Beginning Bioinformatics

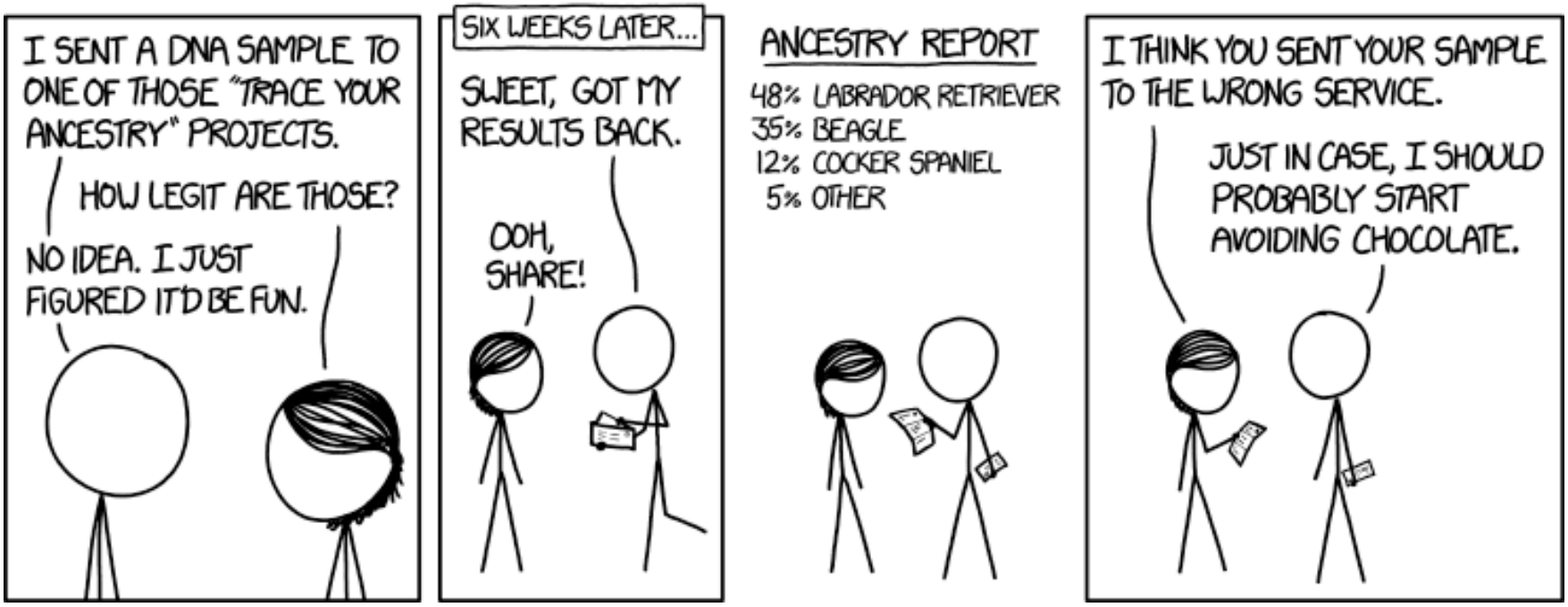
Ammon Corl (he/him)

U.C. Berkeley, 2/2/26

Course Materials: Search on Github for “MVZ bioinformatics”

Materials are in ammoncorl/Ammon_Corl_*bioinformatics*_workshop_2-2-26

Download from the green “code” button



From: xkcd.com

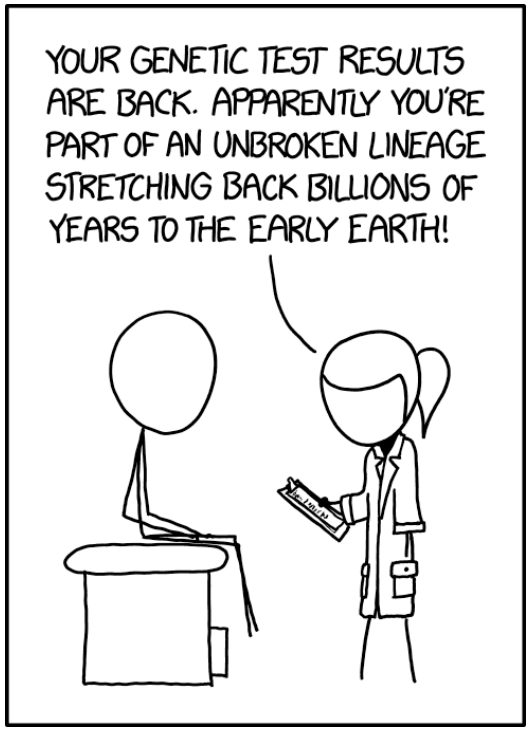
Beginning Bioinformatics: Understanding and cleaning reads

Goal 1: Learn how to assess and filter next-generation sequencing data

Plan for today

a. Basic info about sequencing reads and manipulating them.
b. Read quality reports (FastQC, MultiQC)
c. Filtering reads (HTStream)

Reason to do all this: Understand biology better using genetic data.



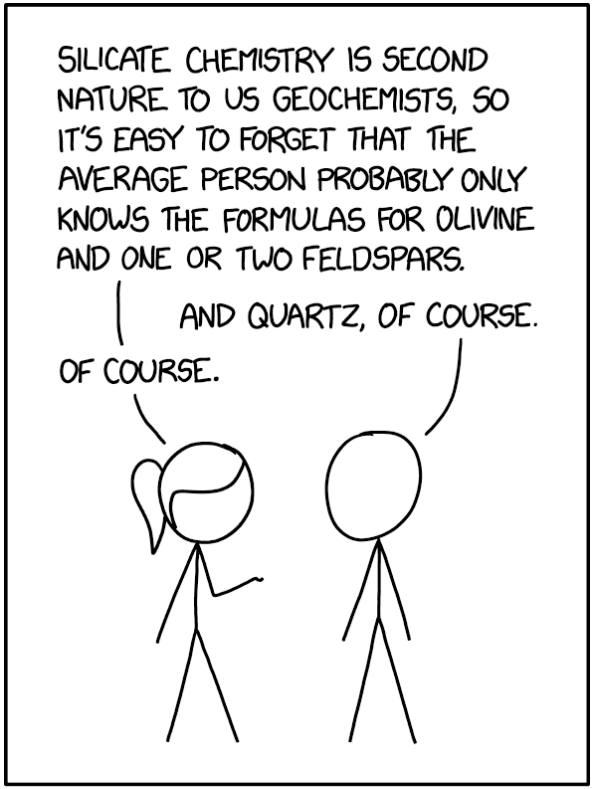
From: xkcd.com

Beating the Bioinformatics Boogeyman

Goal 2: Make this workshop useful for entry into bioinformatics

a. Please ask questions!

b. Please let me know if something is confusing!



EVEN WHEN THEY'RE TRYING TO COMPENSATE FOR IT, EXPERTS IN ANYTHING WILDLY OVERESTIMATE THE AVERAGE PERSON'S FAMILIARITY WITH THEIR FIELD.

From: xkcd.com

Keep a Growth Mindset

1. Mistakes are part of the learning process.

   a. If you were going to learn how to juggle, you would expect to drop things all the time.

2. Things you are learning now will get easier over time, which will allow you to learn more complicated things later.

Michael Moschen

Why Learn Bioinformatics?

1. Learn how to handle BIG DATA.
2. Computer skills to automate repetitive tasks.
3. Good experience with troubleshooting problems.
4. There are jobs for people with bioinformatic skills.

>Leptocoma_aspasia_T1R1_exons1to6_introns_flanking
TTTAAAAGCTATGCACCTTCCAGTTCTGACGGCTCCTCTGAGACATCTGTCAAACATTAGCTTTTTATTTTTTTCCTTCTCTTAAATCTCATGCAGCACTTAAAAGCTTGCAGAACATGGGAAGACTCACCGGTTCAGTAACAGG
TTAATTAGATATCTGTTGAATGTAGATTTCCCAGTGTTTTTTGGGCCACACACCAGCACCACAGGGACACCTTCATCTTCAGCTAGAAATAGAAGCTGAGTGTGAACTGGACTGATGCTGGCATCACTGATCCACCCCAGGCA
ATGCCTAAGCTCTAGATGCACATTCTTTATTATAGAGAAAAATTTGACCCTCAGCATATGCTGGTCAGGGCACTCCCGCAACTTTAACTCATGAAAAGCTTCCTGAATTTTCTTTTTTCCAGGCTACCAAAGCCCCAACATGGCC
AAAGGCCTGCACTCACCACAGCAGGCCTGGATCAGCTCCTCCAGCGCCAGGCACATGCTCTCCGACACCAGCAGCCCGCGGTCGGAGCTGCACTTGACGATGCCCACGGACGCCAGCACCGCGTCCTCTGGCGTGAA
GTTAGACTCCTCCTTTTTCTAGGCGAGAGGGCGAGCACTCAGCACGGGGTGTGTGAGGCCACGGCCTGGCGCTGACAGGGGCGGCCGCCGCTGGCTCTGCATGGGCGGTGCGGGGACGGCACAAGGCGCAGCAGGC
ACCGCGCCGCCTCACCTGGGGCTCGAAAAACCGCGACAGCGGTGGGTGGCTCAGCAGGAATCGCGTCACCGCCGTGTCCAGGTGCTCCAGCAGCACCACGGCGCAGTCGGGAGAGAACCGTGCCATCAACTTCACCC
GCGCCTCTGAAAGAGGCAAGGGCCGCCATCACCCTCAGCCCCGCACCCTCCCGCGAAATGGCACCCACCCGCTGCGCAGCGCCCCGCGCACAAAATGGCCGCTCCCCGCACCGCCCGTGAAATGGCGCCCGCCCGC
GGCCCCGTCCCGTCCCGTCCCGCACTCACGGCGGCGGACGCGGTGCGAGCGCATGGCGGCGGCGGCGCGCAGCTGGCGGAGGGCGGCGGCNGCGGTGCGGGGCCACGCCACCATGCTGCCGCCCGCG
CTGCTCCGCCTGTGCCTCTGCGCGGCTGCTGCCGCCGCCTCTTCGTTCACCCTCCGCGGCGATTACCGCCTGGCTGGGCTGTTCCCGCTGCACACCGCGGCGGACCAGAACGACGGCAGCCTGCTCATGAACGGCTG
CGACGAGTGAGTGCCCGCGCTGCCCGTGCCCCCAACTAGCTCTGTCCTCCCTAACAACCACTCTCGATTCCATAGCGCTGCCTTCAAGAGCCACGGCTACTGCCTATCCCAGGCCCTGCGCTTCGCTGTGGAGGAGATCA
ACAACTGCACCACGCTGCTCCCCAATGTCACCCTGGGCTATGAAATCTACGACACCTGCTCTGAGCCTACCAACTTCCATGCCACGCTGTGTGCCCTTGCTCGTAAAGACAGGCAGGATGTCCAGGTGCTCCCCGGTTTCC
AACACTATGAACCCCAGGCTGTGGCTGTCATTGGCCCTGACAGCACCCGGCTGGCCCTCACCACAGCTGCTGTTCTCAGCCTCTTTCTTGTACCAGAGGTAAGTGCAGCTCCTGCAGCTGATGAAAAACAGGGATCCCAA
GGAGTGATGGTTTCAGGGAGGGTGGGCTCAGACCTGAGGAGCCTGGTAAAACTGGAAATGCAGTCCCTCTGTGTGTGCTGTCACAGGATCTTTCCCCGGCAGTCAGCGGGCTGGCTTGCCATCCATCACAGCTTTCTCCT
GCTTCAGCACGCTTGGCACTGGGTTCGTGGCAGCCTGCCCTGCTCTAAGTCTTCCCAGGACAGGCTGCCCATGCCATTAGCTCTCAGGGGTGCAGCTGCCTGGGTCAGCACATCCCCCTTCCCTGTGGCTCCCTTTGCA
GATCAGCTACGAAGCCTCCACGGAGTCGCTGAGCCTGAAGCGGCTGTACCCCTCTTTCCTGCGCACCATCCCCAGTGACAGGCAGCAGGTGAAGGCCATCTTCTTGCTGCTGCAGCACTTTGGGTGGACCTGGGTGGTG
CTGCTGGGCAGCGACAACACCTATGGCAGGGCTGGCCTGGATGCCCTGCAGGAGCTGCTGACCACAAGCAATATGTGTGTAGCCTATCGAGGCACTATCCCCGCCAACTCGGACGCCAACAACCCGGAGCTTCACAACC
TGGTCCGAATCATCACAGACCTCAAGGTCAATGTCACTATTGTTCTCCAACAAAGCGAGTGTTCTGCCGTTCTTTGAGGTGGTGGTCCGGAGGAACATCACGGACATGGTATGGGTGGCCTCCGAAGACTGGTCGCTGG
CTCAAACTATCTGGCAGGTGCCTGGCATCCAGACCATTGGCTCAGTGTTTGGGATGGCAGTTGAGAAGCCAGAGTCTCCGATGCTGGAACGCTTTGAAGCTTGGAGGATGTCAGAGGAAGGTGCTGTGGCTGAGTGTGCC
AGCAGTGCAGAGGCAGGCAGGGAATCTGTGGGGAACGAGCGGCTGGACTGCACCCAGTGCTGCACTGGCTGCCATGCGCTCGCCACTGTCCCTGACATGTATGACGCTCAAGGCTCCTTCAACGTGTACTCAGCCGTGT
ATGCTGTGGCCCATGGCCTTCATGATCTGCTGGGCTGTGCCTCGGGGGCCTGCAGCAAAGGCATTGTCTATCCCTGGCAGGTAAGAGCAACCTTGTGGAAGATCCATGAGGATCCCATCCCACAGGGAAGGATTTTTCCAT
TGTGCCCAAAGGTTTCCTGTCTTTCCCAAAGGCTCAAATCCCAGCTGAAAACCCTTGTGGACTTCAGCTACCCCTCACAGTCAGAGCAGCCAAGACCTAGACCTAGACTCCATCCTAGGTCACTTGTTCCTCTACCACCAGC
CCATCAGTGGATGGGATTTCCTGCAGATGTTGGGGGCTTGAAAGCAGCTGCAGGACAGAGGAAGCGTCCTCCAACACATCTCCCTCCCAATGGCAGGAGTCACGCCATTGGAAGGGCTCCTCTCTTACTCTTCTGTTTTAC
AGCTCCTACAAAAGATTAGGCAAGTAAACTTCACCCTGCACAAGAGCCGAATCTCTTTCGATGCCAACGGGGACATTCATAAAGGCTATGACATTGTCATGTGGAAGTGGCTGGGCACAAATTGGTCTTCTGATGTGATAGGA
ACCTTCCGTGTGAACCCTGACAGGCTGAGCATCGACCCAGGCAAAGTCCTGTGGCACACACAAGATGGCAAGGTATGTTCTGCACCCAGGTCTTATTTTTTGCTCACTTCTGTGCTGACTGCATCTAAATGCCACCAGTTCT
GGGGTGTTTCCTCATAGCAGGGCAAAACTGAGGGCACAGGCAGTGTGGTGGGGTGCCAGAGTGCTGCTGAGCCTGAACTGTGCTGCTGCTTGCCAGCCTGACCCTGTCTGTGCTCCCGGCAGGCTCCCAGCTCGGTGT
GTTCCGAGGCCTGTGAGCAAGGGGAGATGCGGCTGCAGCAGAGCCGCCACAAATGCTGCTTCAGCTGTGTGGCCTGTAAACCGGGAACCTTCCTGAACACATCAGGTCAGTACACCACAGGGTCTGCTCCTTCCCCTTC
CTCCCCTGCCCACATCCTACCTCCTGCTCCTATCTCACCATTCCCAGCCTCTCACCTCCTGTGGTCAGTAAAATCTCTGAGCTGACCTGAAGGCAGTGAGTCTGGCTCCTGTTTGCTTACCTTACCAGTGGATATGCAAAGCC
CATGTCCTGTGCTTTTCTGGGGAGTACTGCCCACAACAGGTGAGCCCACCTGGGCAAACAGAGCCCCTTCACCTGTGACCAGGGAAAGGGGATGTCCGGCCCGGTTGCTGTTCCCTGCACCGCATCCTCTGCCAGGAAA
ATTCAACCTAGTGGTTGCCTTTGTGCTCCAGATGGCACACGGGGCTTCCGCACTACGTGGTCATCAGCACCTGGCAGATGGAAATGTCTCCTGCCTGGGACAGGCCCCAGTAGTCTCCTGGCTCAGAATCATCAGCTACAG
CAACAAACCCCATCCTGGCACCTACTCCCTCAACACCGTTGGCCTGTGGAGTGGGCAGGGGCAGCTCTGGGGACAGTGAGAGCAAAAAGCCCCTCTACCTGGTACTGTGCCATTCAACAGCAGCTGCTGCAGATCCTTC
CCAGGTTACTGCTTGAAGCACAGGCCCCTTCCCACCCATAGCCCCCACGGCCCTGAGCCTCACACAGCTGAGCCCCTGTCTCAGCCCAGGCCATGCATTTTGGGAGAGTCCACGGGGTGAATTTTAGGTGGGGGGGTTA
TTTTGGGAAAAGAGCCCCCTCTGCTACCCCAGGCCTGGGAGTGCCTGCAGGAAGGAGCAGGCGTGATGGCTGAGCAGAGCACAAGGAGCTGCGGGCCGGGCCCTGCACCGTGGCAGAGCTCTTACCGGATCTGAGGT
GTGTGAGGTGTCCGTGTCCTCCACAGACCTCTTTGACTGCCAGGCCTGTGGCTTGGATGAGTGGGCCCCAGCAGGGAGTGAGGTCTGCTTCAACCGCACCATCGAGTTCCTGTCCTGGTCTGAGCCCCTCTCCTGGGTG
CTGCTGGCCCTGGCTGCTCTTCTCATGCTGCTCATAGCGGCGCTGGCTGTCCTGTTCGCCCTCAACGCCTCCACACCTGTGGTCAAATCTGCGGGCGGGAAGACGTGTTTCCTCATGCTGGGCTCCCTGGCCTGCACCT
GCAGCAGCCTCTTCTGCTACTTCGGGGAGCCCACGCGGATGGCGTGCCTCCTGCGGTTGCCGCTGTTCGACATCAGCTTCACCGTGTTCCTCTCGTGCGTGGCGACCCGCTCCTTCCAGATCCTCTGCATCTTCAAGCTG
AACGCGCGCTACCCGGCGCTCTACGAGGCCTGGATGCGACACCAGGGGCCGGTGCTGTTCGTGGCCGCCAGCACGGCGGCACAAGTGGTGCTGTGCGTGGTCATCGAGGCCGCCAGCCCCTCGGTGCCGCGCCGGG
AGTACGGCGCGCGGGACGAGTGGGTGGTGCTGGAGTGCGCCCAGAGCGCCGCGGCCGACGCCGCCACCGCCTACACGGTGCTGCTCAGCGCCGGCTGCTTCGCGCTCAGCTACGCGGGCACGGACCTGCCCGCCG
CCTACAACGAGGCCAAGAGCCTGACCGTGAGCCTGCTGCTACACCTGAGCTGCTCGGCCGCCGTGCTCTGCTGGCAAGGCGCGCTGCGCGGCCGGGCCGAGACGACGGCGCGGGTGCTCAGCACGCTGGGCACGCT
CGCAGCACTGCTGGGCGGGTACTTCGTGCCGCGGGCCTTCGTCATCCTGCTGCGGCCGCACCAAAACACGGCCGAGCACTTCCAGATGGCCATCCAGGAGTACACGCGCCGCCTGGCCGCCGCCTGAGCCGCGGGGC
TCGGCCCGCACCGCGCCTCCCCCGCC

General Recommendations

1. Keep detailed notes, just like lab or field notes.
    a. I use the text editor BBedit for note taking.
    b. Example of my bioinformatics notebook below

10/13/21

Coming up with a file of all the top candidate genes. (**General problem that I am working on).**

Note, I will have to redo this, since some have the DNA sequence on multiple lines. **(Issues that I will have to deal with later.)**

for file in *.txt; do cat $file >> Top_candidate_genes_in_sunbird_and_flowerpecker.fa; done

(This is command that I use all the time, but I still record it to have a record that I can later look back on.)

grep -v ">" Top_candidate_genes_in_sunbird_and_flowerpecker.fa.txt|wc

1,108,246 **(This records the amount of DNA sequence in my file.)**

Looks like in hummingbirds I checked for repetitive elements and mitochondrial dna.

A reminder of what I need to work on later on.

You've got data! Now what?

1. Download it. Filezilla (https://filezilla-project.org/) is a good option because it lets you know if something went wrong with the download.

2. Back it up! Store in at least two independent places. Also archive it in the Sequence Read Archive (https://www.ncbi.nlm.nih.gov/sra)

3. Check that the files downloaded correctly.

   This is done with the file "md5sum.txt" that the sequencing center provides.

4. Example of what is in that file.

bb67e2d8a2cc09b5d865fb624571d208 HB877_S1_R1_001.fastq.gz

b300a3ffd9279be8d358599eb7188dce HB877_S1_R2_001.fastq.gz

5f760330808a095b10b7a4792f40238f HB878_S2_R1_001.fastq.gz

22312336bedaed00b1a4a5aadc346911 HB878_S2_R2_001.fastq.gz

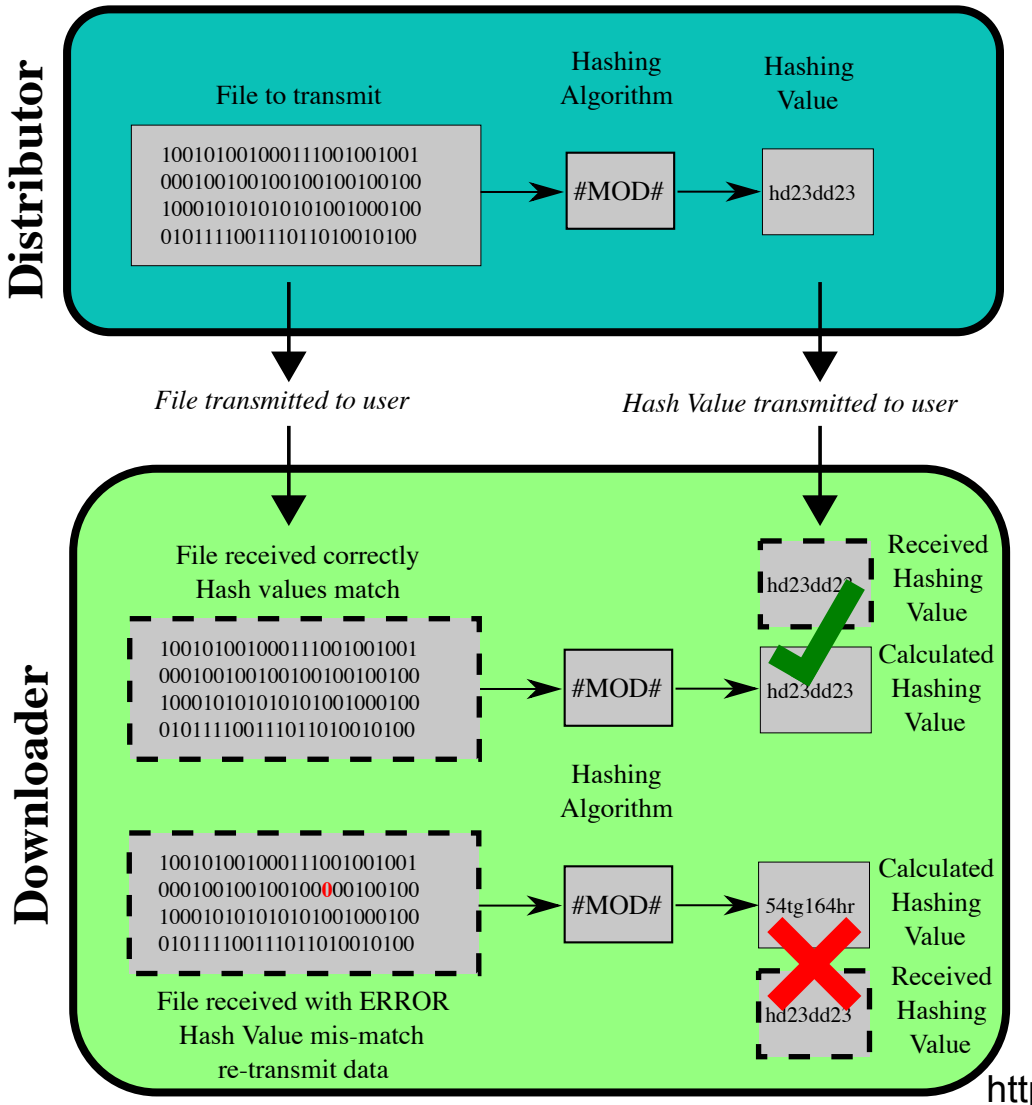
34c589522b415f0611f9480a69c12c0c HB879_S3_R1_001.fastq.gz

c9463ecaa882995c19c7fd24add2bea3 HB879_S3_R2_001.fastq.gz

33966ba263f0f8f727a0295084185bd2 HB880_S4_R1_001.fastq.gz

Pretend you are a spy with md5

1. MD5 is a “hash function” that used to be used for cryptography.
2. It is now used to check if files have been transferred successfully.



https://en.wikipedia.org/wiki/MD5

Checking files with md5

1. Mac version: Need to run command in Terminal. Navigate into the folder with the data. Then type: md5 filename

Example: md5 HB877_S1_R1_001.fastq.gz

The resulting code: bb67e2d8a2cc09b5d865fb624571d208

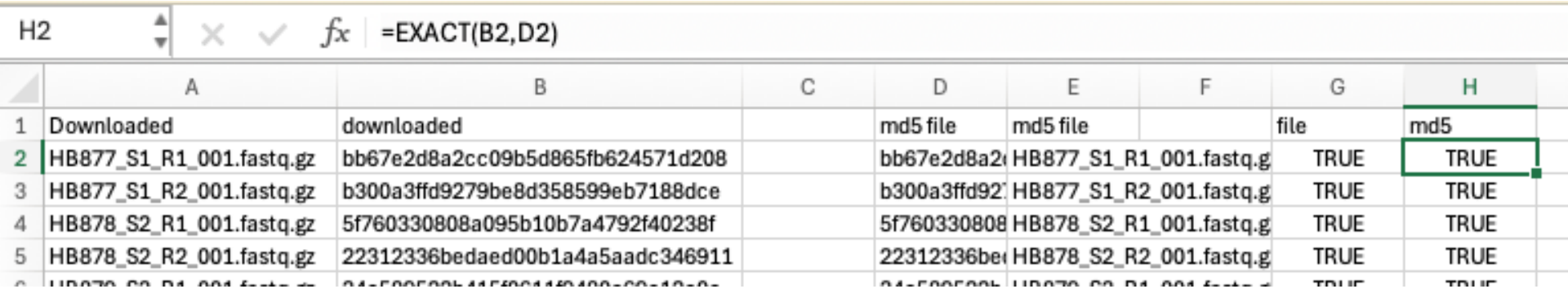
2. Linux version

md5sum HB877_S1_R1_001.fastq.gz

3. Looped Mac version (will process all the files ending in .fastq.gz in the folder)

for file in *.fastq.gz; do md5 $file >> HBplates12-23_md5check_1-20-26.txt; done

4. Compare the hashes between the original and downloaded files. Can use the "EXACT" function in Excel.



	A	B	C	D	E	F	G	H
1	Downloaded	downloaded		md5 file	md5 file		file	md5
2	HB877_S1_R1_001.fastq.gz	bb67e2d8a2cc09b5d865fb624571d208		bb67e2d8a2	HB877_S1_R1_001.fastq.g		TRUE	TRUE
3	HB877_S1_R2_001.fastq.gz	b300a3ffd9279be8d358599eb7188dce		b300a3ffd92	HB877_S1_R2_001.fastq.g		TRUE	TRUE
4	HB878_S2_R1_001.fastq.gz	5f760330808a095b10b7a4792f40238f		5f760330808	HB878_S2_R1_001.fastq.g		TRUE	TRUE
5	HB878_S2_R2_001.fastq.gz	22312336bedaed00b1a4a5aadc346911		22312336be	HB878_S2_R2_001.fastq.g		TRUE	TRUE

Cleaning the reads

1. What does “cleaning the reads” mean?

2. It is a quality control step for Illumina data (i.e. short DNA sequences from 50-300 bp long that may be paired with a nearby sequence of the same size).

3. During read cleaning we want to assess the quality of the reads (i.e. how reliable the data is) and remove poor quality data.

4. We also want to trim off the adapter sequences that were attached to the DNA that you wanted to sequence.

5. After these steps you can assess how much good data you have and have more reliable data for downstream applications.

6. The raw data typically come in compressed files in FastQ format, one for each of the pair of reads (R1 and R2).

   a. For example:

samplename_L001_R1_001.fastq.gz

samplename_L001_R2_001.fastq.gz

Reading the reads

What does a FASTQ file look like?

For each cluster that passes filter, a single sequence is written to the corresponding sample’s R1 FASTQ file, and, for a paired-end run, a single sequence is also written to the sample’s R2 FASTQ file. Each entry in a FASTQ files consists of 4 lines:

A sequence identifier with information about the sequencing run and the cluster. The exact contents of this line vary.

The sequence (the base calls; A, C, T, G and N).

A separator, which is simply a plus (+) sign.

The base call quality scores. These are Phred +33 encoded, using ASCII characters to represent the numerical quality scores.

Here is an example of a single entry in a R1 FASTQ file:

```
@ML-P2-14:9:000H003HG:1:11102:17290:1073 1:N:0:TCCTGAGC+GCGATCTA
TTTGGTAACAGCATGAATTATTCTAGCCACTAAAACTCTATGAACATCTTGTGAAGGTTTCAGATAGAGCCTGAAGTACACAGAGAACAATTCTTAAAAAA
+
AAAAAEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEAEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE<AEEEEEEE
```

https://support.illumina.com/bulletins/2016/04/fastq-files-explained.html

Manipulating the reads

1. Look at the reads:

   In terminal: head filename (will show the top of the file)

   In terminal: tail filename (will show the bottom of the file

   In terminal: less filename (will show a portion of the file, where you can scroll through with arrow keys or the space bar, press “q” to quite the viewer)

   In terminal: cat filename (will output the whole contents of the file to the terminal screen)

2. To count the reads:

   In terminal: grep “@” filename| wc -l

   (wc stands for “word count” and the -l flag specifies just count the lines)

3. To look at zipped files that are in gzip format (.gz at end): type “zcat” and then file name (Linux system). On a Mac terminal, you type “gunzip -c" and then the file name.

   a. For example (zcat filename | less -S) or (gunzip –c filename | less -S)

   b. Count the reads in a compressed file : gunzip –c filename | grep “@”| wc -l

Quality Control of the Reads: FastQC

1. FastQC is a quality control tool for high throughput sequence data.

2. Can download at: https://www.bioinformatics.babraham.ac.uk/projects/fastqc/

   a. To run it on the command line (even on a Mac), download the Win/Linux zip file. In the terminal, go into the FastQC folder where there will be a file called “fastqc”, which you may need to make executable. To do this, type “chmod +x fastqc”

   a. FastQC website has examples of output from good and bad data.

3. Requires a suitable Java Runtime Environment

4. Can run in a GUI or on the command line

5. Sometimes you need to make programs executable due to security settings on computer.

   a. Mac OS prevents lots of command line software from opening. Right clicking on it and then opening it will make an exception and allow it to be used.

   b. Making things executable in the terminal

   ./program_name = how to run an executable file in the current directory.

   chmod +x filename = to change a file into an executable file

   chmod +x * = to change all files into a folder into executables.

Adding a Program to the Path

1. Typically you want to run command line programs from any folder on your computer, not just the folder where the program is kept. To do this you need to add the programs to the path of the computer.

2. Open the terminal. Then type nano ~/.bash_profile or nano ~/.bash_rc (only the latter worked for me on MacOS Catalina but the former now works on my new laptop) and hit enter after the following:

3. Paste in the paths to the programs you need. The following is what I put in for running FastQC.

Note: This is the path to the folder containing the program. There is an executable program called “fastqc” in the folder FastQC.

```
PATH=$PATH:/home/ammon/programs/FastQC

export PATH
```

4. After making modifications, press “control X” to save changes, then hit enter when it shows the file name to write. Then run the newly modified path file with the command below.

   a. source ~/.bash_profile or source ~/.bash_rc

   b. Will have to do the source command again after restarting your computer (and when using a new terminal window.).

5. You can test if it worked by typing in the name of the program in the terminal, which should now show you the options for using it.

   a. Try: fastqc -help

Running FastQC

1. You can run FastQC through a graphical user interface (GUI).

   a. Open the program, then go to File, then open, then open a sequence file.

2. You can also run the program in the Terminal.

   a. fastqc filename -o output_folder_name

   b. Output folder needs to be made beforehand: mkdir output_folder_name

3. In the terminal, you can also automate getting data for a large number of files.

   a. Navigate into the folder where you have the sequence data.

   b. Type the following:

   for file in *.fastq.gz; do fastqc $file -o output_folder_name; done

   c. This will work, but you need to keep the terminal window open or it will stop processing data. This is a problem when running things on a server if you wish to turn off your own computer.

4. To make running the program easier, I put the commands into a bash/shell script that can be run in a different way.

Running FastQC Via a Shell Script

1. I made a shell/bash script for running FastQC on the command line called "run_fastqc_keep_zip.sh"

    a. This lets you automate it or run it on a server.

    b. To switch your terminal window to bash from zsh (the Mac default), type "bash" and then enter.

2. To use it, you type in the terminal: bash run_fastqc_keep_zip.sh -i input_folder_name -o output_folder_name.

    a. The script will make the output folder in the folder with the reads, so do not make an output folder before running it.

3. I normally run the command in a screen so that the script will keep running when I log out of the server or close my screen window.

4. To run a script in a screen, screen -S name_of_screen bash scriptname.sh

    a. screen -ls lists all the screens running

    b. In the terminal after initiating the screen command, if you do Control A then D, it detaches that screen from the window.

    b. screen -r name_of_screen reattaches the screen to being interactive

5. Example command to run FastQC in a screen:

screen -s ammon.fastqc bash run_fastqc_keep_zip.sh -i Vertlife_bird_genomes10-24-20 -o Vertlife_bird_genomes10-24-20_rawread_fastqc

Questions

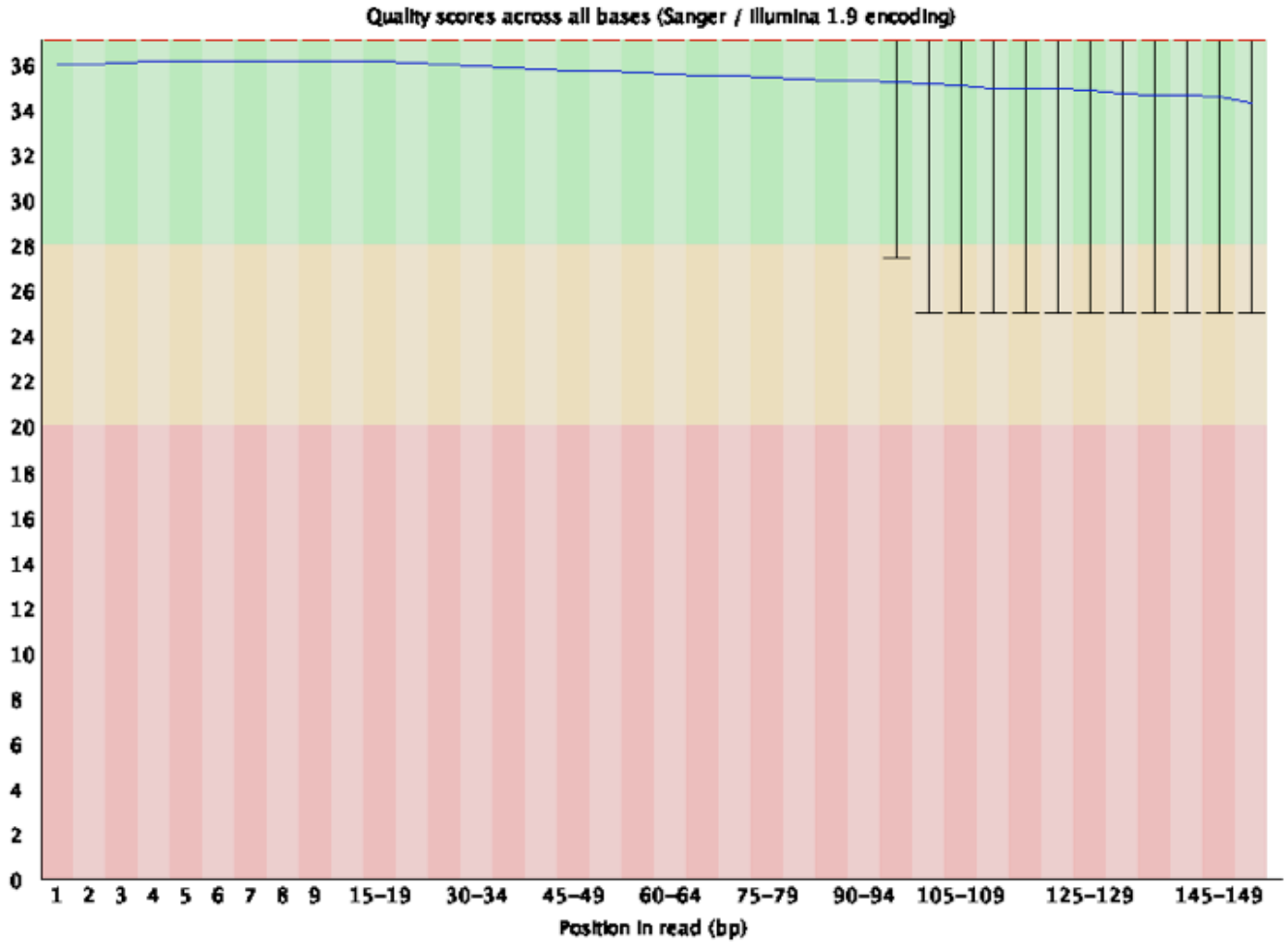
FastQC - High Quality Data NovaSeq (read 1)

1. All sequences and all positions have average sequence quality in the green area = high quality.

2. Q10 = 90% accurate, Q20 = 99% accurate, Q30 = 99.9% accurate (i.e. 1 in 1000 bases are called inaccurately. (https://www.illumina.com/science/technology/next-generation-sequencing/plan-experiments/quality-scores.html)

Per base

sequence quality

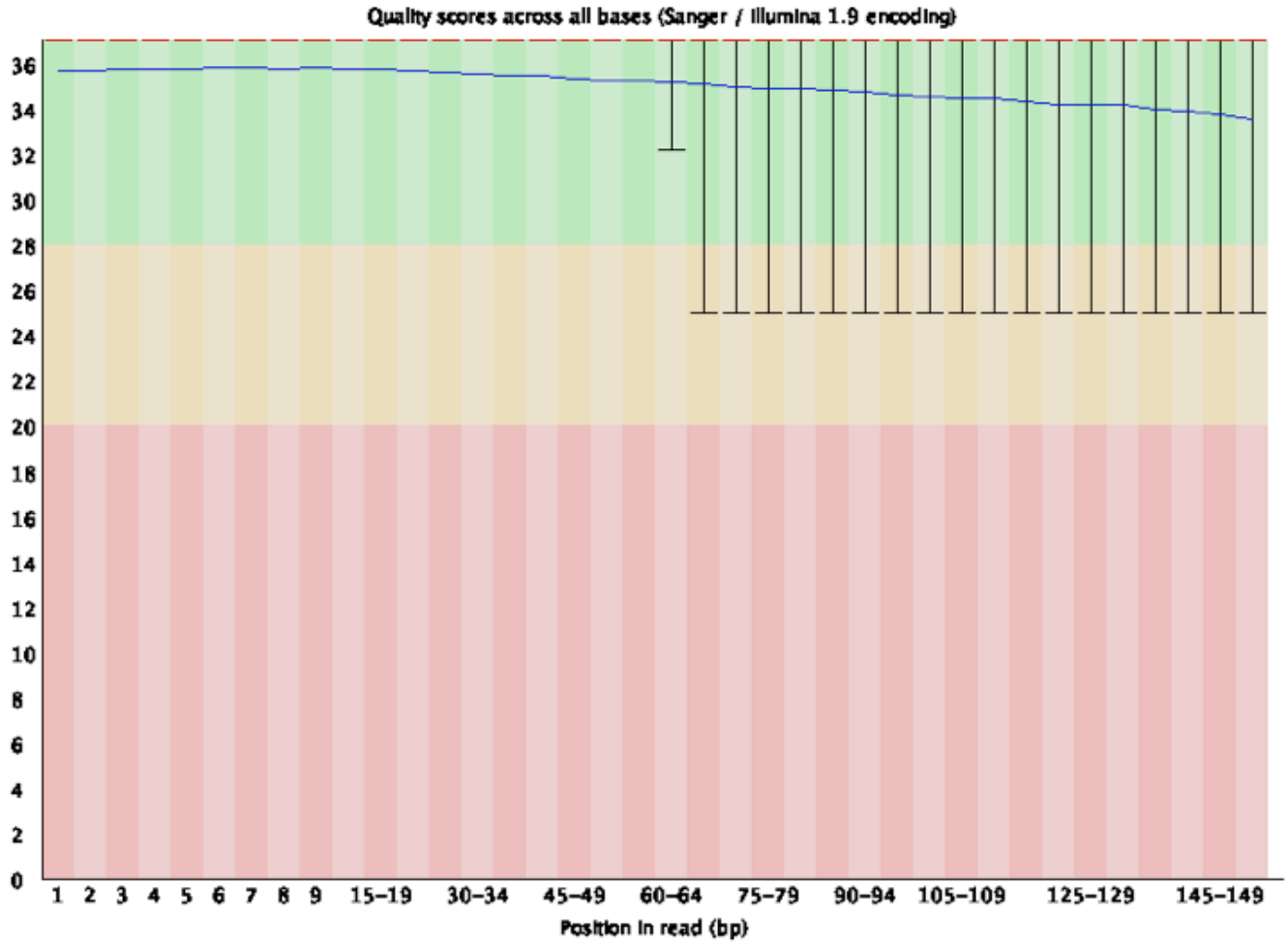


FastQC - High Quality Data NovaSeq (read 2)

1. Read 2 data is typically a bit lower in quality than read 1 data.

Per base

sequence quality

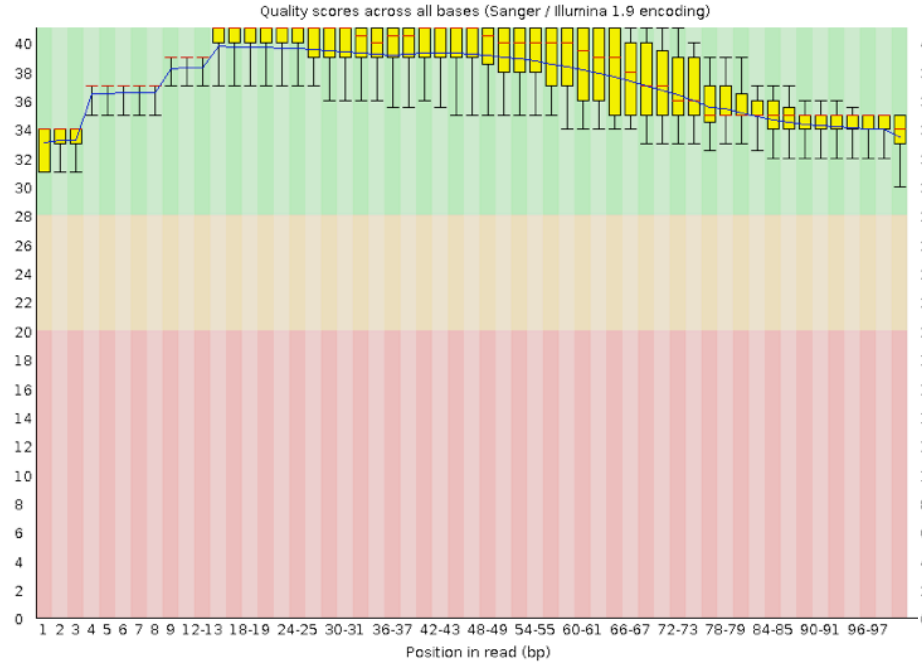


FastQC - High Quality Data HiSeq/MiSeq

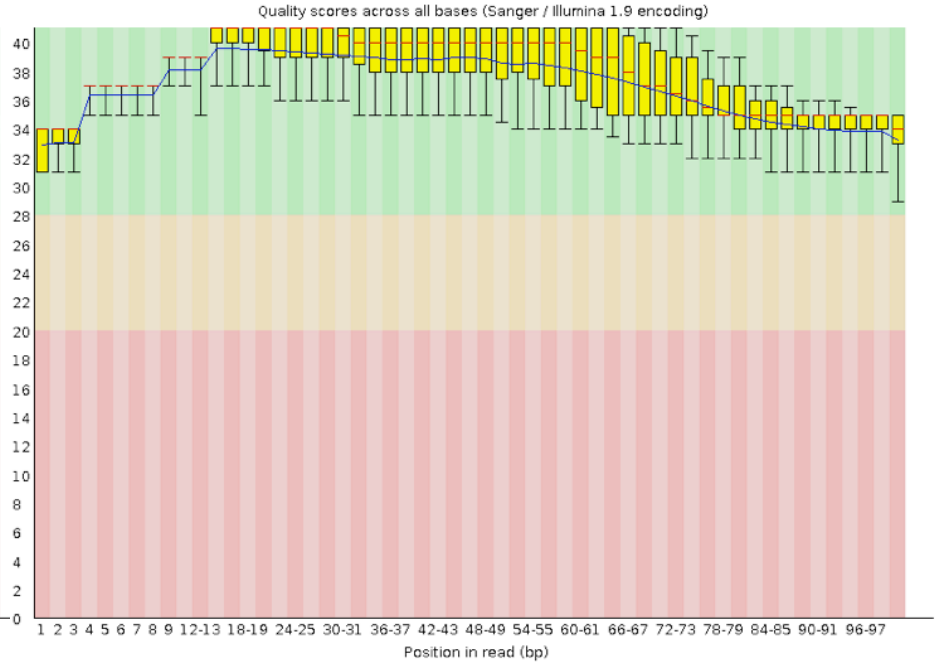
1. The Illumina Novaseq sequencer has data with simplified quality scores compared to HiSeq and MiSeq machines.

2. Below is HiSeq data (after it has been cleaned). Lots more gradations in quality scores are apparent.

Per base sequence quality: Read 1



Per base sequence quality: Read 2

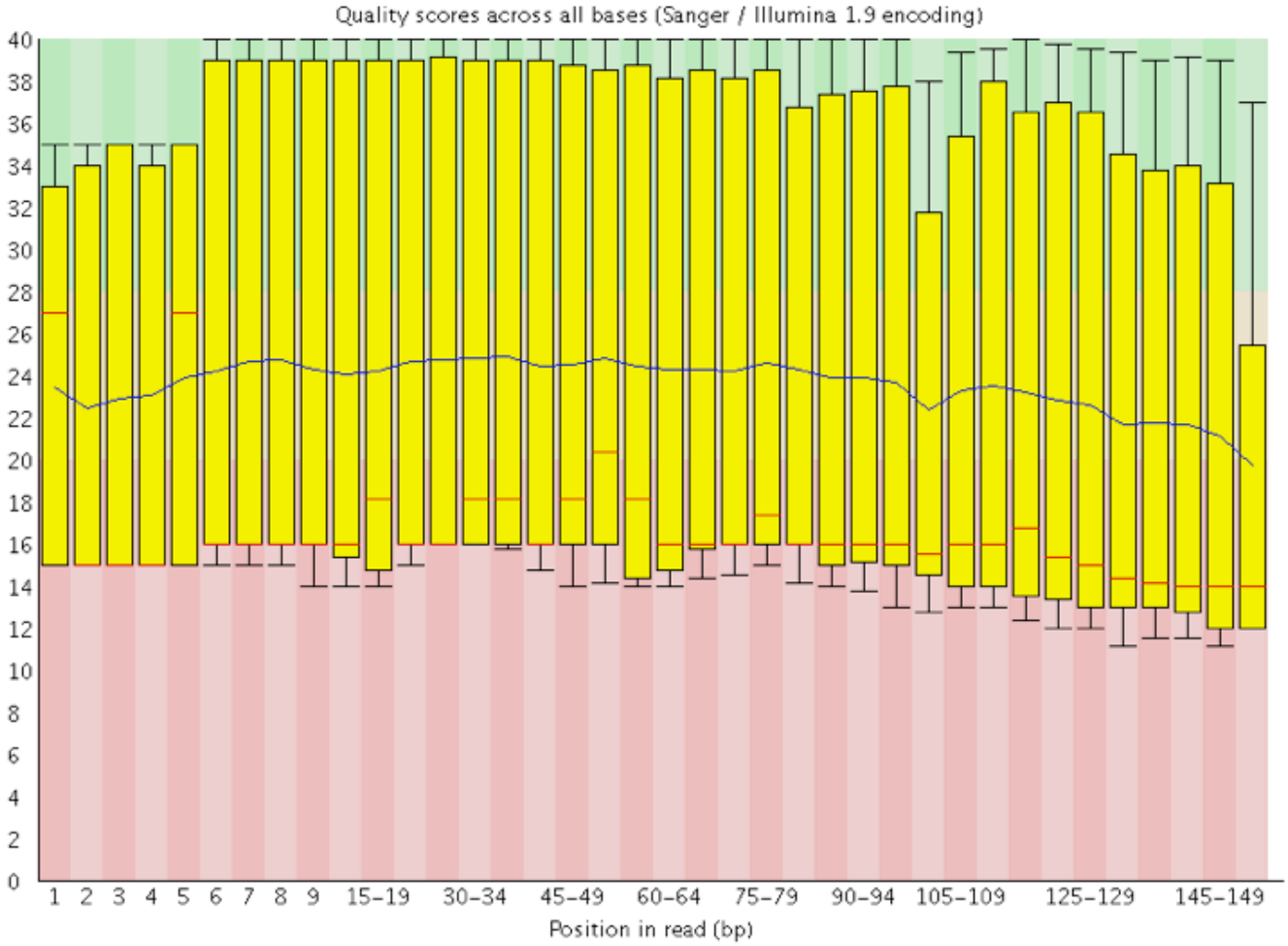


FastQC - Low Quality Data

1. Low sequence quality scores in this case resulted from non-specific amplification during the PCR of this 16S microbiome data.

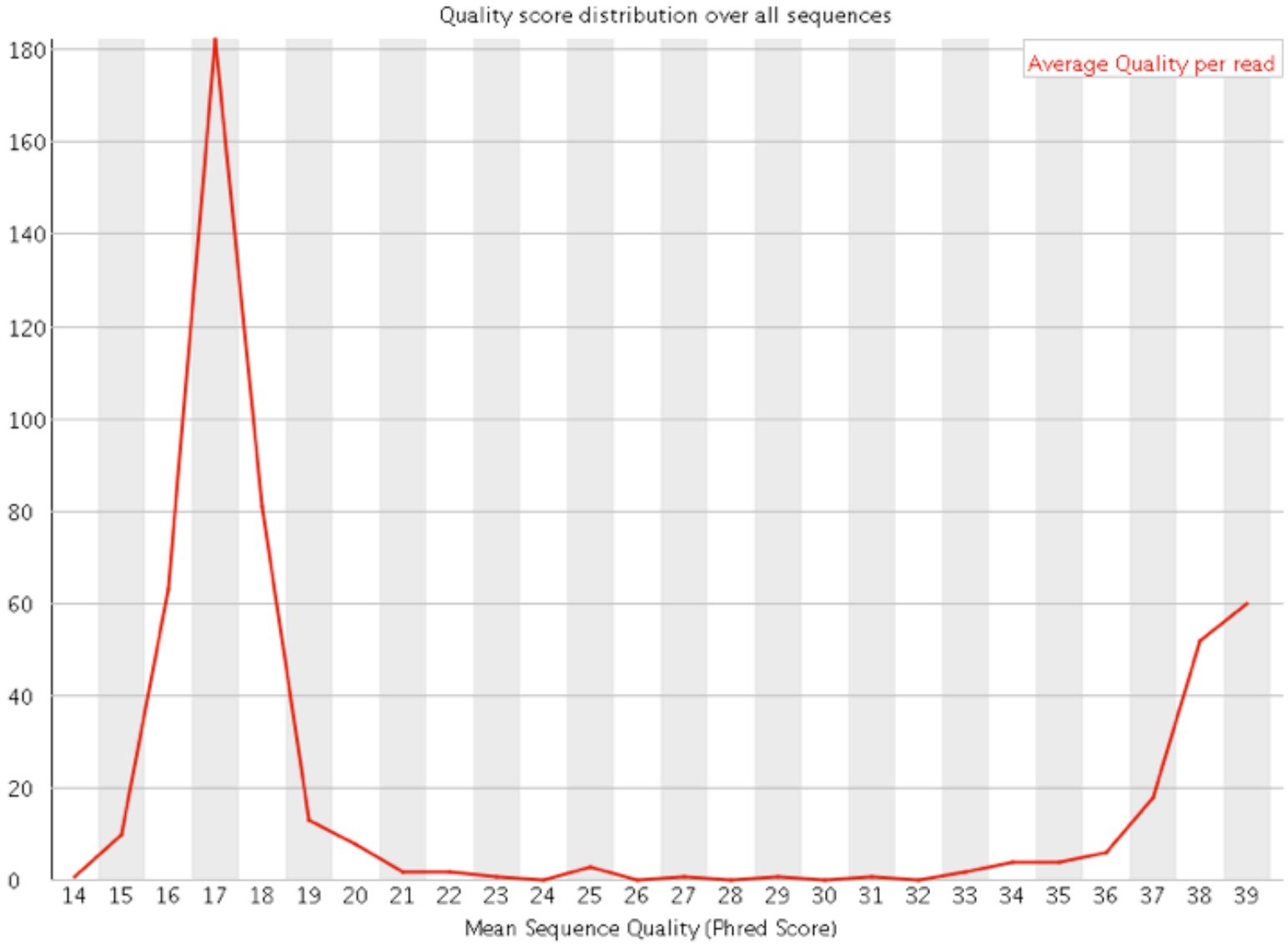
Per base

sequence quality



FastQC - Low Quality Data: Per sequence quality scores

1. Bimodal quality scores in this case resulted from non-specific amplification during the PCR of this 16S microbiome data.

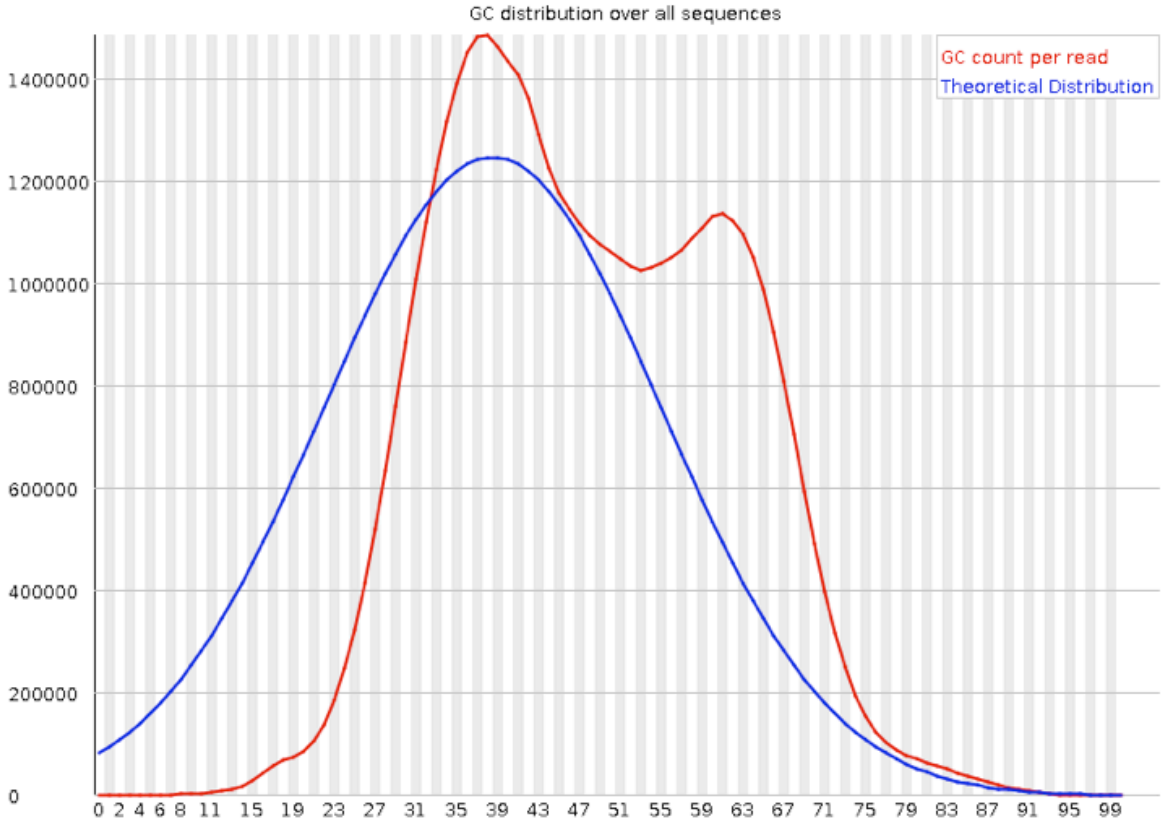


FastQC - Per sequence GC content

1. Expect roughly a normal distribution of GC content. The data (in red) is clearly different!

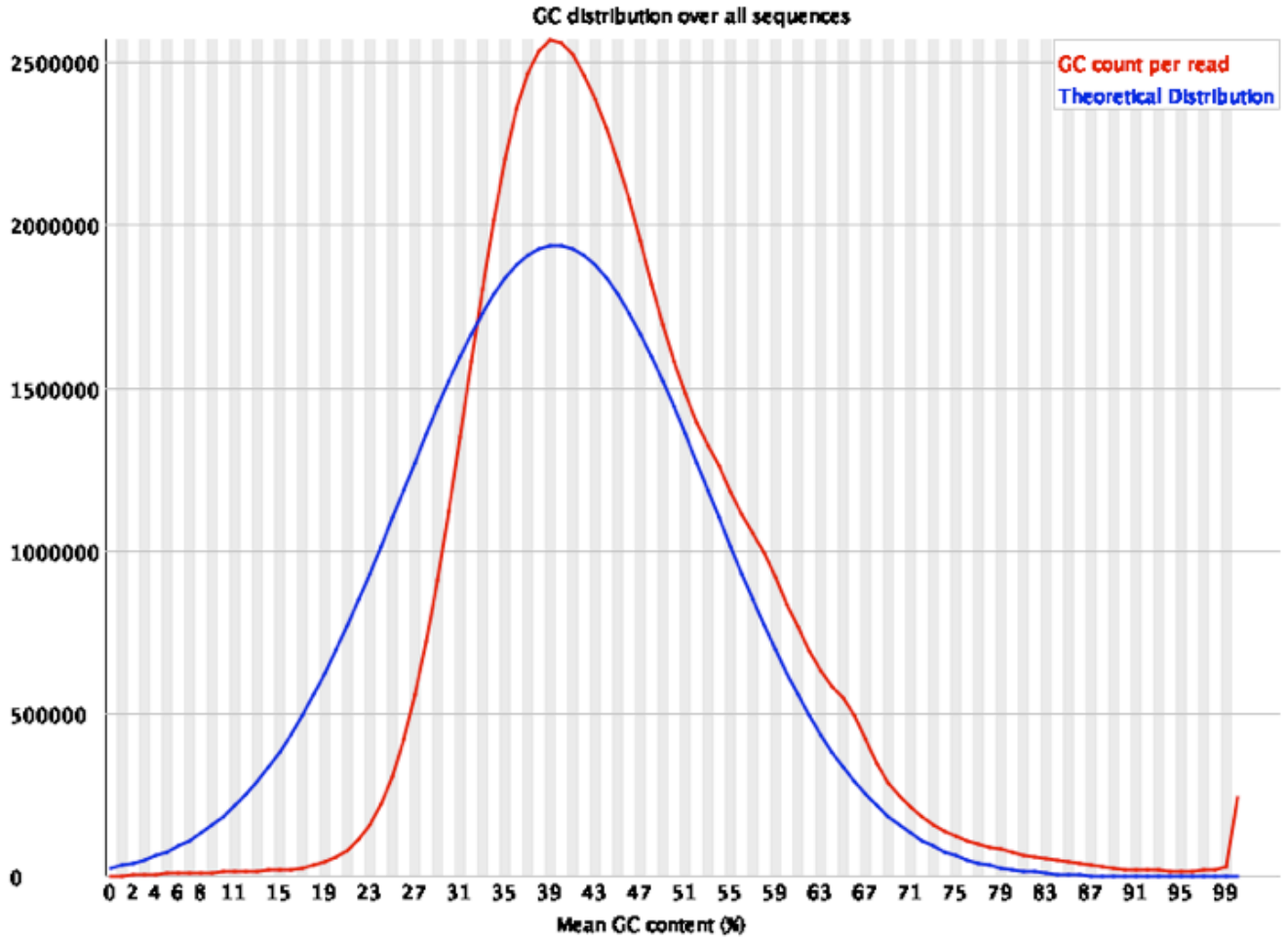
2. Potentially the sample had two genomes, the bird genome and a parasite genome.

Score = red x (ie bad)



FastQC - Per sequence GC content

1. This one doesn't seem as bad, but it was still flagged for GC content.



FastQC - Overrepresented sequences

1. The same sample had a string of Gs as an overrepresented sequence.
    a. This was a pattern across libraries, especially in the read 2 data.
2. Novaseq two color chemistry has a known issue with it, which is that it can call a “G” when really there is no signal.

https://sequencing.qcfail.com/articles/illumina-2-colour-chemistry-can-overcall-high-confidence-g-bases/

Overrepresented sequences

Sequence	Count	Percentage	Possible Source
GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG	493846	0.7430167594403236	No Hit

MultiQC

1. “Aggregate results from bioinformatics analyses across many samples into a single report. MultiQC searches a given directory for analysis logs and compiles a HTML report. It's a general use tool, perfect for summarizing the output from numerous bioinformatics tools.”

    a. MultiQC can summarize the output from FastQC, which is very useful if you are dealing with a large number of files.

2. Found at: https://multiqc.info/

    a. The website has videos on how to install and use the program.

    b. The website also lists the 150+ supported tools that it can summarize, which is a great resource for learning about new bioinformatic tools.

MultiQC Installation

1. Make a conda environment (need to have Miniconda or Anaconda installed)

conda create -n Bio

2. Activate that environment (need to do in each shell (ie in each Terminal window)

conda activate Bio (Can deactivate: “conda deactivate)

3. Add channels to look for the program

conda config --add channels defaults

conda config --add channels bioconda

conda config --add channels conda-forge

4. Installing MultiQC

conda install multiqc

5. Test it out

multiqc --help

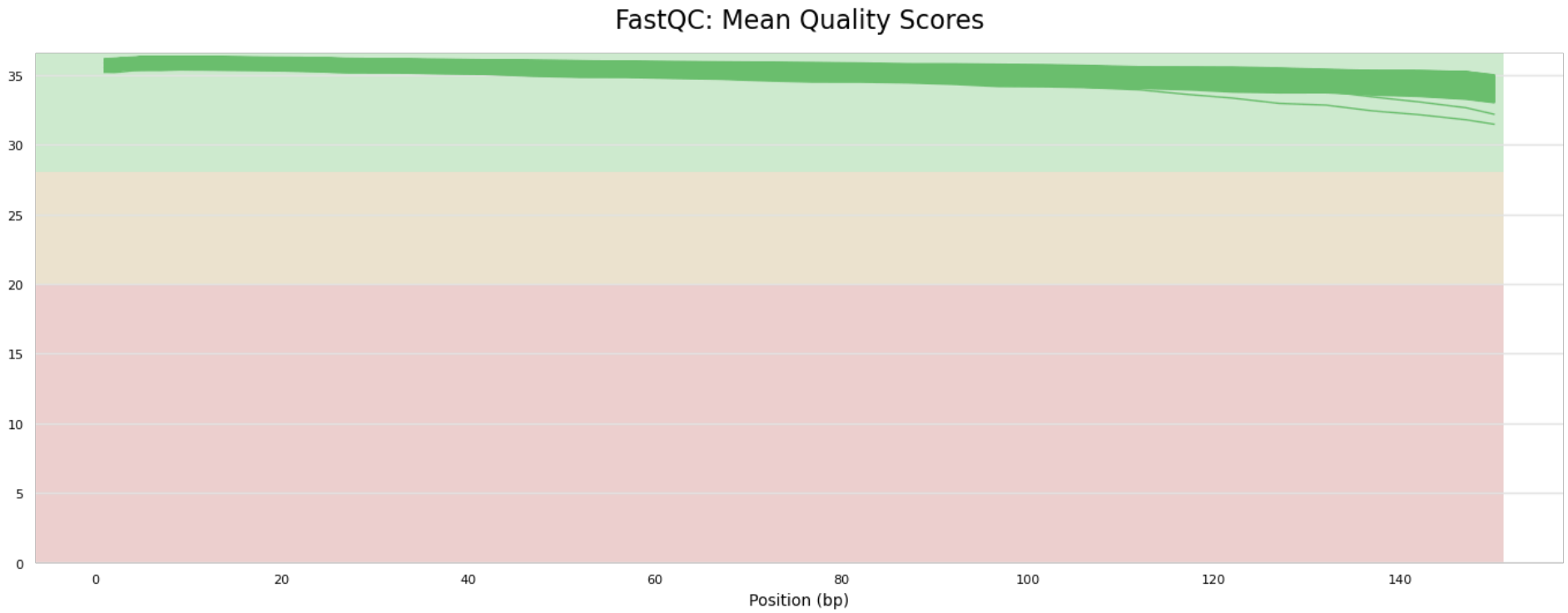
5. To run the program, navigate into the folder you want to scan in the terminal. Then type: multiqc .

a. Need to have the zip files output by FastQC for MultiQC to work.

MultiQC - FastQC Summaries

1. Sequence Quality Histograms: The mean quality value across each base position in the read.

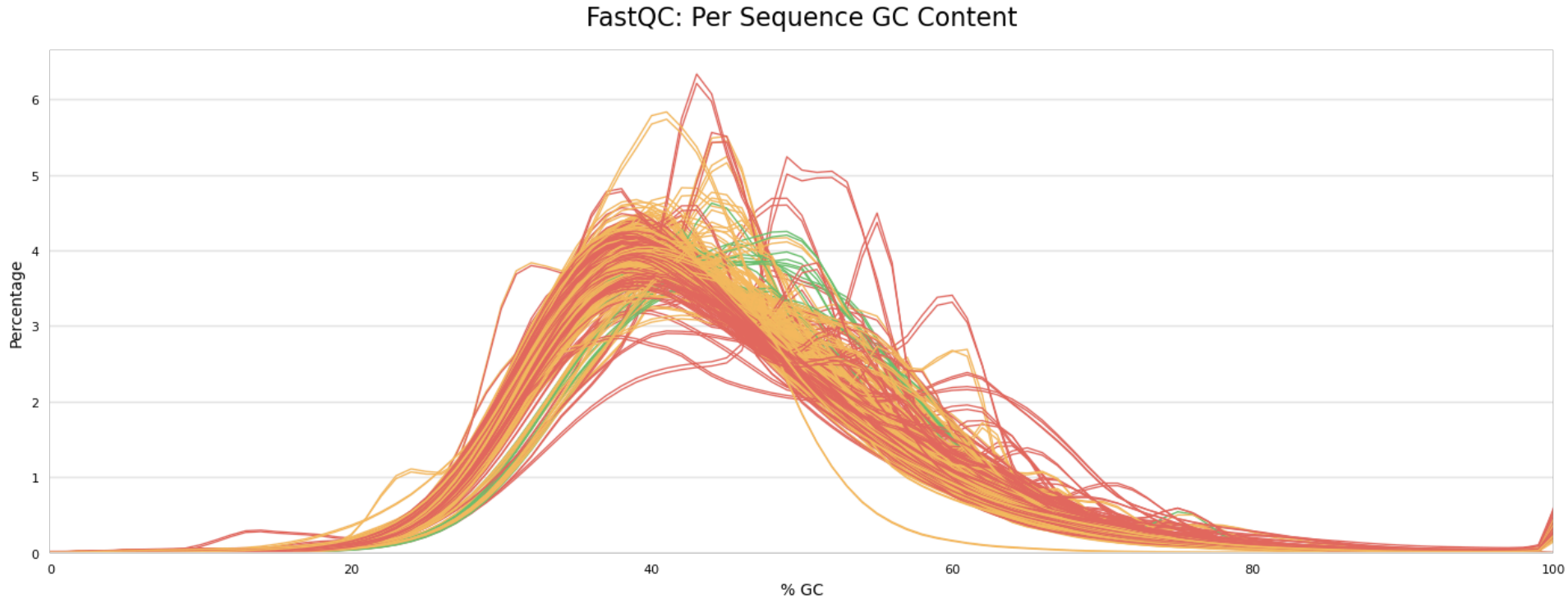
2. The plot below summarized the quality scores for 500 files. I could see that all of them were good quality.



MultiQC - FastQC Summaries

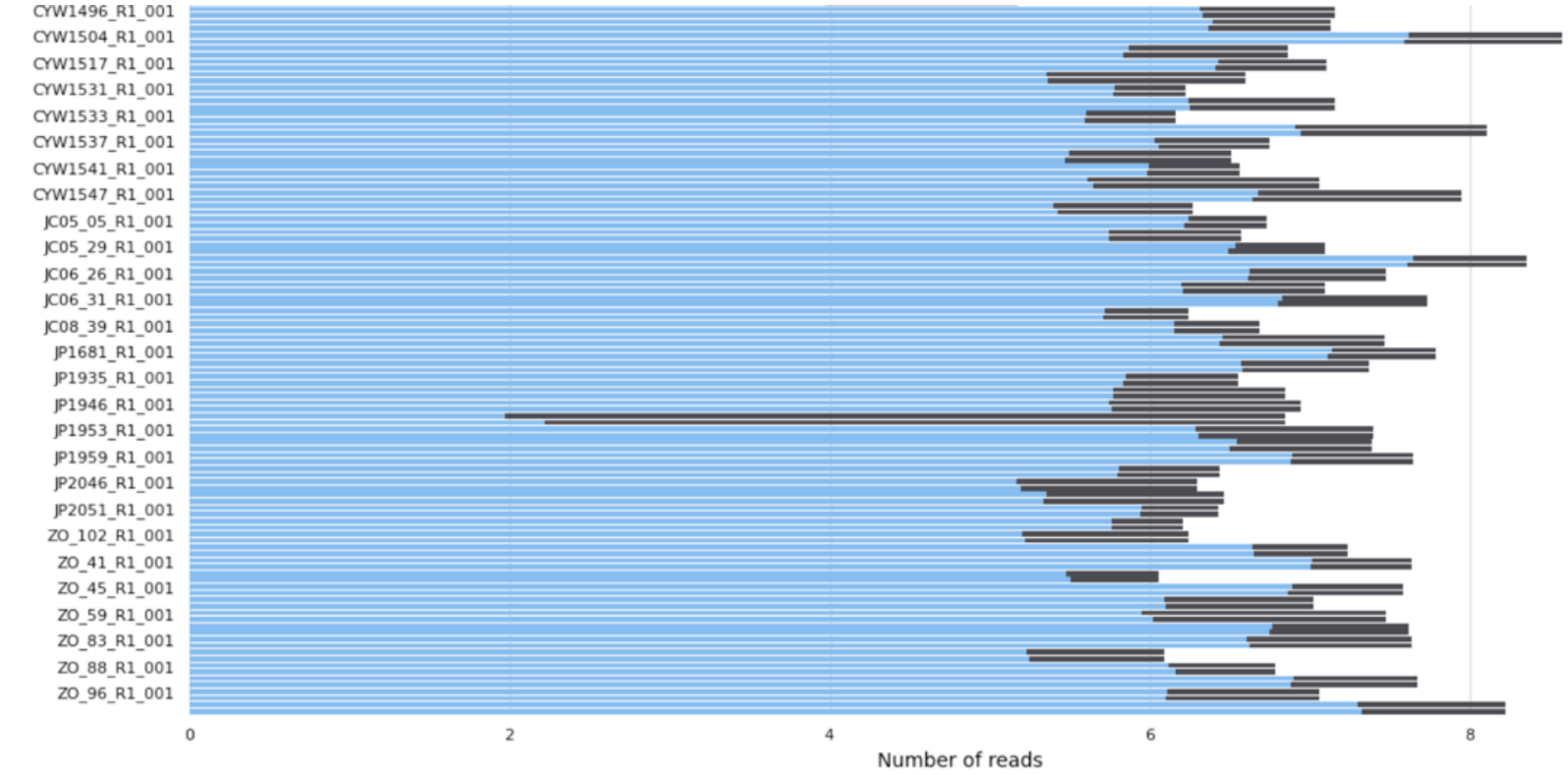
1. Per Sequence GC Content. The average GC content of reads. Normal random library typically have a roughly normal distribution of GC content.

2. Can see what outliers look like and how many there are.



MultiQC - FastQC Summaries

1. Sequence Counts. Sequence counts for each sample. Duplicate read counts (in gray) are an estimate only.



Questions

HTStream

1. HTStream can be found at:

https://s4hts.github.io/HTStream/

2. "HTStream is a quality control and processing pipeline for High Throughput Sequencing data. The difference between HTStream and other tools is that HTStream uses a tab delimited fastq format that allows for streaming from application to application. This streaming creates some awesome efficiencies when processing HTS data and makes it fully interoperable with other standard Linux tools."

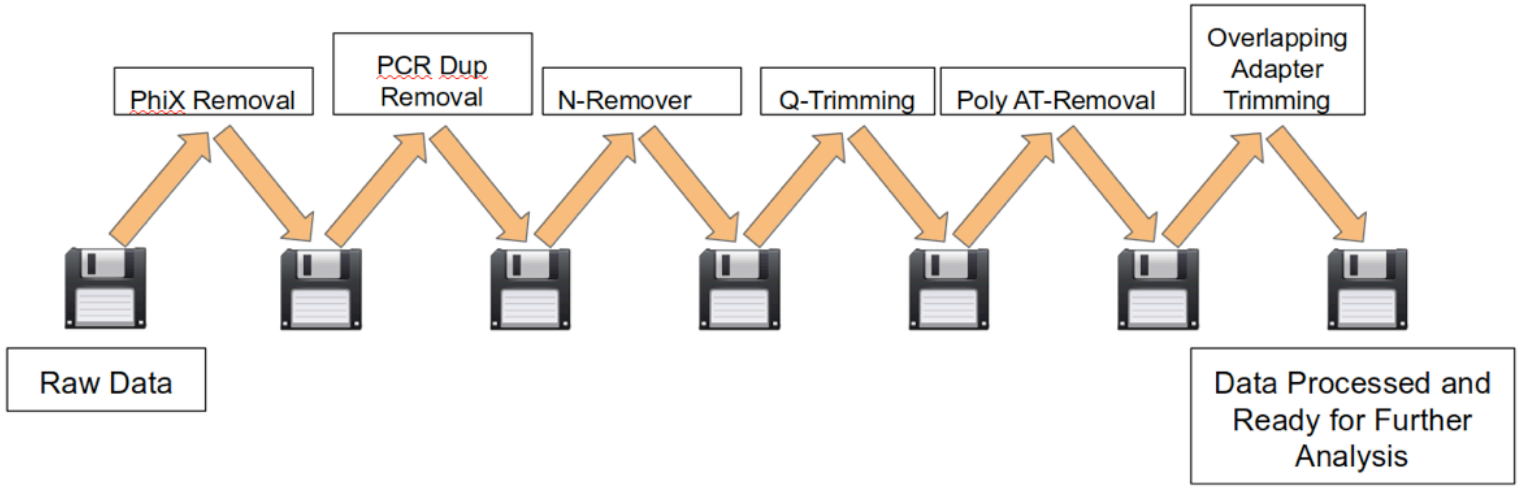
3. One big advantage is that it is just a single unified program for cleaning the data, so you have only one piece of software to install, learn, and troubleshoot.

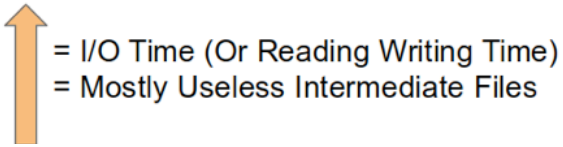
4. Benefits include no intermediate files (reduces storage footprint) and files that are only read in and written out once.

Benefits of HTStream

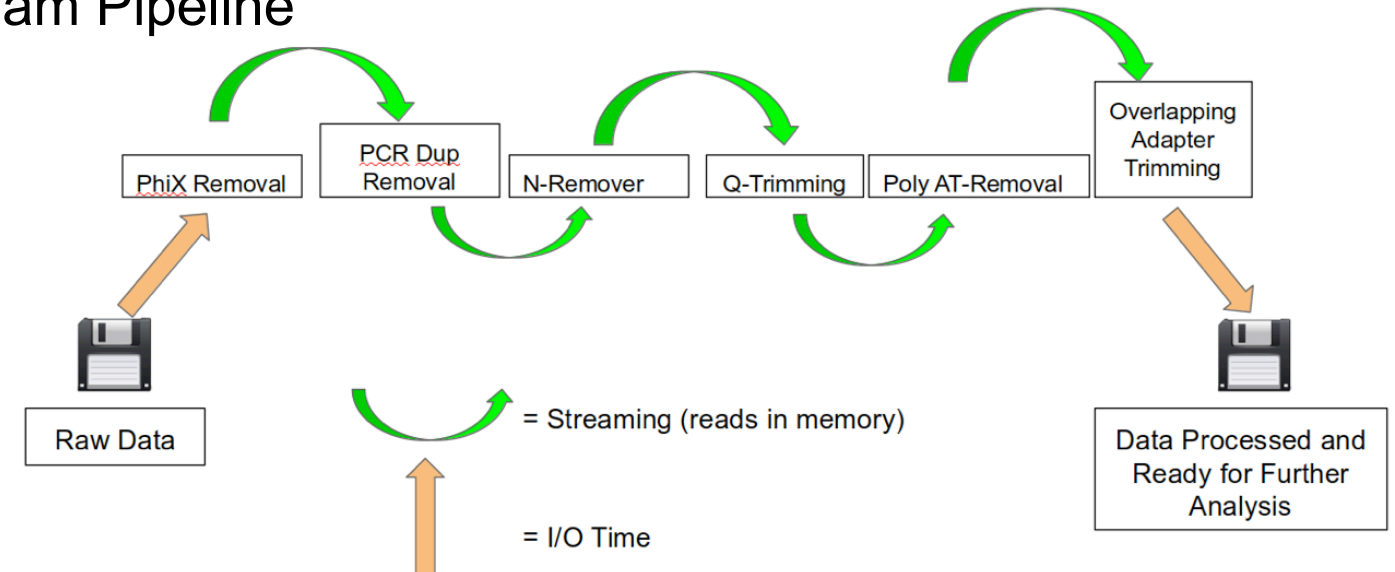
1. Traditional Pipeline

https://ucdavis-bioinformatics-training.github.io/2020-mRNA_Seq_Workshop/data_reduction/01-preproc_htstream_mm





2. HTStream Pipeline



Installing HTStream: Conda

1. Install via Python

    a. Anaconda: https://www.anaconda.com/products/individual

    b. Miniconda: https://docs.conda.io/en/latest/miniconda.html

2. “Bioconda is a channel for the conda package manager specializing in bioinformatics software.

The conda package manager makes installing software a vastly more streamlined process.”

    https://bioconda.github.io/

3. “Many people use conda environments to manage software. HTStream is available through the Bioconda channel.”

https://s4hts.github.io/HTStream/

Installing HTStream

1. In terminal type “conda install htstream” (Should still be in the conda Bio environment and have the channels configured.)

```
(base) Hummingbird:~ ammoncorl$ conda install htstream
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /opt/anaconda3

  added / updated specs:
    - htstream


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    boost-1.70.0               |   py38hbf1eeb5_1         347 KB  conda-forge
    boost-cpp-1.70.0           |       hd59e818_1        19.0 MB  conda-forge
    htstream-1.3.3             |       hbba97d9_0         1.3 MB  bioconda
    ------------------------------------------------------------
                                           Total:        20.7 MB

The following NEW packages will be INSTALLED:

  boost              conda-forge/osx-64::boost-1.70.0-py38hbf1eeb5_1
  boost-cpp          conda-forge/osx-64::boost-cpp-1.70.0-hd59e818_1
  htstream           bioconda/osx-64::htstream-1.3.3-hbba97d9_0


Proceed ([y]/n)?
```

Testing if HTStream Installed Correctly

1. In terminal type: “hts_Stats --help”
    a. If you get output from the program, it is installed!

```
ERROR: Input data absent

HTStream <https://github.com/s4hts/HTStream> application: hts_Stats
Version: 1.4.1
The hts_Stats app produce basic statistics about the reads in a dataset.
  Including the basepair composition and number of bases Q30.

Standard Options:
  -v [ --version ]                      Version print
  -h [ --help ]                         Prints help documentation
  -N [ --notes ] arg                    Notes for the stats JSON
  -L [ --stats-file ] arg (=stats.log)  Write to stats file name
  -A [ --append-stats-file ] arg        Append to stats file name

Input Options [default: tab6 format on stdin]:
  -1 [ --read1-input ] arg              Read 1 paired end fastq input <space
                                        separated for multiple files>
  -2 [ --read2-input ] arg              Read 2 paired end fastq input <space
                                        separated for multiple files>
  -U [ --singleend-input ] arg          Single end read fastq input <space
                                        separated for multiple files>
  -I [ --interleaved-input ] arg        Interleaved fastq input <space
                                        separated for multiple files>
  -o [ --qual-offset ] arg (=33)        Quality offset for ascii q-score
                                        (default is 33) (min 1, max 10000)
  -T [ --tab-input ] arg                Tab-delimited (tab6) input <space
                                        separated for multiple files>
```

HTStream Pipeline Overview

1. hts_Stats: get stats on raw reads

2. hts_SeqScreener: screen out (remove) phiX

3. hts_SuperDeduper: identify and remove PCR duplicates

4. hts_Overlapper: identify and remove adapter sequence, merge any reads that significantly overlap

5. hts_SeqScreener: screen for any remaining adapters

6. hts_QWindowTrim: remove poor quality sequence

7. hts_NTrimmer: remove any remaining N characters

8. hts_LengthFilter: use to remove all reads < 50bp

9. hts_Stats: get stats on output reads

Note: Add --help after any of these functions to learn about all their options. For example: hts_Stats --help

We will go into the details of all functions so that you can modify the code for your particular needs (and in case the program changes).

https://ucdavis-bioinformatics-training.github.io/2018-June-RNA-Seq-Workshop/tuesday/preproc.html
https://ucdavis-bioinformatics-training.github.io/2020-mRNA_Seq_Workshop/data_reduction/01-preproc_htstream_mm

HTStream Pipeline Workflow

1. I learned how to use HTStream from a bioinformatics workshop website at U.C. Davis.

   a. https://ucdavis-bioinformatics-training.github.io/2018-June-RNA-Seq-Workshop/tuesday/preproc.html

   https://ucdavis-bioinformatics-training.github.io/2020-mRNA_Seq_Workshop/data_reduction/01-preproc_htstream_mm

2. Most of the code that I will present is based on their code.

   a. I have added lots of comments to the code in the script.

   b. I modified the code for DNA sequencing rather than RNAseq (you should check out their code and my alternative RNAseq file if doing work with RNA).

   c. I made a few other minor modifications.

3. In short, the hard work was done by the UC Davis Bioinformatics Core and credit goes to them. I highly recommend looking at their course pages.

hts_Stats

1. The hts_Stats app produce basic statistics about the reads in a dataset. Including the basepair composition and number of bases Q30. (This and subsequent descriptions come from the help file for each app.)

2. Options that the script uses:

a. For the log, we specify -L with the same log file name for all routines, and use -A for the second routine onward to append log output, generating a single log file at the end.

b. Use -N to make a note in the log file

c. Use -1 and -2 to specify the original reads

d. Pipe to next part of pipeline: |

e. New line in script: \

f. A variable in the script: ${something}

3. Code: hts_Stats -L ${outpath}/HTS_stats_log_files/${sample}_htsStats.log -N 'initial stats' -1 ${inpath}/${sample}*R1* -2 ${inpath}/${sample}*R2* | \

https://ucdavis-bioinformatics-training.github.io/2018-June-RNA-Seq-Workshop/tuesday/preproc.html
https://ucdavis-bioinformatics-training.github.io/2020-mRNA_Seq_Workshop/data_reduction/01-preproc_htstream_mm
HTStream help files

hts_SeqScreener

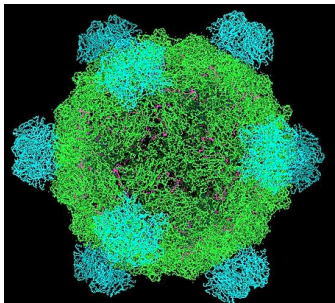
1. hts_SeqScreener identifies and removes any reads which appear to have originated from a contaminant DNA source. Because bacteriophage Phi-X is commonly spiked into Illumina runs for QC purposes, sequences originating from Phi-X are removed by default. If other contaminants are suspected their sequence can be supplied as a fasta file, however the algorithm has been tuned for short contaminant sequences, and may not work well with sequences significantly longer than Phi-X (5Kb).

2. Phi-X Fun Facts!

   a. The phi X 174 (or ΦX174) bacteriophage is a single-stranded DNA (ssDNA) virus that infects *Escherichia coli,* and the first DNA-based genome to be sequenced. This work was completed by Fred Sanger and his team in 1977.[1]

   b. Nobel prize winner Arthur Kornberg used ΦX174 as a model to first prove that DNA synthesized in a test tube by purified enzymes could produce all the features of a natural virus, ushering in the age of synthetic biology.[3][4]

   c. In 2003, it was reported by Craig Venter's group that the genome of ΦX174 was the first to be completely assembled in vitro from synthesized oligonucleotides.[6]



https://en.wikipedia.org/wiki/Phi_X_174

hts_SeqScreener

3. The first call of SeqScreener in the pipeline filters out the Phi-X sequence, which is the default.

4. Options that the script uses

    a. Remember that -A is to append to the log file

5. Code: hts_SeqScreener -A ${outpath}/HTS_stats_log_files/${sample}_htsStats.log -N 'screen phix' | \

https://ucdavis-bioinformatics-training.github.io/2018-June-RNA-Seq-Workshop/tuesday/preproc.html
https://ucdavis-bioinformatics-training.github.io/2020-mRNA_Seq_Workshop/data_reduction/01-preproc_htstream_mm
HTStream help files

hts_SuperDeduper

1. hts_SuperDeduper is a reference-free PCR duplicate remover. It uses a subsequence within each read as a unique key to detect duplicates in future reads. Reads with 'N' character(s) in the key sequence are ignored. hts_SuperDeduper is not recommended for single-end reads. WARNING: hts_SuperDeduper will only work correctly on untrimmed reads.

    a. Thus, only use this for paired end reads and before adapter trimming.

2. Options that the script uses

    a. The -e flag is the "Frequency in which to log duplicates in reads, can be used to create a saturation plot (0 turns off)." If -e = 250000, this thus records how many duplicates are in the first 250000 reads, then the next 250000 reads and so on.

    b. You may want to adjust the value for -e depending on your read number.

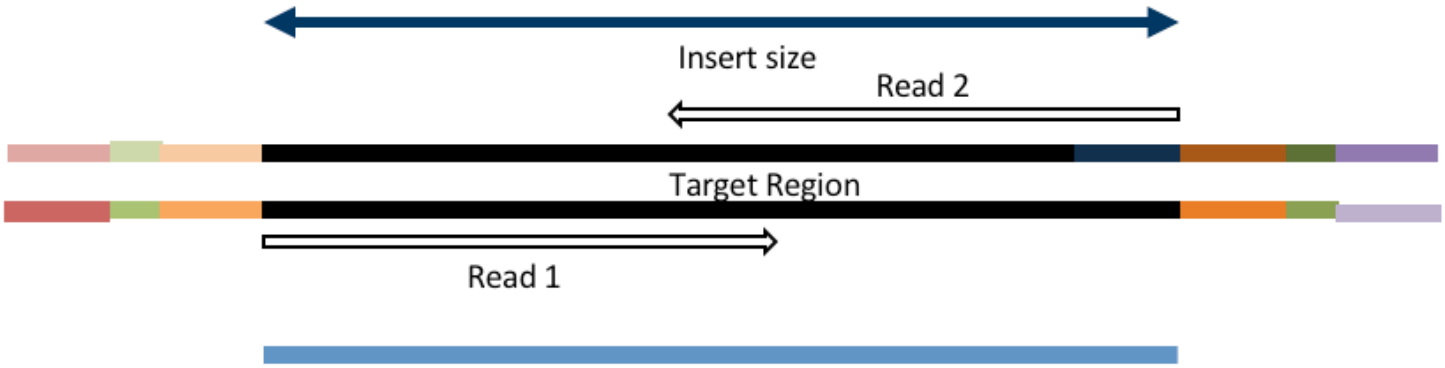
3. Code: hts_SuperDeduper -e 250000 -A ${outpath}/HTS_stats_log_files/${sample}_htsStats.log -N 'remove PCR duplicates' | \

https://ucdavis-bioinformatics-training.github.io/2018-June-RNA-Seq-Workshop/tuesday/preproc.html
https://ucdavis-bioinformatics-training.github.io/2020-mRNA_Seq_Workshop/data_reduction/01-preproc_htstream_mm
HTStream help files

hts_Overlapper

1. Sometimes your paired reads overlap. It is standard to combine them into a single extended read.

    a. These reads are no longer labeled as R1 and R2. They are now SE, for single end.



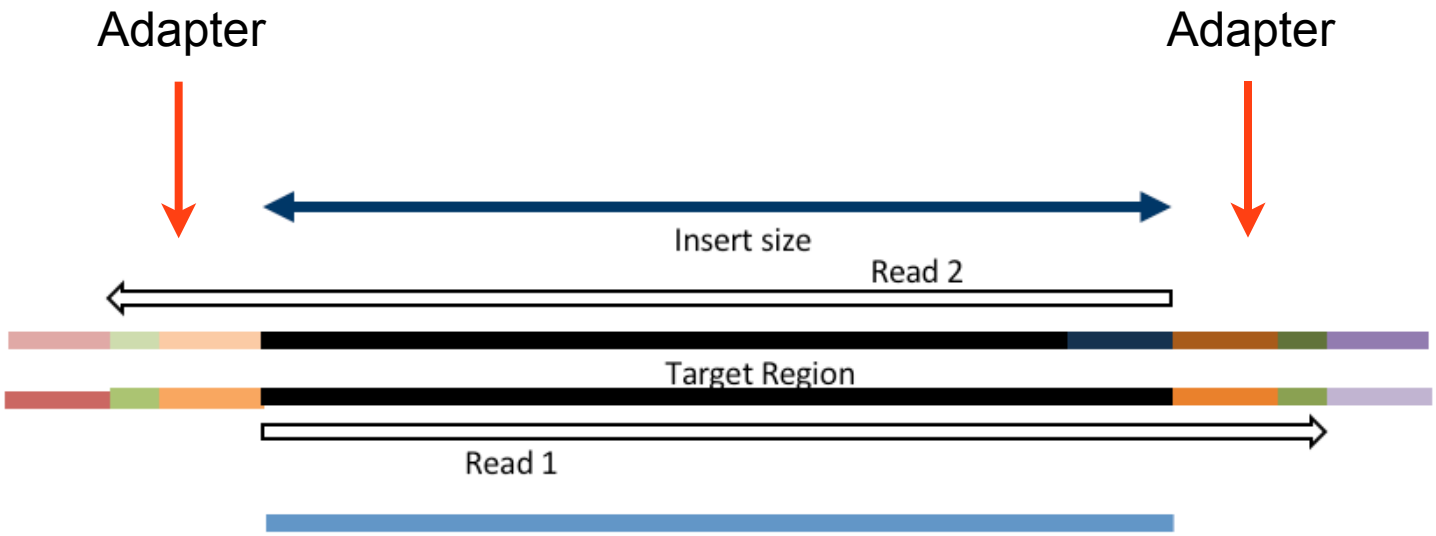
Size of resulting SE read

https://ucdavis-bioinformatics-training.github.io/2020-mRNA_Seq_Workshop/data_reduction/01-preproc_htstream_mm
HTStream help files

hts_Overlapper

1. Sometimes your DNA insert is so small that you sequence beyond the DNA of your species into the adapter sequence.

    a. Want to trim off the adapter sequences



https://ucdavis-bioinformatics-training.github.io/2020-mRNA_Seq_Workshop/data_reduction/01-preproc_htstream_mm
HTStream help files

hts_Overlapper

1. The hts_Overlapper application attempts to overlap paired end reads to produce the original transcript, trim adapters, and in some cases, correct sequencing errors. If the reads overlap it will export a single merged read. If the reads do not overlap significantly, it will export PE reads.

    a. If you want to maintain the PE format, use Adapter Trimmer instead, which exports PE data. It trims off adapters by first overlapping paired-end reads and trimming off overhangs which by definition are adapter sequence in standard libraries.

    b. Maintain the PE format for RNAseq data (also use hts_PolyATTrim).

2. We have already covered the options that the script uses.

3. Code: hts_Overlapper -A ${outpath}/HTS_stats_log_files/${sample}_htsStats.log -N 'trim adapters and overlap' | \

4. Alternative code for RNAseq data:

hts_AdapterTrimmer -A ${outpath}/HTS_stats_log_files/${sample}_htsStats.log -N 'trim adapters' | \

hts_PolyATTrim -A ${outpath}/HTS_stats_log_files/${sample}_htsStats.log -N 'remove polyAT tails' | \

https://ucdavis-bioinformatics-training.github.io/2018-June-RNA-Seq-Workshop/tuesday/preproc.html
https://ucdavis-bioinformatics-training.github.io/2020-mRNA_Seq_Workshop/data_reduction/01-preproc_htstream_mm
HTStream help files

hts_SeqScreener

1. hts_SeqScreener identifies and removes any reads which appear to have originated from a contaminant DNA source.

2. The second call of SeqScreener filters out any remaining adapter sequences. The HT Stream website says "This tool can also be useful in removing primer dimers and other reads containing sequencing adapters. For example, setting -k 15 -x .01 in combination with a collection of adapters in fasta format, has been found to work well for this purpose."

3. Options that the script uses

    a. -s to specify the file to screen against, in this case the adapters file.

4. Code: hts_SeqScreener -s ${adapters} -k 15 -x .01 -A ${outpath}/HTS_stats_log_files/${sample}_htsStats.log -N 'screen remaining adapters' | \

https://ucdavis-bioinformatics-training.github.io/2018-June-RNA-Seq-Workshop/tuesday/preproc.html
https://ucdavis-bioinformatics-training.github.io/2020-mRNA_Seq_Workshop/data_reduction/01-preproc_htstream_mm
HTStream help files

hts_QWindowTrim

1. hts_QWindowTrim uses a sliding window approach to remove low quality bases (5' or 3') from a read. A window will slide from each end of the read, moving inwards. Once the window reaches an average quality <avg-qual> it will stop trimming.

    a. I used the defaults of a window size of 10 and an average quality threshold of 20.

2. We have already covered the options that the script uses and defaults don't need to be specified.

3. Code: hts_QWindowTrim -A ${outpath}/HTS_stats_log_files/${sample}_htsStats.log -N 'quality trim the ends of reads' | \

https://ucdavis-bioinformatics-training.github.io/2018-June-RNA-Seq-Workshop/tuesday/preproc.html
https://ucdavis-bioinformatics-training.github.io/2020-mRNA_Seq_Workshop/data_reduction/01-preproc_htstream_mm
HTStream help files

hts_NTrimmer

1. The hts_NTrimmer application will identify and return the longest subsequence that no N characters appear in.

    a. An “N” stands for an unknown base.

2. We have already covered the options that the script uses.

3. Code: hts_NTrimmer -A ${outpath}/HTS_stats_log_files/${sample}_htsStats.log -N 'remove any remaining N characters' | \

https://ucdavis-bioinformatics-training.github.io/2018-June-RNA-Seq-Workshop/tuesday/preproc.html
https://ucdavis-bioinformatics-training.github.io/2020-mRNA_Seq_Workshop/data_reduction/01-preproc_htstream_mm
HTStream help files

hts_LengthFilter

1. The hts_LengthFilter application filters reads that are too long or too short.

    a. -m flag is the minimum length for acceptable output read (min 1, max 10000), default is unset

    b. Use -m to remove all reads < 20bp .

3. Code: hts_LengthFilter -A ${outpath}/HTS_stats_log_files/${sample}_htsStats.log -N 'remove reads < 20bp' -m 20 | \

https://ucdavis-bioinformatics-training.github.io/2018-June-RNA-Seq-Workshop/tuesday/preproc.html
https://ucdavis-bioinformatics-training.github.io/2020-mRNA_Seq_Workshop/data_reduction/01-preproc_htstream_mm
HTStream help files

hts_Stats

1. The hts_Stats app produce basic statistics about the reads in a dataset. Including the basepair composition and number of bases Q30.

    a. We run it again to get stats on the output file

2. The new options that the script uses are:

    a. -f flag means output Fastq files

3. Code: hts_Stats -A ${outpath}/HTS_stats_log_files/${sample}_htsStats.log -N 'final stats' -f ${outpath}/${sample}"

https://ucdavis-bioinformatics-training.github.io/2018-June-RNA-Seq-Workshop/tuesday/preproc.html
https://ucdavis-bioinformatics-training.github.io/2020-mRNA_Seq_Workshop/data_reduction/01-preproc_htstream_mm
HTStream help files

Code for running hts_Stream

```
for sample in `cat ${sample_names}`

do

    echo "SAMPLE: ${sample}"

  call="hts_Stats -L ${outpath}/HTS_stats_log_files/${sample}_htsStats.log -N 'initial stats' -1 ${inpath}/${sample}*R1* -2 ${inpath}/${sample}*R2* | \

        hts_SeqScreener -A ${outpath}/HTS_stats_log_files/${sample}_htsStats.log  -N 'screen phix' | \

        hts_SuperDeduper -e 250000 -A ${outpath}/HTS_stats_log_files/${sample}_htsStats.log -N 'remove PCR duplicates' | \

        hts_Overlapper -A ${outpath}/HTS_stats_log_files/${sample}_htsStats.log  -N 'trim adapters and overlap' | \

        hts_SeqScreener -s ${adapters} -k 15 -x .01 -A ${outpath}/HTS_stats_log_files/${sample}_htsStats.log -N 'screen remaining adapters' | \

        hts_QWindowTrim -A ${outpath}/HTS_stats_log_files/${sample}_htsStats.log -N 'quality trim the ends of reads'  | \

        hts_NTrimmer -A ${outpath}/HTS_stats_log_files/${sample}_htsStats.log -N 'remove any remaining N characters'  | \

        hts_LengthFilter -A ${outpath}/HTS_stats_log_files/${sample}_htsStats.log  -N 'remove reads < 20bp' -m 20 | \

        hts_Stats -A ${outpath}/HTS_stats_log_files/${sample}_htsStats.log -N 'final stats'  -f ${outpath}/${sample}"

  echo $call

  eval $call

done
```

Parts of code that you need to change (in bold)

```
#!/bin/bash

    ## assumes htstream is available on the Pathway or in your conda environment

start=`date +%s`

echo $HOSTNAME

    ##Name of the directory for the raw data

inpath='Hummingbird110220'

    ##Name of the directory for the processed data

outpath='HB1_cleaned'

    ##Name of the file of samples

sample_names='hummingbird_plate1_libraries_shortname.txt'

    ##Name of the file of adapter sequences to screen against

adapters='hummingbird_capture_adapter_sequences.txt'

    ##Makes a directory to store the processed data

[[ -d ${outpath} ]] || mkdir ${outpath}

    ##Makes a separate directory to store the log files

mkdir ${outpath}/HTS_stats_log_files
```

Sample Names File

1. Text file with each sample name prefix on a separate line.
    a. Keep the names simple
    b. Only have a single new line at the end of the file.
2. Example:

Pis10

Pis11

2. Here is what the file names look like for the data:

Pis10_S10_L001_R1_001.fastq.gz

Pis10_S10_L001_R2_001.fastq.gz

Pis11_S11_L001_R1_001.fastq.gz

Pis11_S11_L001_R2_001.fastq.gz

Adapters File

1. Format of the file: Fasta file with header on one line, sequence on the other

2. Example:

>Illumina–TruSeq_DNA_UDI_sequence_for_adapter_trimming_read1

AGATCGGAAGAGCACACGTCTGAACTCCAGTCA

>Illumina–TruSeq_DNA_UDI_sequence_for_adapter_trimming_read2

AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT

3. Need to include these two sequences if cleaning Illumina reads. At one point I also added in the full adapter sequences in additional lines, which does not hurt but is probably not necessary.

Example of a full adapter sequence.

>IDT_384_UID_TruSeq_Plate1_Well_A1_forward,reverse_complement

AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGTCTCATCAGGTGTAGATC
TCGGTGGTCGCCGTATCATT

Running the script

1. In the terminal, navigate to the directory containing the folder with your raw data (but not into the raw data folder).

2. Add the information for the inpath, outpath, sample names, and adapters to the hts_preproc.sh script in a text editing program.

3. Run the script: screen -S name_of_screen bash hts_preproc.sh

4. Check the folder of cleaned reads to see if things are working.

5. Detach the screen.

6. Wait until it is done.

   a. Can type: screen -ls in the terminal window to check if the screen is still active. If your screen is not listed there, the program has finished running.

7. What if something goes wrong?

   a. Reattach the screen: screen -r name_of_screen

   b. Cancel the commands: Control C

Run the script in parallel

1. To speed things up, you can process your data in parallel (assuming adequate computing resources).

2. Make multiple sample names files each with different samples.

3. Make multiple processing scripts, each with referencing the different sample names files.

Example: hts_preproc1.sh, hts_preproc2.sh

4. Run the scripts in different terminal windows or run one script, detach the screen, and then run the next script.

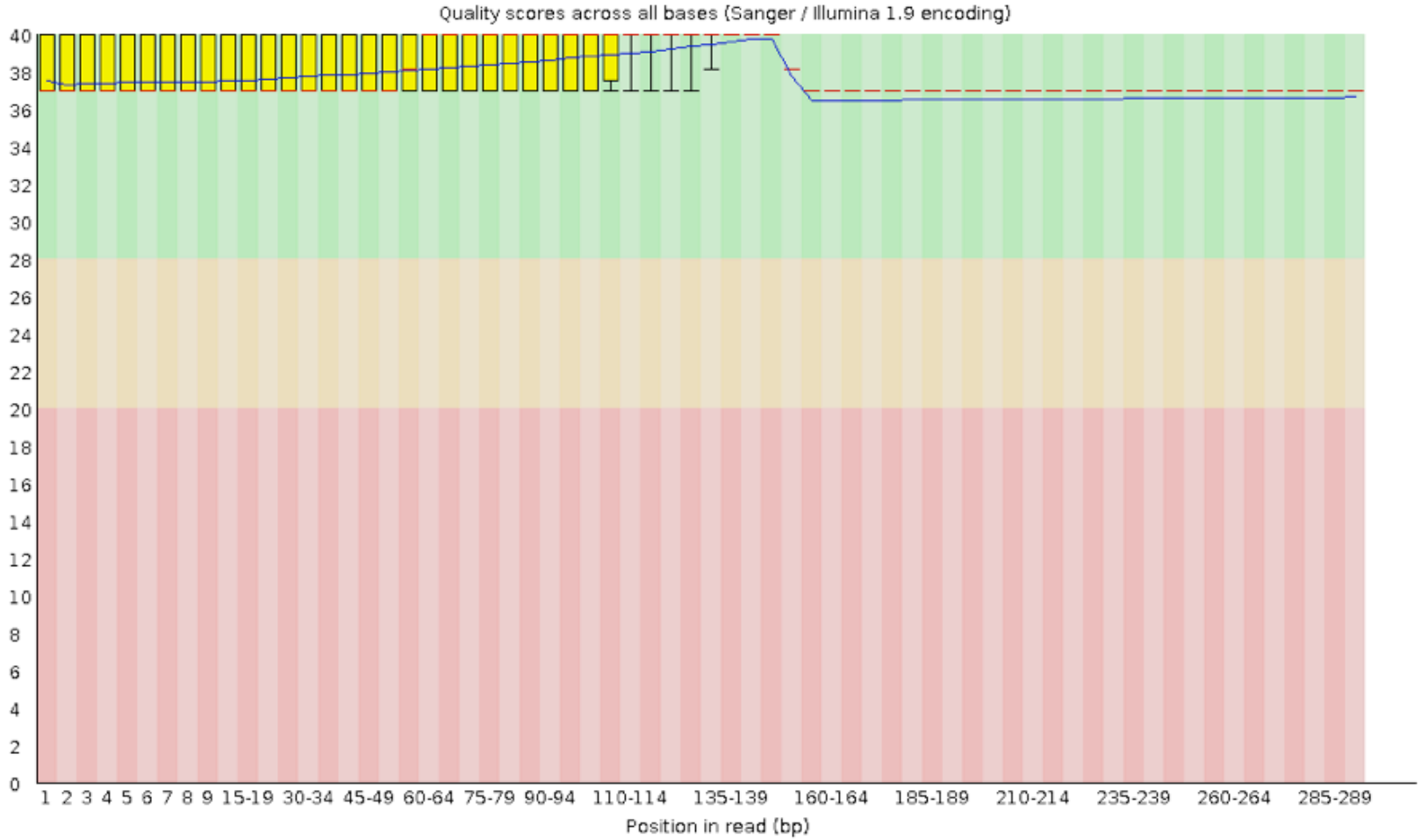
    a. You will need to activate the conda environment in each terminal window.

Questions

After Running HTStream - FastQC

1. In addition to the read 1 (R1) and read 2 (R2) files, you will have a file of the merged sequences (SE).

   a. The length and distribution of quality scores will look different because of the merger.



After Running HTStream - htsStats

1. When running HTStream, htsStats saved information to the log files about the whole read cleaning process.

    a. The output is in JSON format, which is a human readable format that stores attributes with values.

2. You can look at the data in the log files to figure out how your data files changed during each step of the HTStream processing.

3. Can use the data to test how what you did in the lab affected your data, such as:

    a. Number of PCR cycles and duplication rates

    b. Insert sizes and amount of overlap in the data

    c. Amount of input DNA and amount of sequence data for the sample.

htsStats Log File - initial stats


```
{ "hts_Stats_51528": {
    "Notes": "",
    "totalFragmentsInput": 1017471,
    "totalFragmentsOutput": 1017471,
    "R1_readlength_histogram": [ [151,1017471] ],
    "R2_readlength_histogram": [ [151,1017471] ],
    "Base_composition": {
        "A": 63723061,
        "C": 89922440,
        "G": 90795498,
        "T": 62831787,
        "N": 3456
    },
    "Single_end": {
        "SE_in": 0,
        "SE_out": 0,
        "SE_bpLen": 0,
        "SE_bQ30": 0
    },
    "Paired_end": {
        "PE_in": 1017471,
        "PE_out": 1017471,
        "R1_bpLen": 153638121,
        "R1_bQ30": 146143244,
        "R2_bpLen": 153638121,
        "R2_bQ30": 146474662
    }
```

htsStats Log File - hts_Overlapper


```
}, "hts_Overlapper_51531": {
    "Notes": "",
    "totalFragmentsInput": 451763,
    "totalFragmentsOutput": 451763,
    "sins": 60833,
    "mins": 175789,
    "lins": 215141,
    "adapterTrim": 60833,
    "adapterBpTrim": 2236787,
    "readlength_histogram": [ [8,17], [9,25], [10,13],
    "Single_end": {
        "SE_in": 0,
        "SE_out": 236622,
        "SE_discarded": 0
    },
    "Paired_end": {
        "PE_in": 451763,
        "PE_out": 215141,
        "R1_discarded": 0,
        "R2_discarded": 0,
        "PE_discarded": 0
    }
```

Summarizing htsStats in R

1. I wrote an R script called: summarize_hts_stats_v1.3.2_Corl_10-30-21.R

    a. This summarizes just part of the information in the log files.

2. The UC Davis course provided an R script “summary_stats.R” that produces a summary table of all the stats.

    a. I have used a modified version of this script on an earlier version of HTStream, but the new version changed the output format.

    a. The script assumes exact htstream operations and order as described above. Therefore it needs to be modified if you do something different as we did.

3. The UC Davis Bioinformatics group also has some R scripts for processing the log files and outputting some graphs.

    a. https://github.com/ucdavis-bioinformatics/HTStream_reports

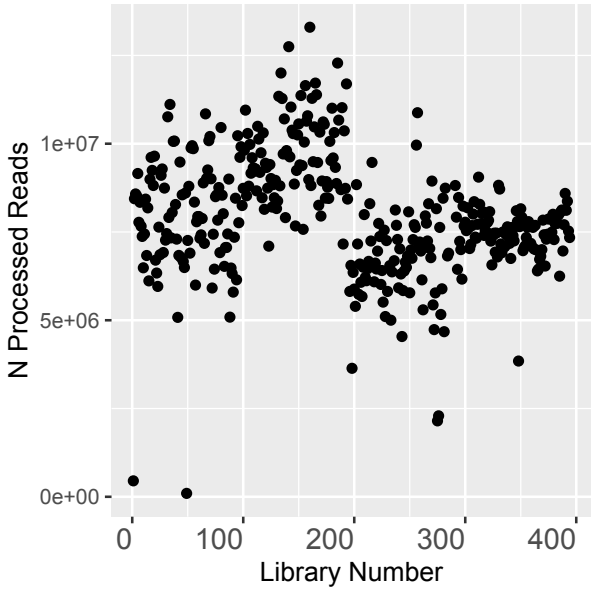
4. I will show some output from my R script.

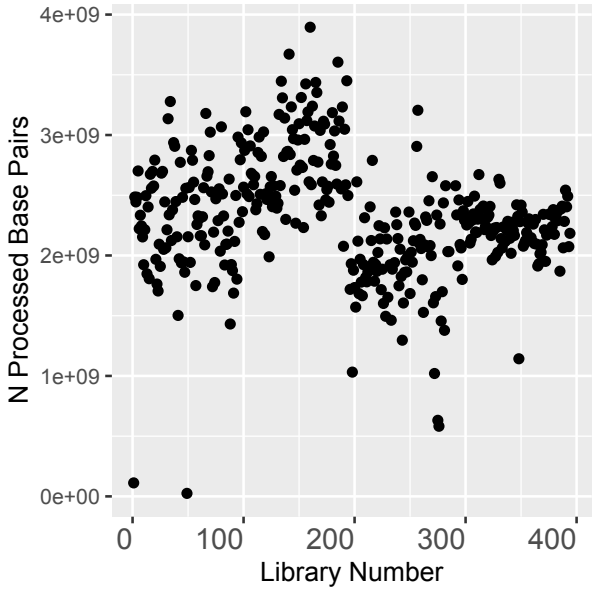
    a. HTStream says that it is working on becoming part of MultiQC.

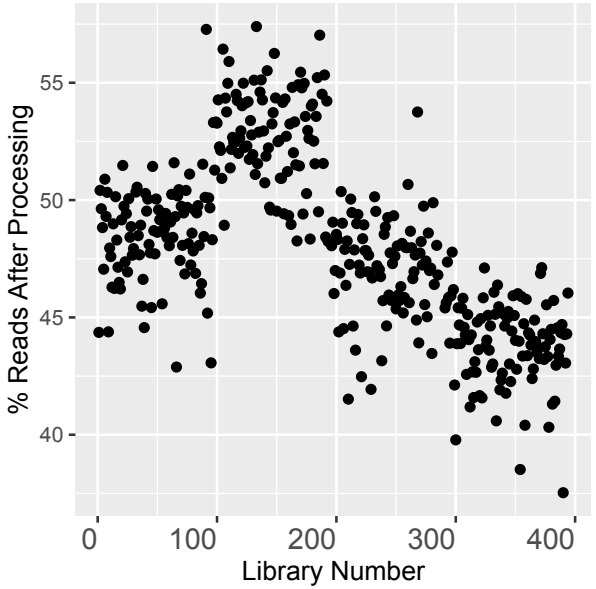
https://ucdavis-bioinformatics-training.github.io/2018-June-RNA-Seq-Workshop/tuesday/preproc.html

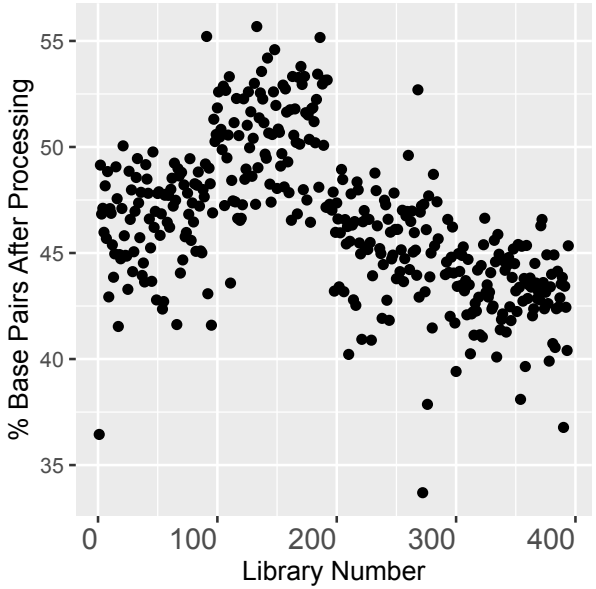
Summarizing htsStats in R

1. Various plots of how much data remains after processing.







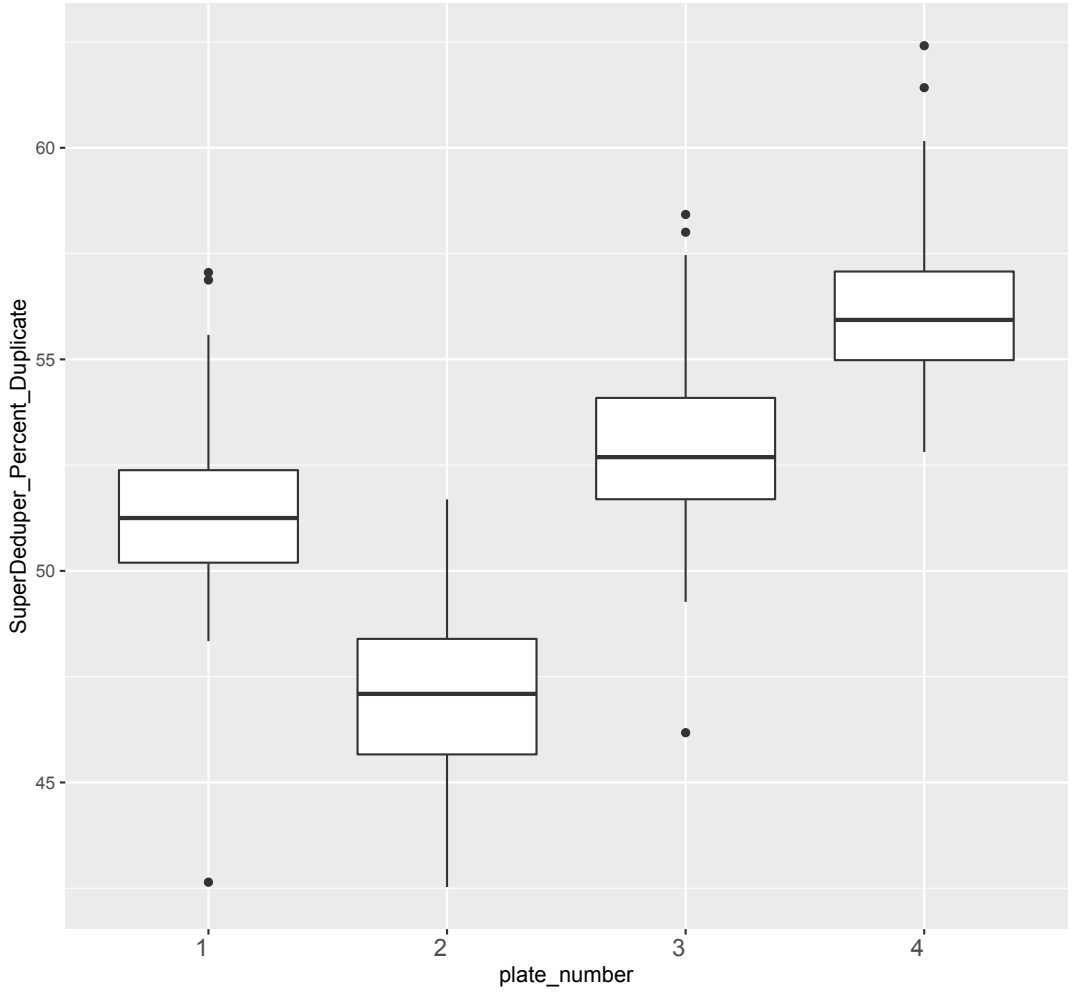


Summarizing htsStats in R

1. Exon capture data. Can have PCR duplicates. Novaseq has lots of optical duplicates.

   a. Can go into any of the steps and see how much it affects the final data.

Percent Duplicate Reads



Putting it all into practice

1. I have provided a folder of Illumina HiSeq sequence reads that were generated as part of my project on plasticity and adaptation at the Pisgah Lava Flow (Corl et al. 2018).

    a. I took a small subset of the data for ten lizards from the lava flow.

Putting it all into practice

1. Try doing the following after installing the programs:

   a. Run FastQC on original data: source ~/.bash_rc or: source ~/.bash_profile

      bash run_fastqc_keep_zip.sh -i input_folder_name -o output_folder_name

   b. Summary of results with MultiQC: multiqc .

   c. HTStream: bash hts_preproc_Pisgah_reads_final.sh

   (I also gave a generic verison: hts_preproc.sh)

   d. Post-cleaning FastQC: run_fastqc_keep_zip.sh

   e. Post-cleaning MultiQC: multiqc .

   f. Summarize the HTStream log files in R:
   summarize_hts_stats_v1.3.2_Corl_10-30-21.R

   g. Compare the status of the reads before and after cleaning with the MultiQC plots and the log files from HTStream. Evaluate whether the cleaning helped.

2. Please note that nstalling the programs is often a particularly frustrating step.

Learning Together

1. Have you liked this workshop? Feedback is appreciated.

2. I have thought that we could all benefit from a regular series of workshops (maybe one every month or two).

   a. We could learn from the expertise of our fellow scientists.

   b. Grad students and postdocs can get some experience as an instructor of their own class.

   c. This could be a good way to promote interactions among the labs.

   d. Would open up new ways to analyze our data.