The 16S Workshop

Claire Duvallet
Thomas Gurry, Scott Olesen

March 9, 2016

after this workshop, I hope you will understand:

- the steps involved in processing 16S data
- what the AWS pipeline can do for you
- how to troubleshoot some pipeline troubles
  - and 16S processing in general
- where to find resources for help
  - mostly your eyes and your brain

the AWS pipeline is not trying to be qiime

- AWS pipeline will one day be much more than qiime
  - different data types
  - automatic analyses
- having an in-house pipeline offers other advantages
  - looking inside the black box
  - adding functionalities
  - troubleshooting with humans
- qiime is great and you should totally use it if you want
  - but make sure you're using it wisely

“4” “easy” “steps” with 16S data

0. figure out what data you have

   your eyes and your brain

1. pre-process data

   AWS pipeline

2. build OTU table

   AWS pipeline

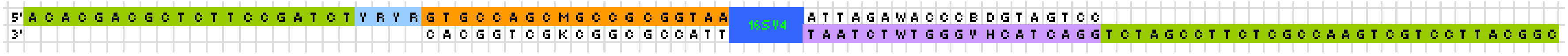
3. analyze

   your eyes and your brain

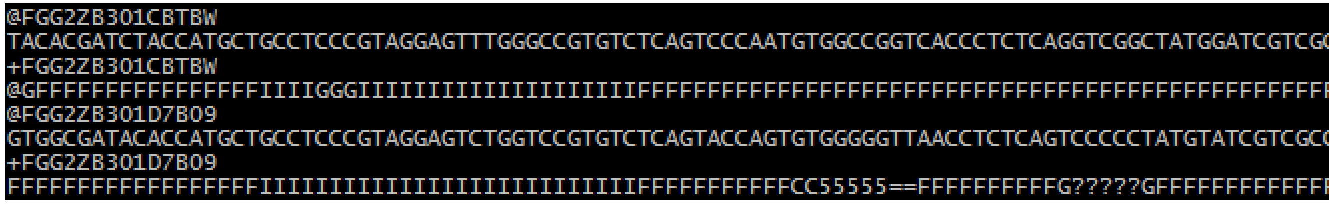
step 0. know your data

How were your regions sequenced?

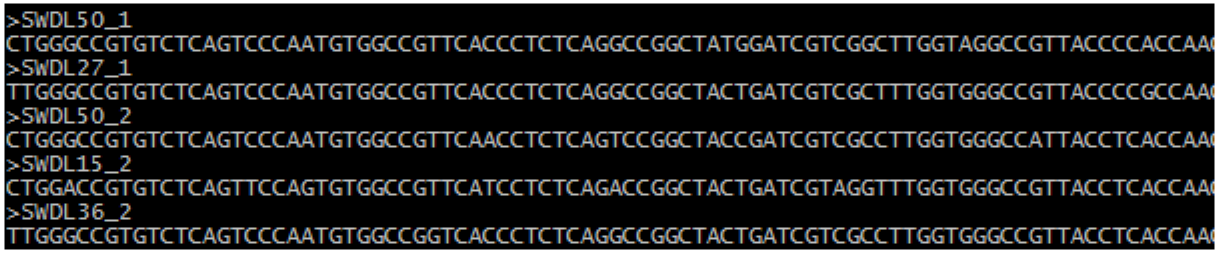
barcode – fwd primer – region – (rev primer)


5' ACACGACGCTCTTCCGATCT YRYR GTGCCAGCMGCCGCGGTAA 16S V4 ATTAGAWACCCBDGTAGTCC
3' CACGGTCGKCGGCGCCATT TAATCTWTGGGVHCATCAGG TCTAGCCTTCTCGCCAAGTCGTCCTTACGGC
16S V4

What kind(s) of file(s) do you have?



```
@FGG2ZB301CBTBW
TACACGATCTACCATGCTGCCTCCCGTAGGAGTTTGGGCCGTGTCTCAGTCCCAATGTGGCCGGTCACCCTCTCAGGTCGGCTATGGATCGTCG
+FGG2ZB301CBTBW
@GFFFFFFFFFFFFFFFFIIIIGGGIIIIIIIIIIIIIIIIIIIIIFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
@FGG2ZB301D7B09
GTGGCGATACACCATGCTGCCTCCCGTAGGAGTCTGGTCCGTGTCTCAGTACCAGTGTGGGGGTTAACCTCTCAGTCCCCCTATGTATCGTCGC
+FGG2ZB301D7B09
FFFFFFFFFFFFFFFFFFIIIIIIIIIIIIIIIIIIIIIIIIIIIIIFFFFFFFFFFFFCC55555==FFFFFFFFFFG?????GFFFFFFFFFFFFF
```

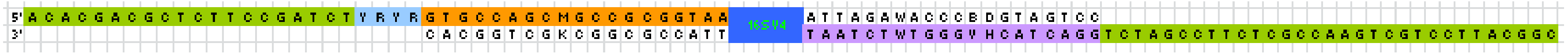


```
>SWDL50_1
CTGGGCCGTGTCTCAGTCCCAATGTGGCCGTTCACCCTCTCAGGCCGGCTATGGATCGTCGGCTTGGTAGGCCGTTACCCCACCAA
>SWDL27_1
TTGGGCCGTGTCTCAGTCCCAATGTGGCCGTTCACCCTCTCAGGCCGGCTACTGATCGTCGCTTTGGTGGGCCGTTACCCCGCCAA
>SWDL50_2
CTGGGCCGTGTCTCAGTCCCAATGTGGCCGTTCAACCTCTCAGTCCGGCTACCGATCGTCGCCTTGGTGGGCCATTACCTCACCAA
>SWDL15_2
CTGGACCGTGTCTCAGTTCCAGTGTGGCCGTTCATCCTCTCAGACCGGCTACTGATCGTAGGTTTGGTGGGCCGTTACCTCACCAA
>SWDL36_2
TTGGGCCGTGTCTCAGTCCCAATGTGGCCGGTCACCCTCTCAGGCCGGCTACTGATCGTCGCCTTGGTGGGCCGTTACCTCACCAA
```

step 0. know your data

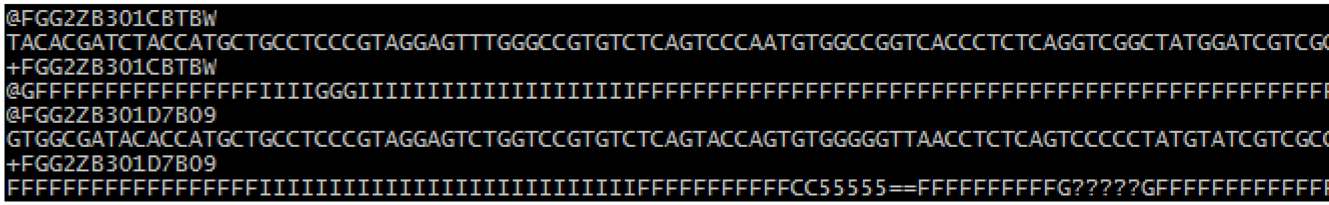
How were your regions sequenced?

barcode – fwd primer – region – (rev primer)


5' A C A C G A C G C T C T T C C G A T C T Y R Y R G T G C C A G C M G C C G C G G T A A
3' C A C G G T C G K C G G C G C C A T T
16S V4
A T T A G A W A C C C B D G T A G T C C
T A A T C T W T G G G V H C A T C A G G T C T A G C C T T C T C G C C A A G T C G T C C T T A C G G C

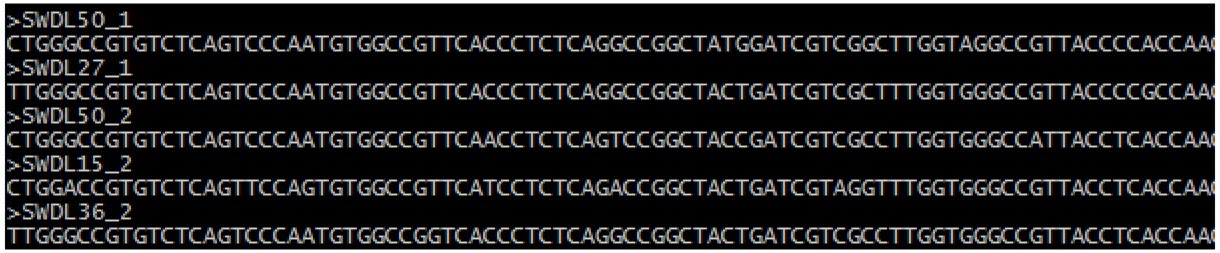
What kind(s) of file(s) do you have?

fastq



```
@FGG2ZB301CBTBW
TACACGATCTACCATGCTGCCTCCCGTAGGAGTTTGGGCCGTGTCTCAGTCCCAATGTGGCCGGTCACCCTCTCAGGTCGGCTATGGATCGTCG
+FGG2ZB301CBTBW
@GFFFFFFFFFFFFFFFFIIIIGGGIIIIIIIIIIIIIIIIIIIIIFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
@FGG2ZB301D7B09
GTGGCGATACACCATGCTGCCTCCCGTAGGAGTCTGGTCCGTGTCTCAGTACCAGTGTGGGGGTTAACCTCTCAGTCCCCCTATGTATCGTCG
+FGG2ZB301D7B09
FFFFFFFFFFFFFFFFFFIIIIIIIIIIIIIIIIIIIIIIIIIIIIFFFFFFFFFFFFCC55555==FFFFFFFFFFG?????GFFFFFFFFFFFFF
```

fasta



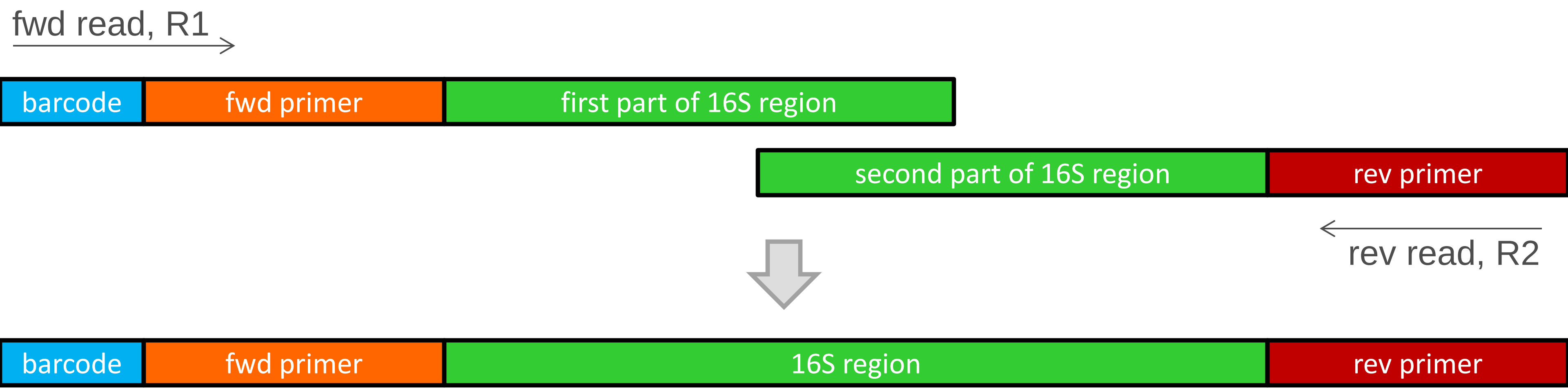
```
>SWDL50_1
CTGGGCCGTGTCTCAGTCCCAATGTGGCCGTTCACCCTCTCAGGCCGGCTATGGATCGTCGGCTTGGTAGGCCGTTACCCCACCAA
>SWDL27_1
TTGGGCCGTGTCTCAGTCCCAATGTGGCCGTTCACCCTCTCAGGCCGGCTACTGATCGTCGCTTTGGTGGGCCGTTACCCCGCCAA
>SWDL50_2
CTGGGCCGTGTCTCAGTCCCAATGTGGCCGTTCAACCTCTCAGTCCGGCTACCGATCGTCGCCTTGGTGGGCCATTACCTCACCAA
>SWDL15_2
CTGGACCGTGTCTCAGTTCCAGTGTGGCCGTTCATCCTCTCAGACCGGCTACTGATCGTAGGTTTGGTGGGCCGTTACCTCACCAA
>SWDL36_2
TTGGGCCGTGTCTCAGTCCCAATGTGGCCGGTCACCCTCTCAGGCCGGCTACTGATCGTCGCCTTGGTGGGCCGTTACCTCACCAA
```

step 0. understand your data

- single end reads? paired end? already merged?
  - if not merged, you need to do this step yourself
- one single fastq? one fastq file per sample?
  - if downloading data: already a fasta?
- what about the sequences?
  - barcodes removed?
  - primers removed?
  - length trimmed already?
- every sequencing center does it differently!!
  - use your eyes and your brain

step 0. paired-end sequencing will require merging

- merge forward and reverse reads
  - can use usearch or Scott’s script
  - note: barcodes may be in a separate index file
- AWS pipeline does not do this step



“4” “easy” “steps” for 16S data

0. figure out what data you have

   your eyes and your brain

1. pre-process data

   AWS pipeline

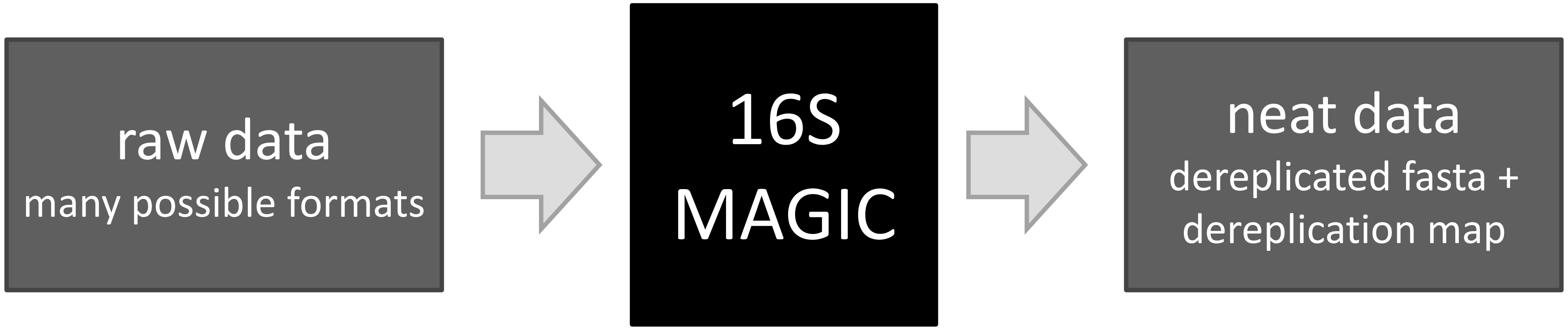
2. build OTU table

   AWS pipeline

3. analyze

   your eyes and your brain

step 1. pre-processing



Raw data options:

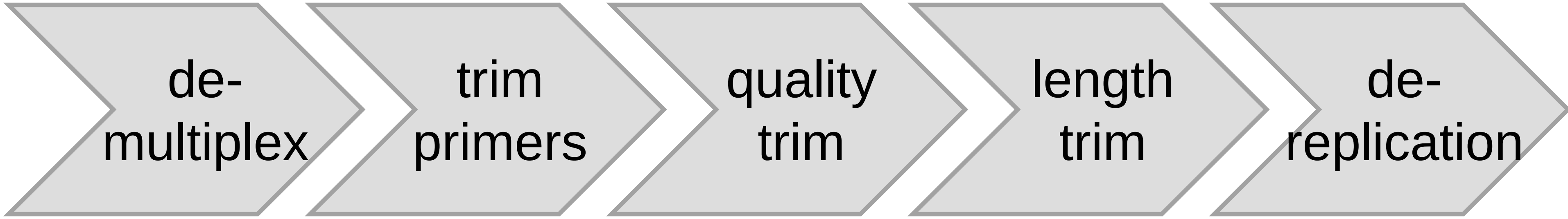
- one fastq or multiple fastq files
- with or without barcodes in sequences
- with or without primers in sequences
- all different lengths, possibly low quality
- can also take in a fasta

Outputs:

- dereplicated fasta: all of the unique sequences
- dereplication map: counts of each sequence per sample

step 1. pre-processing with AWS pipeline

- which steps are done depends on inputs
  - talk to pipeline via `summary_file.txt`
- AWS pipeline calls a medley of scripts
  - lab scripts (`2.split_by_barcodes.py`, etc)
  - usearch functions



step 1.1: need to split by barcodes if fastqs aren’t already demultiplexed

- if you only have one fastq file with all samples in it, need to split by barcodes
  - basically put sample IDs in the fastq sequence IDs
- sequencing centers often demultiplex fastqs for you
  - i.e. you have one file per sample

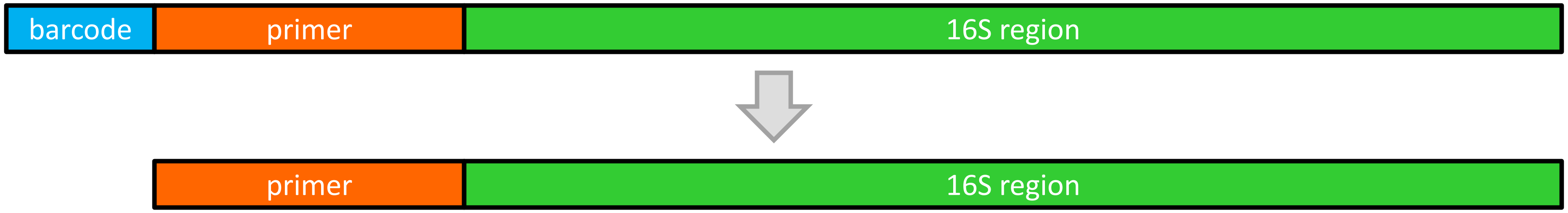
de-multiplex
trim primers
quality trim
length trim
de-replication

step 1.1: split by barcodes

- barcodes can be in multiple places
  - beginning of sequence, sequence ID, index file
  - this can be specified in `summary_file.txt`
- pipeline calls `2.split_by_barcodes.py`
  - or simply renames sequences with sample IDs if fastq is already de-multiplexed

de-multiplex
trim primers
quality trim
length trim
de-replication

step 1.1: demultiplexing also trims off barcode and relabels sequences

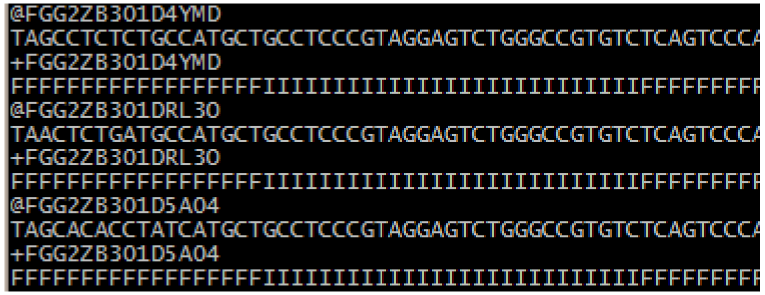


xaa

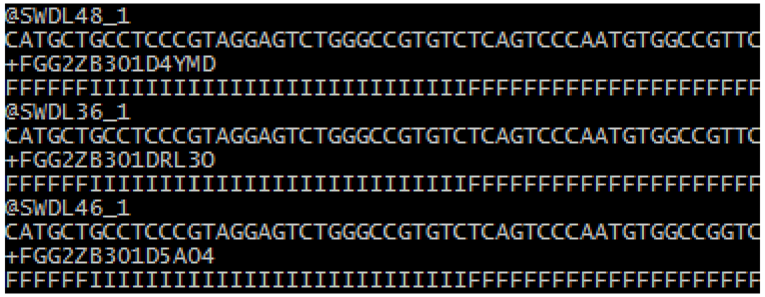
or original fastq file name

xaa.sb

sample IDs in seq IDs
barcodes removed
no seqs with barcode mismatches



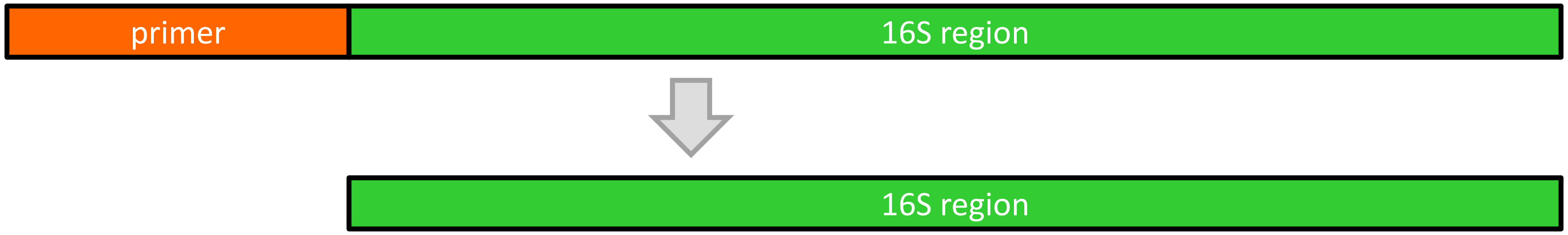
```
@FGG2ZB301D4YMD
TAGCCTCTCTGCCATGCTGCCTCCCGTAGGAGTCTGGGCCGTGTCTCAGTCCCA
+FGG2ZB301D4YMD
FFFFFFFFFFFFFFFFFFIIIIIIIIIIIIIIIIIIIIIIIIIIIFFFFFFFFF
@FGG2ZB301DRL30
TAACTCTGATGCCATGCTGCCTCCCGTAGGAGTCTGGGCCGTGTCTCAGTCCCA
+FGG2ZB301DRL30
FFFFFFFFFFFFFFFFFFIIIIIIIIIIIIIIIIIIIIIIIIIIIFFFFFFFFF
@FGG2ZB301D5A04
TAGCACACCTATCATGCTGCCTCCCGTAGGAGTCTGGGCCGTGTCTCAGTCCCA
+FGG2ZB301D5A04
FFFFFFFFFFFFFFFFFFIIIIIIIIIIIIIIIIIIIIIIIIIIIFFFFFFFFF
```



```
@SWDL48_1
CATGCTGCCTCCCGTAGGAGTCTGGGCCGTGTCTCAGTCCCAATGTGGCCGTTC
+FGG2ZB301D4YMD
FFFFFFIIIIIIIIIIIIIIIIIIIIIIIIIIIFFFFFFFFFFFFFFFFFFFFF
@SWDL36_1
CATGCTGCCTCCCGTAGGAGTCTGGGCCGTGTCTCAGTCCCAATGTGGCCGTTC
+FGG2ZB301DRL30
FFFFFFIIIIIIIIIIIIIIIIIIIIIIIIIIIFFFFFFFFFFFFFFFFFFFFF
@SWDL46_1
CATGCTGCCTCCCGTAGGAGTCTGGGCCGTGTCTCAGTCCCAATGTGGCCGGTC
+FGG2ZB301D5A04
FFFFFFIIIIIIIIIIIIIIIIIIIIIIIIIIIFFFFFFFFFFFFFFFFFFFFF
```

step 1.2: remove primers if they're still in sequences

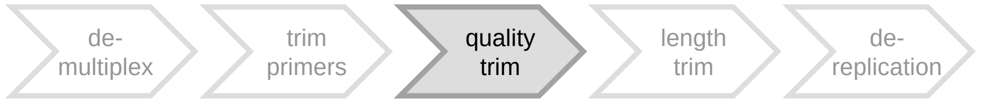
- should manually check if primers are in seqs
- pipeline calls wrapper to `1.remove_primers.py`
  - checks for primers in forward and reverse complement sequence



de-multiplex
trim primers
quality trim
length trim
de-replication

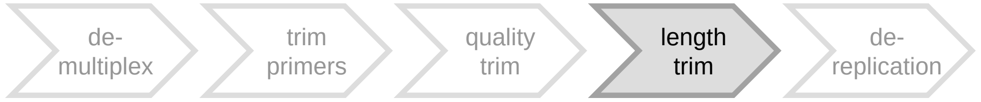
step 1.3: quality trimming

- you need to figure out quality score encoding
  - use usearch –fastq_chars to make a guess
  - specify in ASCII_ENCODING in summary file
- AWS quality trims with usearch –fastq_filter
  - truncates at first base with p(error) > 0.003
  - default Q=25, can specify in summary file
- what kind of quality filtering do you prefer?



step 1.4: length trimming

- sequences must be same length for later steps
  - i.e. alignment and clustering
- AWS length trims with usearch –fastx_truncate
  - AWS default is 101, specify in TRIM_LENGTH
- results are in *.raw_trimmed.fasta
  - has all of the reads for all samples
  - all same length, no barcodes or primers
  - sample IDs are in sequence headers
- we're almost there!



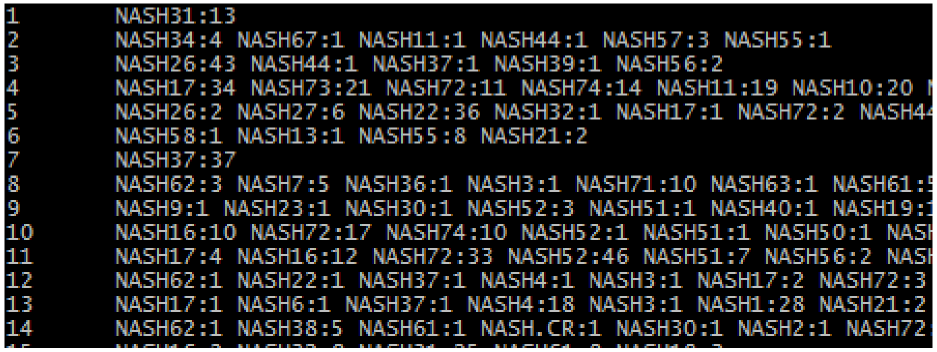
step 1.5: dereplication

- keeps only the unique sequences
  - in AWS these are in *.raw_dereplicated.fasta
- and does “provenancing”
  - i.e. maps how many times each sequence appeared in each sample
  - in AWS this file is *.dereplication_map
- pipeline calls `3.dereplication.py`

de-multiplex
trim primers
quality trim
length trim
de-replication

step 1.5: dereplication documentation is confusing but outputs are recognizable

dereplication map



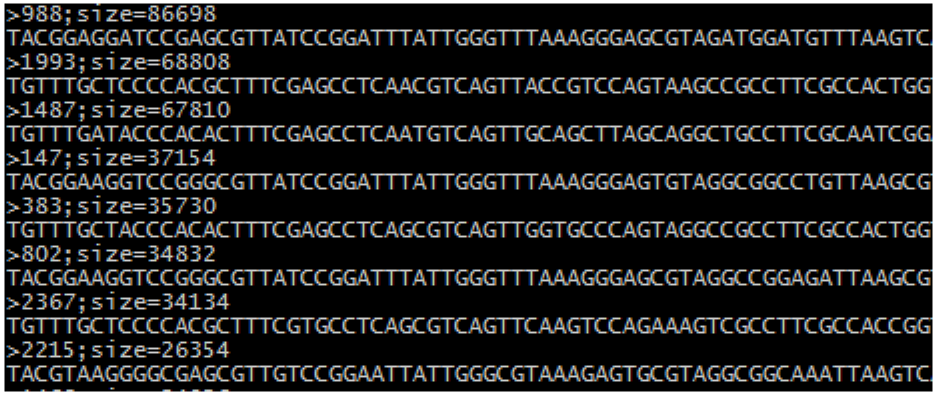
```
1       NASH31:13
2       NASH34:4 NASH67:1 NASH11:1 NASH44:1 NASH57:3 NASH55:1
3       NASH26:43 NASH44:1 NASH37:1 NASH39:1 NASH56:2
4       NASH17:34 NASH73:21 NASH72:11 NASH74:14 NASH11:19 NASH10:20 N
5       NASH26:2 NASH27:6 NASH22:36 NASH32:1 NASH17:1 NASH72:2 NASH44
6       NASH58:1 NASH13:1 NASH55:8 NASH21:2
7       NASH37:37
8       NASH62:3 NASH7:5 NASH36:1 NASH3:1 NASH71:10 NASH63:1 NASH61:5
9       NASH9:1 NASH23:1 NASH30:1 NASH52:3 NASH51:1 NASH40:1 NASH19:1
10      NASH16:10 NASH72:17 NASH74:10 NASH52:1 NASH51:1 NASH50:1 NASH
11      NASH17:4 NASH16:12 NASH72:33 NASH52:46 NASH51:7 NASH56:2 NASH
12      NASH62:1 NASH22:1 NASH37:1 NASH4:1 NASH3:1 NASH17:2 NASH72:3
13      NASH17:1 NASH6:1 NASH37:1 NASH4:18 NASH3:1 NASH1:28 NASH21:2
14      NASH62:1 NASH38:5 NASH61:1 NASH.CR:1 NASH30:1 NASH2:1 NASH72:
```

sequence 1 occurred 13 times in sample NASH31

sequence 2 occurred 4 times in sample NASH34, 1 time in NASH67, 1 time in NASH11, …

sequence 3 occurred 43 times in NASH26, 1 time in NASH37, 1 time in NASH39, and 2 times in NASH6

dereplicated fasta

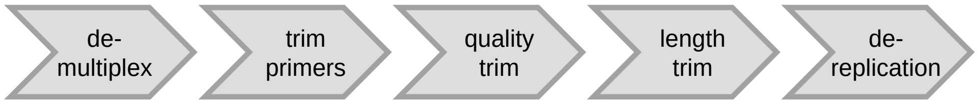


```
>988;size=86698
TACGGAGGATCCGAGCGTTATCCGGATTTATTGGGTTTAAAGGGAGCGTAGATGGATGTTTAAGTC
>1993;size=68808
TGTTTGCTCCCCACGCTTTCGAGCCTCAACGTCAGTTACCGTCCAGTAAGCCGCCTTCGCCACTGG
>1487;size=67810
TGTTTGATACCCACACTTTCGAGCCTCAATGTCAGTTGCAGCTTAGCAGGCTGCCTTCGCAATCGG
>147;size=37154
TACGGAAGGTCCGGGCGTTATCCGGATTTATTGGGTTTAAAGGGAGTGTAGGCGGCCTGTTAAGCG
>383;size=35730
TGTTTGCTACCCACACTTTCGAGCCTCAGCGTCAGTTGGTGCCCAGTAGGCCGCCTTCGCCACTGG
>802;size=34832
TACGGAAGGTCCGGGCGTTATCCGGATTTATTGGGTTTAAAGGGAGCGTAGGCCGGAGATTAAGCG
>2367;size=34134
TGTTTGCTCCCCACGCTTTCGTGCCTCAGCGTCAGTTCAAGTCCAGAAAGTCGCCTTCGCCACCGG
>2215;size=26354
TACGTAAGGGGCGAGCGTTGTCCGGAATTATTGGGCGTAAAGAGTGCGTAGGCGGCAAATTAAGTC
```

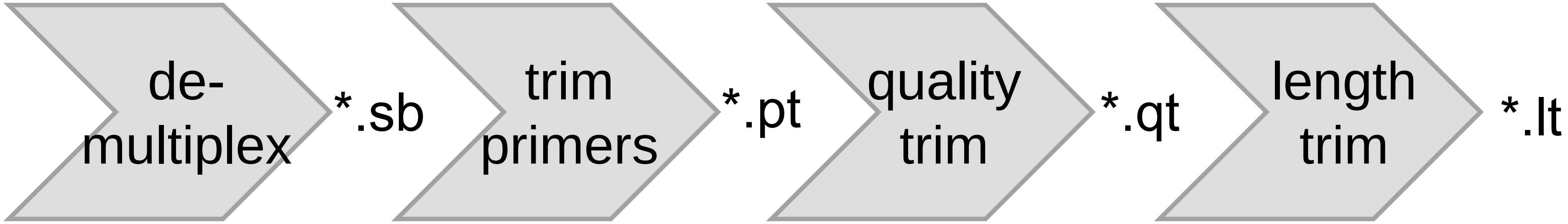
unique sequences only
sequences sorted by counts

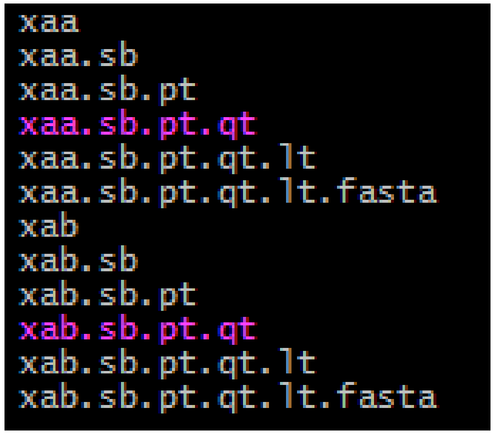
step 1: dereplicated fasta and map are the end goal of pre-processing

- pipeline handles many possible inputs
  - one fastq or multiple fastq files
  - with or without barcodes in sequences
  - with or without primers in sequences
  - all different lengths, possibly low quality
  - can also take in a fasta
- all of this gets specified in `summary_file.txt`
  - if there are no primers/barcodes, put None
  - for other options, read documentation



step 1: AWS pipeline saves intermediate files, identified with suffixes





```
xaa
xaa.sb
xaa.sb.pt
xaa.sb.pt.qt
xaa.sb.pt.qt.lt
xaa.sb.pt.qt.lt.fasta
xab
xab.sb
xab.sb.pt
xab.sb.pt.qt
xab.sb.pt.qt.lt
xab.sb.pt.qt.lt.fasta
```

so you can use your eyes and your brain

processing: eyes and brain practice

- datasets on Alm lab node
- what data do we have?
  - fastq or fasta?
  - de-multiplexed already?
  - primers in there?
  - need to merge?
  - trimmed yet?

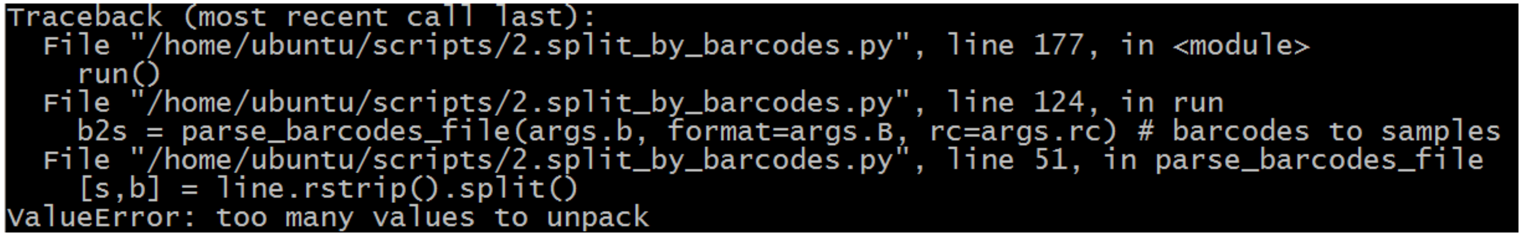
processing: eyes and brain practice

- datasets on workshop node
  - ssh -i 16S_workshop.pem ubuntu@52.70..179.175
- for datasets 7-10: what happened?
  - where would you start de-bugging?
  - where can you find more info about what went wrong?

where did it fail? what went wrong?

dataset7

dataset8



```
Traceback (most recent call last):
  File "/home/ubuntu/scripts/2.split_by_barcodes.py", line 177, in <module>
    run()
  File "/home/ubuntu/scripts/2.split_by_barcodes.py", line 124, in run
    b2s = parse_barcodes_file(args.b, format=args.B, rc=args.rc) # barcodes to samples
  File "/home/ubuntu/scripts/2.split_by_barcodes.py", line 51, in parse_barcodes_file
    [s,b] = line.rstrip().split()
ValueError: too many values to unpack
```

dataset9

expected 128 samples, got 56
have intermediate files

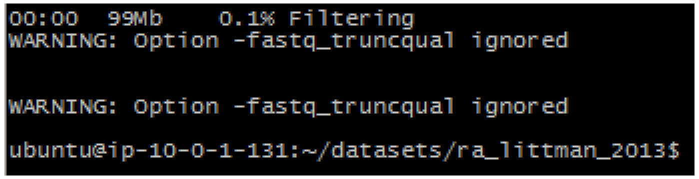
dataset10

USEARCH broke
have intermediate files

what went wrong? (hints)

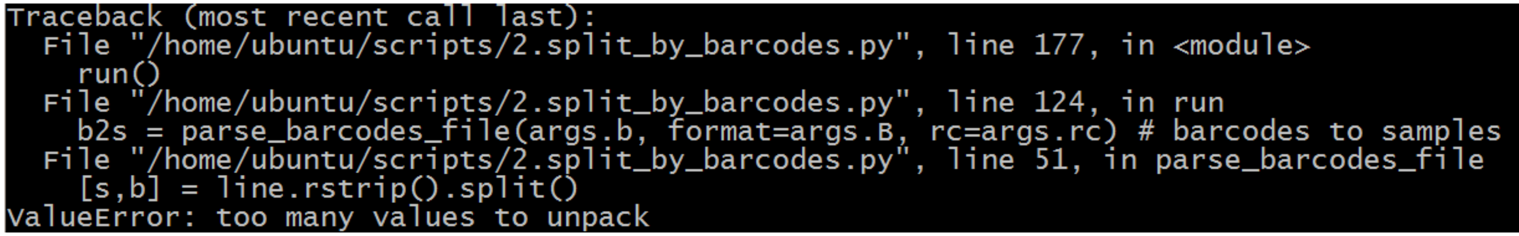
dataset7

no useful errors sry



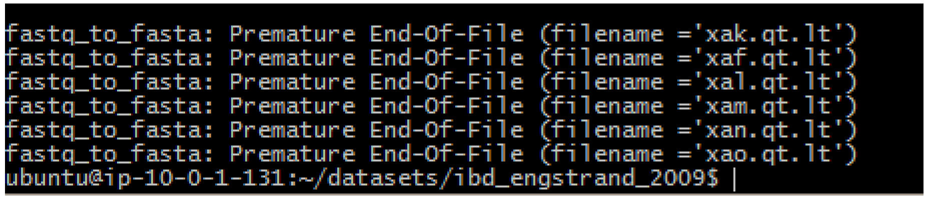
```
00:00   99Mb     0.1% Filtering
WARNING: Option -fastq_truncqual ignored


WARNING: Option -fastq_truncqual ignored

ubuntu@ip-10-0-1-131:~/datasets/ra_littman_2013$
```

dataset8



```
Traceback (most recent call last):
  File "/home/ubuntu/scripts/2.split_by_barcodes.py", line 177, in <module>
    run()
  File "/home/ubuntu/scripts/2.split_by_barcodes.py", line 124, in run
    b2s = parse_barcodes_file(args.b, format=args.B, rc=args.rc) # barcodes to samples
  File "/home/ubuntu/scripts/2.split_by_barcodes.py", line 51, in parse_barcodes_file
    [s,b] = line.rstrip().split()
ValueError: too many values to unpack
```

dataset9



```
fastq_to_fasta: Premature End-Of-File (filename ='xak.qt.lt')
fastq_to_fasta: Premature End-Of-File (filename ='xaf.qt.lt')
fastq_to_fasta: Premature End-Of-File (filename ='xal.qt.lt')
fastq_to_fasta: Premature End-Of-File (filename ='xam.qt.lt')
fastq_to_fasta: Premature End-Of-File (filename ='xan.qt.lt')
fastq_to_fasta: Premature End-Of-File (filename ='xao.qt.lt')
ubuntu@ip-10-0-1-131:~/datasets/ibd_engstrand_2009$ |
```

dataset10

“4” “easy” “steps” for 16S data

0. figure out what data you have

    your eyes and your brain

1. pre-process data

    AWS pipeline

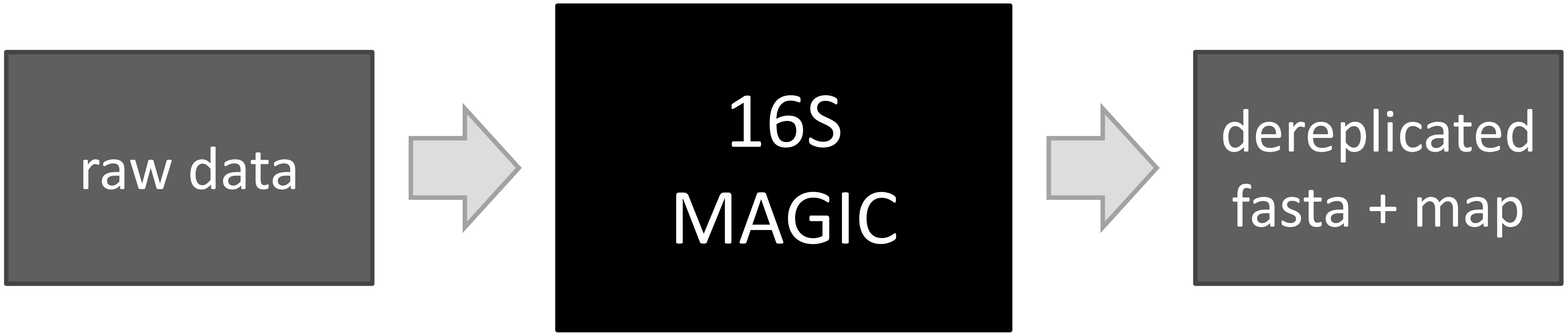
2. build OTU table

    AWS pipeline

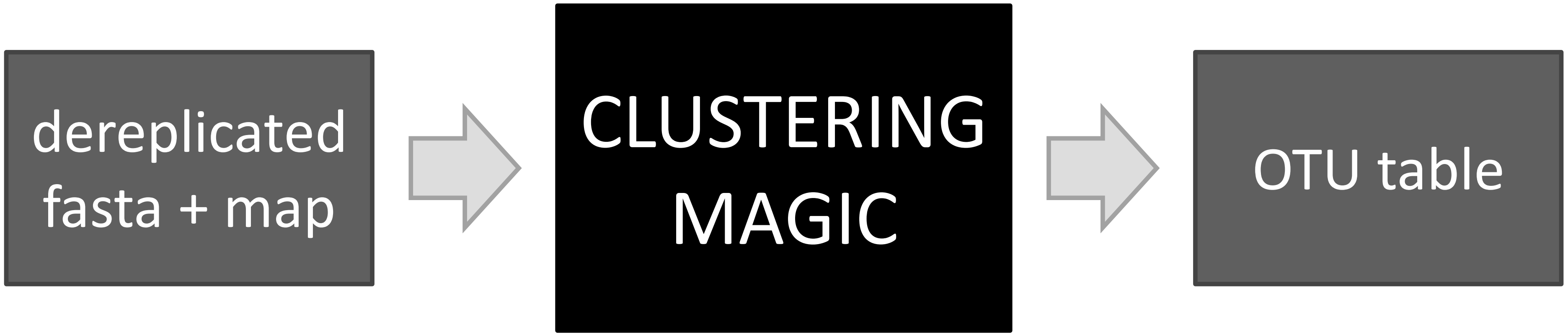
3. analyze

    your eyes and your brain

step 1. pre-processing



step 2. build OTU table



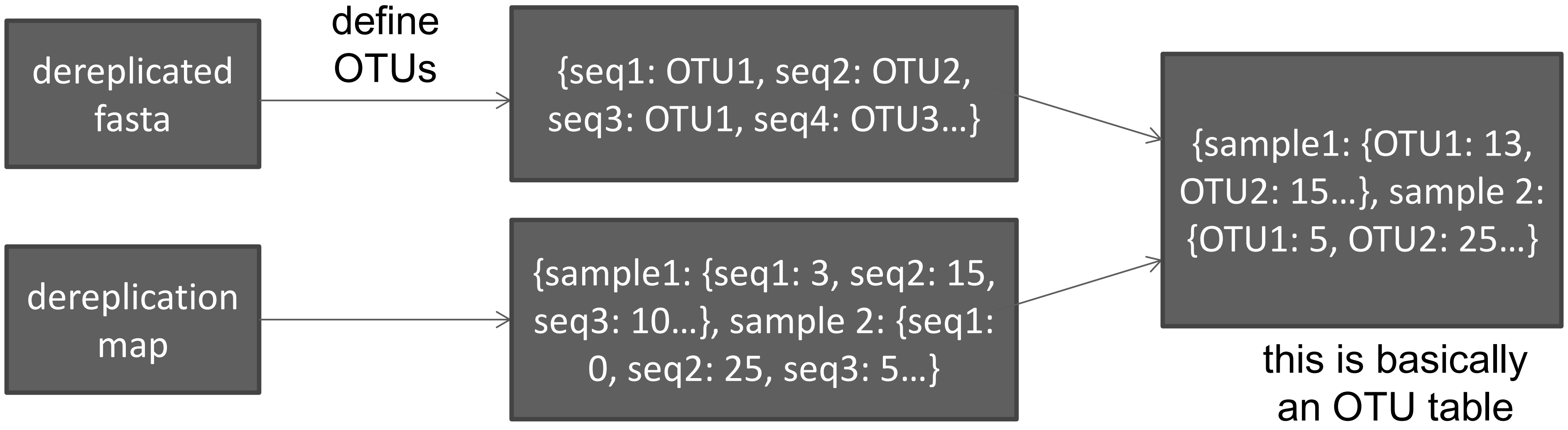
step 2: building an OTU table is essentially combining dictionaries

- an OTU table is a nested dictionary
  - it shows the counts for each OTU in each sample
  - {sample1: {OTU1: counts, OTU2: counts}…}
- so we first need to map:
  - sequence ID → OTU ID for each original seq
    sequence ID → counts for each sample

step 2: building an OTU table is essentially combining dictionaries

dereplicated fasta is used to map sequences to OTUs

dereplication map already contains the map of sequence counts to samples


dereplicated fasta
define OTUs
{seq1: OTU1, seq2: OTU2, seq3: OTU1, seq4: OTU3…}
dereplication map
{sample1: {seq1: 3, seq2: 15, seq3: 10…}, sample 2: {seq1: 0, seq2: 25, seq3: 5…}
{sample1: {OTU1: 13, OTU2: 15…}, sample 2: {OTU1: 5, OTU2: 25…}
this is basically an OTU table

step 2: there are 3 categories of methods for defining OTUs

de novo

make OTUs based only on sequences (ATCG)
no database

closed reference

map sequences to a database to define OTUs
throw out anything that doesn’t match
most commonly: map to Green Genes

open reference

map sequences to a database
then do de novo clustering on leftover sequences
and add to the mapped sequences

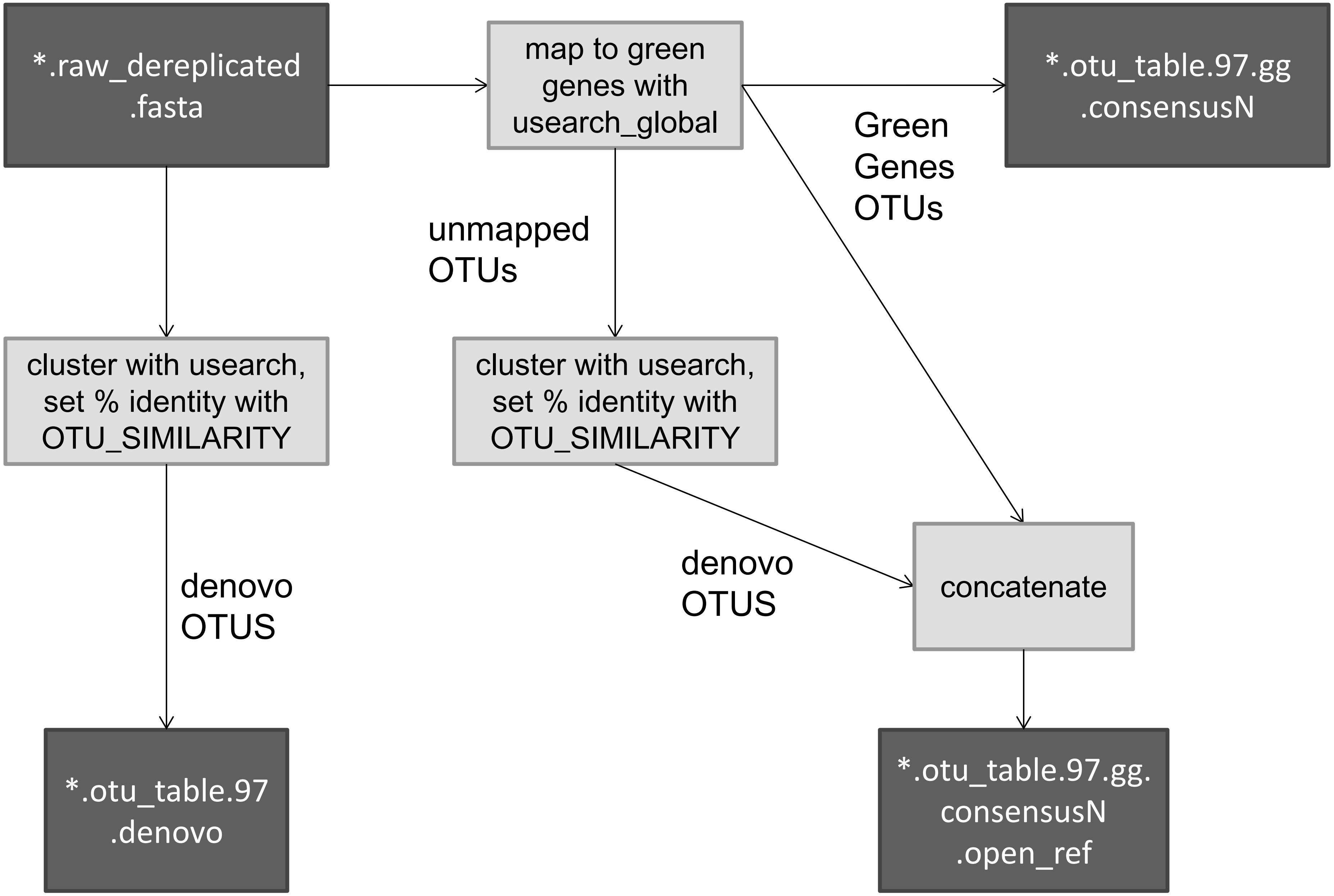
step 2: denovo OTU calling approaches

- similarity based OTU clustering
  - simplest and most common
  - usearch –cluster_otus
  - also removes chimeras
- oligotyping
  - uses information theory to separate meaningful sequences
- distribution-based clustering (DBC)
  - uses both similarity and distribution of sequences to call OTUs
- DADA2
  - seems awesome, only works for Illumina data
- if not using usearch, you need to remove chimeras!
  - at some point in the processing pipeline

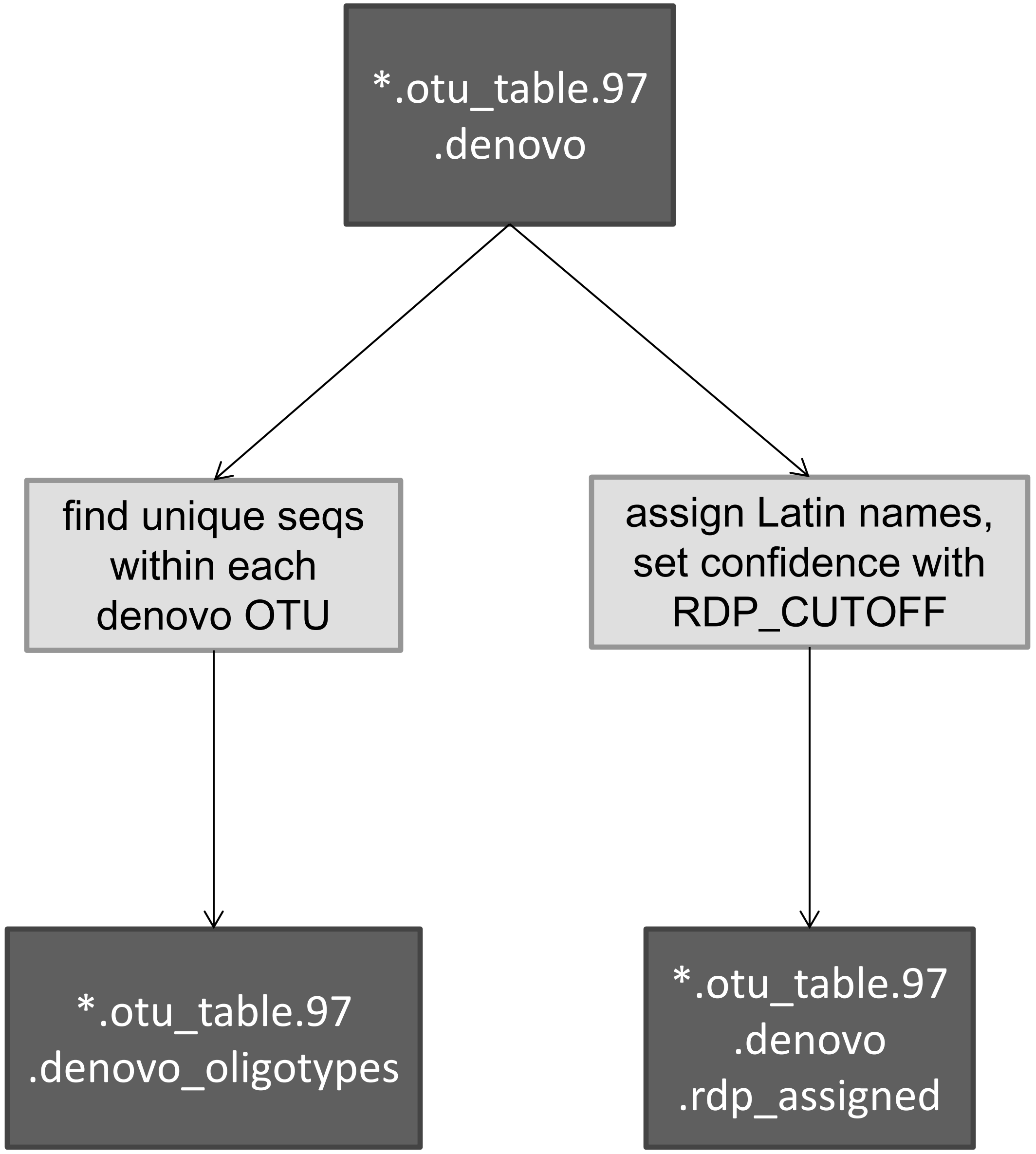
step 2: all OTU calling approaches have pros and cons

- denovo
  - pro: no need for database
  - con: need to do more work for taxonomic info
  - con: can’t compare across datasets
- closed reference
  - pro: get taxonomic info for cheap
  - pro: might be able to compare across datasets
  - con: database-dependent
  - con: how clustering method deals with ties is important
  - con: throws out all unmatched sequences
- open reference
  - pro: you don’t throw out your unknown sequences
  - pro: have taxonomic info for some OTUs
  - cons: same disadvantages as denovo and closed
  - cons: is the distance metric really the same?

step 2: AWS pipeline spits out heaps of files


*.raw_dereplicated .fasta
map to green genes with usearch_global
*.otu_table.97.gg .consensusN
Green Genes OTUs
unmapped OTUs
cluster with usearch, set % identity with OTU_SIMILARITY
cluster with usearch, set % identity with OTU_SIMILARITY
denovo OTUS
denovo OTUS
concatenate
*.otu_table.97 .denovo
*.otu_table.97.gg. consensusN .open_ref

step 2: but wait there's more!


*.otu_table.97
.denovo
find unique seqs within each denovo OTU
assign Latin names, set confidence with RDP_CUTOFF
*.otu_table.97
.denovo_oligotypes
*.otu_table.97
.denovo
.rdp_assigned

*.otu_seqs.97.fasta

contains representative sequences for the denovo OTUs

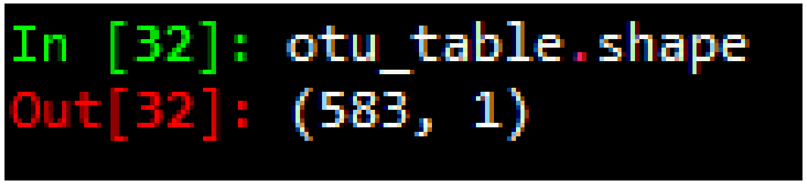
*.otu_table.97.gg
.consensusN

GreenGenes assignment obtained by taking the consensus of the top N hits when aligned to the GreenGenes database

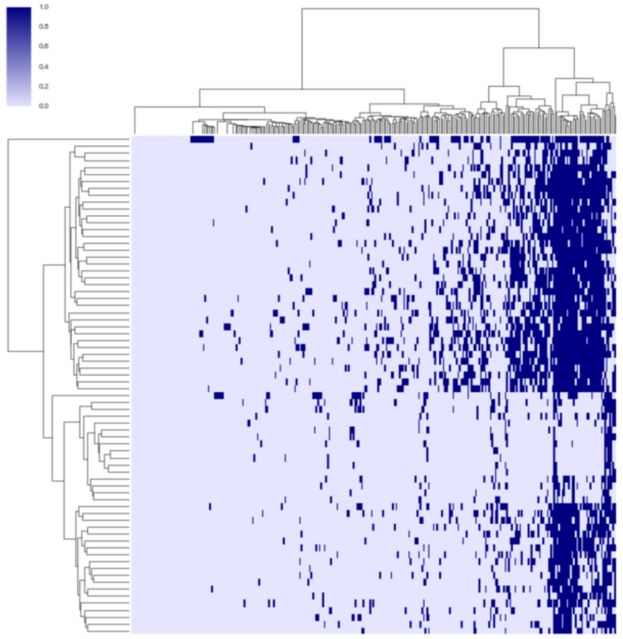
step 3: analyze data – sanity checks

sanity checks

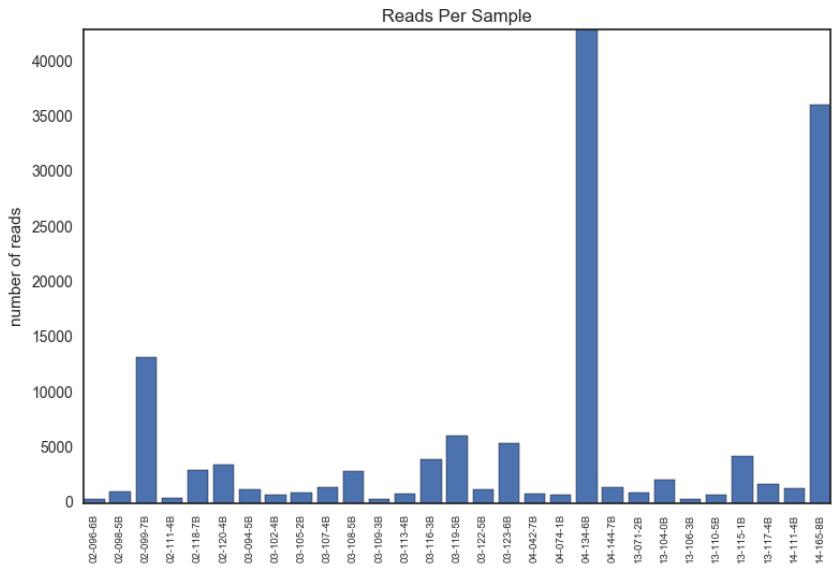
- what's in your OTU table
- number of reads per sample
- distribution of OTUs across samples
- heatmap of abundances

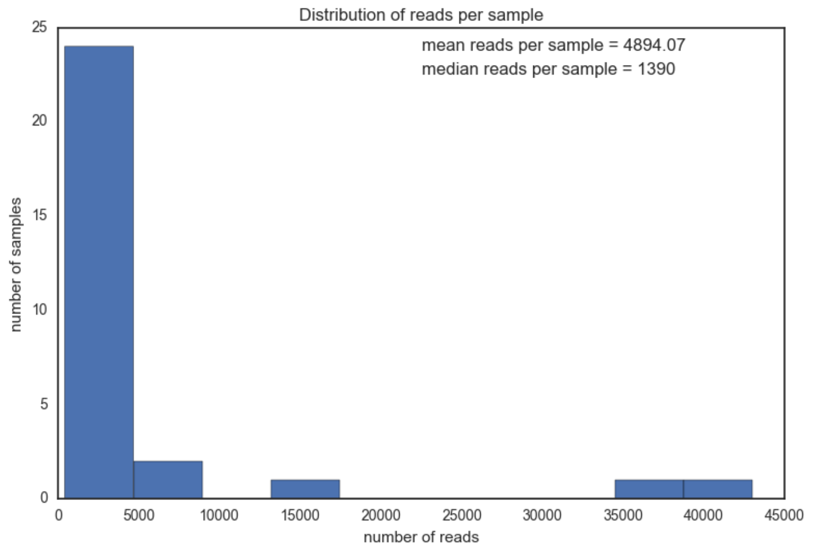


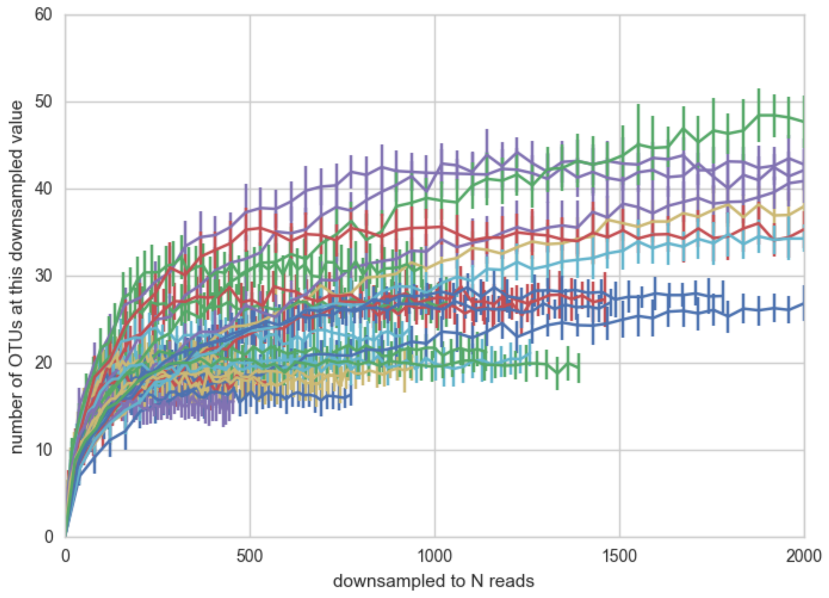
```
In [32]: otu_table.shape
Out[32]: (583, 1)
```



step 3: low reads per sample can be bad but can also be okay







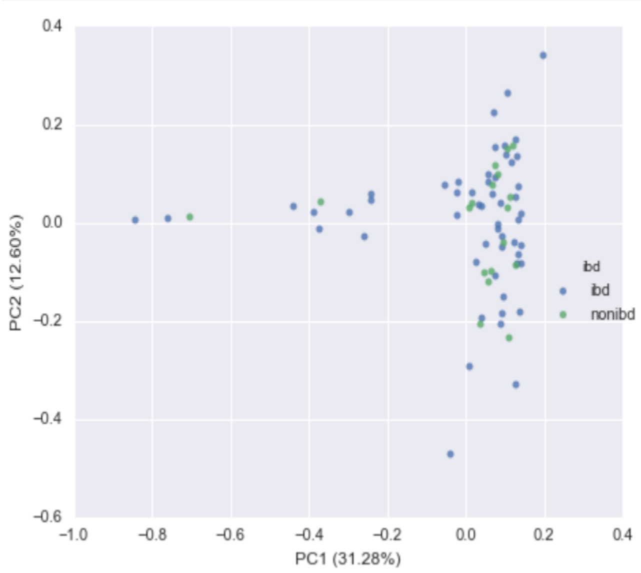
step 3: some places to start with analysis

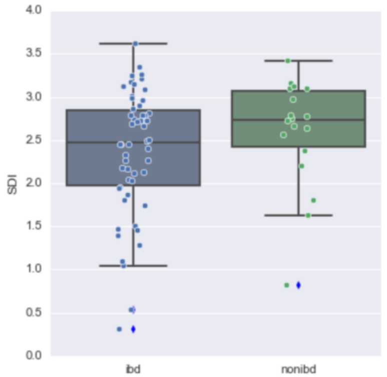
alpha diversity

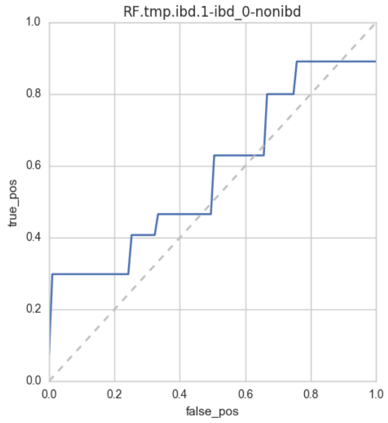
PCA

univariate comparisons

basic classifiers







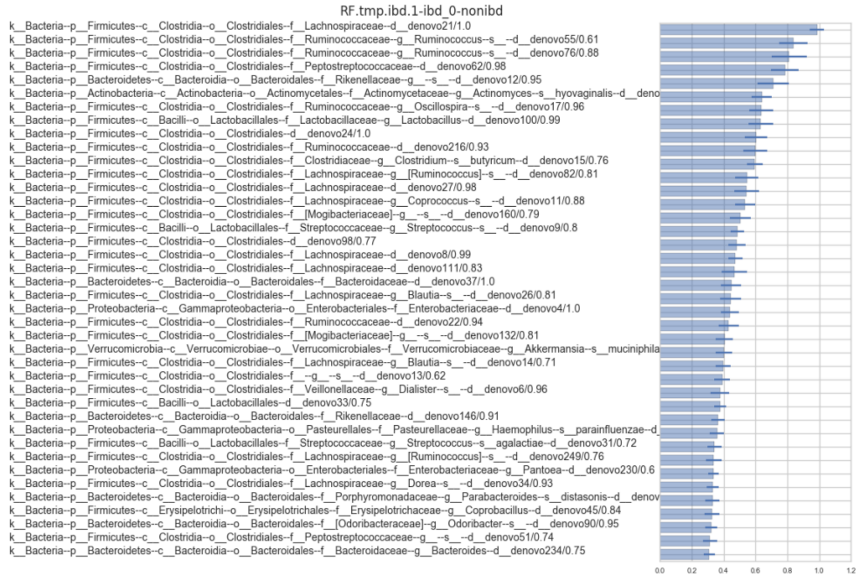
Class 0 = 18 samples

Class 1 = 54 samples

AUC = 0.580

conf mat p-value = 1.000

```
   Class 0 Class 1
Class 0 0      18
Class 1 0      54
```



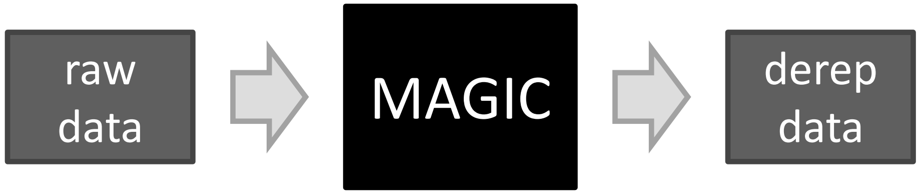
“4” “easy” “steps” for 16S data

0. figure out what data you have

   your eyes and your brain

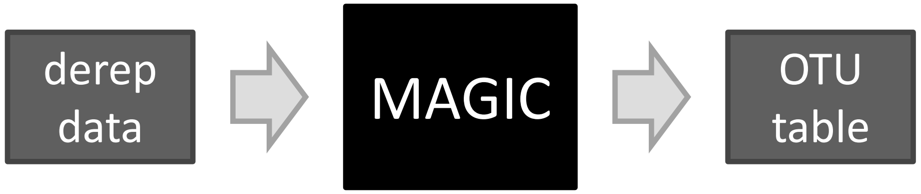
1. pre-process data

   AWS pipeline



2. build OTU table

   AWS pipeline



3. analyze

   your eyes and your brain

using the pipeline is easy and looking inside the black box is also easy

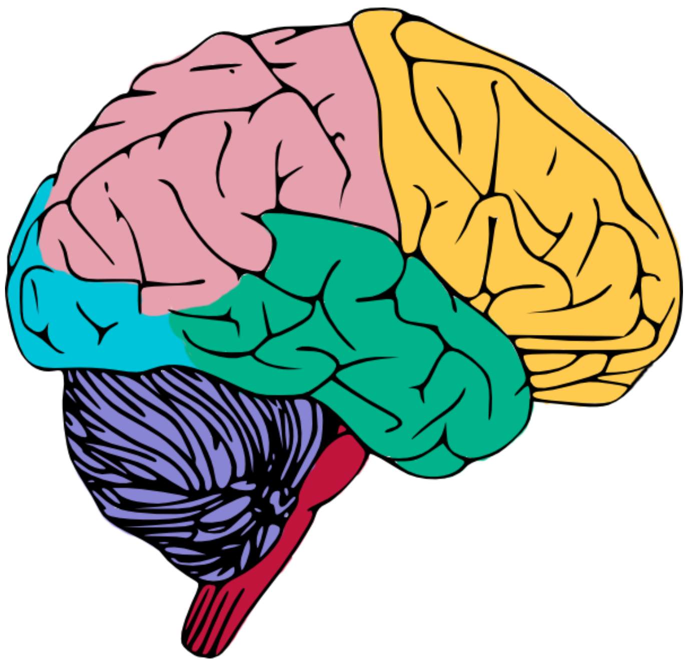
- code currently on AWS and private github
- Master.py calls raw2otu.py
  - reads in `summary_file.txt` for file locations, parameter settings, etc
- raw2otu.py does most of its work in the
- preprocessing_16S.py module
  - many functions are wrappers to either usearch functions or Alm lab scripts

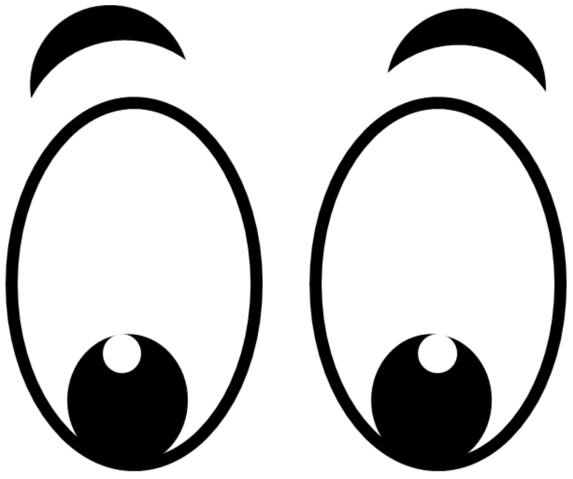
there’s a dropbox folder from 20.106 with problem sets and sample data

- play around with the files
    - look at what input and output files look like
    - figure out the syntax
- don’t be limited by the code given
    - some steps are missing in instructions (i.e. length and quality trim)
    - you can also play around with parameters

https://www.dropbox.com/sh/enyftm9ut2r74ry/AAAZejnQlhUydVNsrCnWgrr-a?dl=0

thanks!





$$\begin{tikzcd}
\boxed{\text{raw data}} \arrow[r] & \boxed{\text{MAGIC}} \arrow[r] & \boxed{\text{derep data}} \\
\boxed{\text{derep data}} \arrow[r] & \boxed{\text{MAGIC}} \arrow[r] & \boxed{\text{OTU table}}
\end{tikzcd}$$