (0.00%) Reverse Only Surviving: 0 (0.00%) Dropped: 2 (0.00%)
TrimmomaticPE: Completed successfully
```

1.5 Now let's summarize the trimming results.

[14]:
```
!fastqc -t $numthreadsint -q -o Tutorial1/Trimmed Tutorial1/Trimmed/*fastq.gz
!multiqc -o Tutorial1/QC -f Tutorial1/Trimmed 2> Tutorial1/QC/multiqc_log.txt

dframe = pd.read_csv("Tutorial1/QC/multiqc_data/multiqc_general_stats.txt",␣
↪sep='\t')
display(dframe)
```

```
application/gzip
application/gzip
application/gzip
application/gzip
application/gzip
application/gzip
application/gzip
application/gzip
```

```
              Sample  FastQC_mqc-generalstats-fastqc-percent_duplicates  \
0       CTLtrimmed_R1                                          47.710628
1       CTLtrimmed_R2                                          48.900371
2      CTLunpaired_R1                                           0.000000
```

```
3    CTLunpaired_R2                                        0.000000
4   Mutanttrimmed_R1                                      36.622134
5   Mutanttrimmed_R2                                      38.416652
6  Mutantunpaired_R1                                       0.000000
7  Mutantunpaired_R2                                       0.000000

   FastQC_mqc-generalstats-fastqc-percent_gc  \
0                                       43.0
1                                       42.0
2                                        0.0
3                                       48.0
4                                       42.0
5                                       42.0
6                                        0.0
7                                        0.0

   FastQC_mqc-generalstats-fastqc-avg_sequence_length  \
0                                          49.998444
1                                          49.999072
2                                           0.000000
3                                          50.000000
4                                          49.996372
5                                          49.996400
6                                           0.000000
7                                           0.000000

   FastQC_mqc-generalstats-fastqc-median_sequence_length  \
0                                                 50
1                                                 50
2                                                  0
3                                                 50
4                                                 50
5                                                 50
6                                                  0
7                                                  0

   FastQC_mqc-generalstats-fastqc-percent_fails  \
0                                          10.0
1                                          20.0
2                                           0.0
3                                          30.0
4                                          10.0
5                                          20.0
6                                           0.0
7                                           0.0

   FastQC_mqc-generalstats-fastqc-total_sequences
0                                        249999.0
```

1	249999.0
2	0.0
3	1.0
4	249998.0
5	249998.0
6	0.0
7	0.0

Step3: Mapping

Our fastq files include sequences and quality scores for each base, but we want to figure out which genomic location these sequences came from. To do this we will map each sequence to a reference genome using bowtie2.

Mapping reads requires a reference genome. Due to time and memory considerations, in this tutorial we prepared that file for you and will only map to chr4. However, in a full analysis, we would map to the entire genome. To do so you would need a fasta file corresponding to the reference genome (e.g. hg38.fa) from which you'd create an index of the genome using bowtie2-build. This can be done with the command:

bowtie2-build reference_genome_file.fa outputprefix.

As mentioned, we've gone ahead and created the index for you, and, earlier, you copied them into the RefGenome directory. These index files end in the bt2 extension.

[ ]:

```

```

[ ]:

```

```

[ ]:

```

```

[15]:

```
!ls Tutorial1/RefGenome/*bt2
```

```
Tutorial1/RefGenome/hg38chr4.1.bt2  Tutorial1/RefGenome/hg38chr4.4.bt2
Tutorial1/RefGenome/hg38chr4.2.bt2  Tutorial1/RefGenome/hg38chr4.rev.1.bt2
Tutorial1/RefGenome/hg38chr4.3.bt2  Tutorial1/RefGenome/hg38chr4.rev.2.bt2
```

These index files were created from our fasta file:

[16]:

```
!ls Tutorial1/RefGenome/*fa
```

```
Tutorial1/RefGenome/NexteraPE.fa  Tutorial1/RefGenome/chr4.fa
```

Notice that the single fasta file created mutiple index files. When we align we'll specify the prefix of the index files.

[17]:

```
#Notes: The -x option specifies the prefix of the index. -1 specifies our␣
↪left-end trimmed reads file. -2 specifies our right-end trimmed reads file.␣
↪-S specifies our output file in sam format.
!bowtie2 -p $numthreadsint -x Tutorial1/RefGenome/hg38chr4 -1 Tutorial1/Trimmed/
↪CTLtrimmed_R1.fastq.gz -2 Tutorial1/Trimmed/CTLtrimmed_R2.fastq.gz -S␣
↪Tutorial1/Mapped/CTL.sam
```

```
249999 reads; of these:
  249999 (100.00%) were paired; of these:
    100801 (40.32%) aligned concordantly 0 times
    111601 (44.64%) aligned concordantly exactly 1 time
    37597 (15.04%) aligned concordantly >1 times
    ----
    100801 pairs aligned concordantly 0 times; of these:
      20903 (20.74%) aligned discordantly 1 time
    ----
    79898 pairs aligned 0 times concordantly or discordantly; of these:
      159796 mates make up the pairs; of these:
        143536 (89.82%) aligned 0 times
        6881 (4.31%) aligned exactly 1 time
        9379 (5.87%) aligned >1 times
71.29% overall alignment rate
```

[18]:
```
##Let's do the same thing for our other sample.
!bowtie2 -p $numthreadsint -x Tutorial1/RefGenome/hg38chr4 -1 Tutorial1/Trimmed/
↪Mutanttrimmed_R1.fastq.gz -2 Tutorial1/Trimmed/Mutanttrimmed_R2.fastq.gz -S␣
↪Tutorial1/Mapped/Mutant.sam
```

```
249998 reads; of these:
  249998 (100.00%) were paired; of these:
    75447 (30.18%) aligned concordantly 0 times
    127648 (51.06%) aligned concordantly exactly 1 time
    46903 (18.76%) aligned concordantly >1 times
    ----
    75447 pairs aligned concordantly 0 times; of these:
      21698 (28.76%) aligned discordantly 1 time
    ----
    53749 pairs aligned 0 times concordantly or discordantly; of these:
      107498 mates make up the pairs; of these:
        90592 (84.27%) aligned 0 times
        5712 (5.31%) aligned exactly 1 time
        11194 (10.41%) aligned >1 times
81.88% overall alignment rate
```

1.5.1 Answer the following question only if you are using the example dataset we provided. This question is simply a check to ensure everything was processed correctly.

[20]:
```
# display_flashcards('Tutorial1/LessonImages/alignment.json')
```

1.6 Bowtie2 output a file in sam format which contains the original sequence, quality scores, and the genomic coordinates matching each read.

In the next commands we'll convert the file to the more compressed bam format and sort the reads by chromosomal coordinates.

[21]:
```
#This will convert to bam by using samtools view with the -b option. The h and S option tells samtools that the file has a header and is in sam format. We will pipe this to samtools sort. Pay attention to the "-" at the end of the sort command which tells samtools to use stdin.
!samtools view -q 10 -bhS Tutorial1/Mapped/CTL.sam | samtools sort -o Tutorial1/Mapped/CTL.bam -
print("done")
```

```
done
```

[22]:
```
#Let's do the same thing for our Mutant sample.
!samtools view -q 10 -bhS Tutorial1/Mapped/Mutant.sam | samtools sort -o Tutorial1/Mapped/Mutant.bam -
print("done")
```

```
done
```

You may have noticed the parameters -bhS and -q 10 in the above commands. Briefly, -bhS describes aspects of the file to samtools, such that you want to output a bam file (the b option), that it has a header (the h option), and that it is currently in sam format (the S option). We also specified -q 10 which removes reads with a mapping score <= 10.

Interactive Quiz Question 2: Click on the correct answer in the following cell.

[24]:
```
# display_quiz("Tutorial1/LessonImages/mappingquality.json")
```

Step4: Removal of Duplicates

It's important to remove duplicates from our reads because part of the ATAC-seq method includes a PCR step for library amplification. This can create biases in the data resulting from PCR duplicates. To understand how PCR duplicates can affect the analysis, let's jump ahead a bit. Accessibile sites are represented by ATAC-seq "peaks" of signal.

Interactive Quiz Question 3: Click on the correct answer in the following cell.

[25]:
```
display_quiz("Tutorial1/LessonImages/duplicateQuiz.json")
```

```
<IPython.core.display.HTML object>

<IPython.core.display.Javascript object>
```

Okay, let's remove these duplicates using Picard.

[26]:
```
#this will take the sorted bam file and remove duplicates, saving a new bam file and a summary in a text file.
!picard MarkDuplicates --REMOVE_DUPLICATES TRUE -I Tutorial1/Mapped/CTL.bam -O Tutorial1/Mapped/CTL_dedup.bam --METRICS_FILE Tutorial1/Mapped/CTL_dedup_metrics.txt --QUIET 2> Tutorial1/Mapped/PicardLog.txt
print("done")
```

```
done
```

[27]:
```
#We also should do this for the other sample.
!picard MarkDuplicates --REMOVE_DUPLICATES TRUE -I Tutorial1/Mapped/Mutant.bam␣
↪-O Tutorial1/Mapped/Mutant_dedup.bam --METRICS_FILE Tutorial1/Mapped/
↪Mutant_dedup_metrics.txt --QUIET 2> Tutorial1/Mapped/PicardLog2.txt
print("done")
```

```
done
```

[28]:
```
#We can use multiqc to summarize the metrics
!multiqc -o Tutorial1/QC -f Tutorial1/Mapped 2> Tutorial1/Mapped/multiqc_log.txt
dframe = pd.read_csv("Tutorial1/QC/multiqc_data/multiqc_general_stats.txt",␣
↪sep='\t')
display(dframe)
```

```
   Sample  Picard_mqc-generalstats-picard-PERCENT_DUPLICATION
0     CTL                                            0.297989
1  Mutant                                            0.275425
```

Great job!

We have completed the preprocessing steps and are ready to move on to some downstream analysis. Take a break here or move on to the next tutorial:

Visualization and Peak Detection.

[ ]:
```
```