Introduction

This notebook is a companion to the paper published in Nature by Tao Ding & Pat Schloss entitled "Dynamics and associations of microbial community types across the human body" (http://dx.doi.org/10.1038/nature13178). Dr. Ding was a postdoc in the lab of Dr Schloss within the Department of Microbiology & Immunology at the University of Michigan. If any questions should arise, please feel free to contact Dr. Schloss at pschloss@umich.edu. This notebook was originally developed using mothur ~v.1.30. Since then we have tweaked the running of trim.flows which may have a small effect on the number of samples with 1000 reads. Since then all commands have been re-validated using mothur v.1.33.3. You can find all of the raw data and derivative files generated below (except for the dbGaP data) in the dingschlossnature bucket (http://dingschlossnature.s3-website-us-west-2.amazonaws.com) on the AWS S3 server

The HMP Clinical Sequencing Project

The underlying data were developed as part of the Human Microbiome Project to understand the structure of the microbiome in healthy individuals Peterson et al. 2009 (http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2792171/). The folks at PLoS have been very helpful in collecting the papers published as part of the overall HMP clinical study as well as the other projects associated with HMP funding as the The Human Microbiome Project Collection (http://www.ploscollections.org/article/browseIssue.action?issue=info:doi/10.1371/issue.pcol.v01.i13) (nb: there are other papers from the HMP that don't appear in this collection). As mentioned in our paper, the individuals in the main clinical study were **very** healthy. A talmudic question for you to consider is whether heatlhy == normal. We contend that if 80% of American's have a cavity in their mouth, they are *normal*, but not healthy. In this study, people with cavities were excluded from the entire study. You can learn more about the exclusion criteria at the dbGaP project site (http://www.ncbi.nlm.nih.gov/projects/gap/cgi-bin/study.cgi?study_id=phs000228.v3.p1) and Aagaard et al. ,2013 (http://www.ncbi.nlm.nih.gov/pubmed/23165986).

Overall, the study sampled 15 bodysites from 150 women and 18 bodysites from 150 men. Of the 300 subjects, 150 were enrolled in Houston, TX and 150 were enrolled in St. Louis, MO. The goal was to sample all subjects twice and 100 subjects a third time. All samples were to be characterized by 16S rRNA gene sequencing and a decent number were sequenced via metagenomics shotgun sequencing. The current study only concerned itself with the 16S rRNA gene sequencing. There were several stages to the study. The first consisted of a "Pilot Production Study" where a set of 24 subjects (12 men and 12 women) were recruited in Houston. The samples were processed there and then split with one replicate sent to one sequencing center and another replicate sent to another sequencing center. There were four sequencing centers in total: The Baylor College of Medicine (BCM), The Washington University Genome Sequencing Center (WUGSC), The Broad Institute (BI), and the J Craig Venter Institute (JCVI). These samples have already been described in earlier studies by us and others (Schloss et al. 2011 (http://www.ploscollections.org/article/info%3Adoi%2F10.1371%2Fjournal.pone.0027310), JCHMPDGWG 2012 (http://www.ploscollections.org/article/info%3Adoi%2F10.1371%2Fjournal.pone.0039315)).

Next, the project scaled up to the full 300 subject pool. As we discuss in the Supplementary Information for the paper, the city of origin for each subject was a complicating varaible. The samples from the 150 subjects from St. Louis were extracted and sequenced at WUGSC. The samples from the 150 subjects in Houston were sent to BCM, JCVI, and BI for extraction and sequencing. Thus the city of origin was perfectly confounded with the location of sequencing. As was discussed elsewhere, the largest effect observed in the data was from city of origin (HMPC 2012 (http://www.nature.com/nature/journal/v486/n7402/full/nature11234.html). Although we would love to believe that people vary in their microbiome between Houston and St. Louis and can concoct many reasons why this would be so, the easiest is that there were small differences in how samples were obtained, handled, and processed that affected the microbiome structure. Regardless, prior to our paper, the only data that had been published was from the so-called "May 1, 2010" data freeze. The samples from this data freeze primarily only contained the first sampling point for each subject. Although the second sample was available for some of these subjects by that freeze, the papers published to date (e.g. HMPC 2012 (http://www.nature.com/nature/journal/v486/n7402/full/nature11234.html), HMPC 2012 (http://www.nature.com/nature/journal/v486/n7402/full/nature11209.html), and others (http://www.ploscollections.org/article/browseIssue.action?issue=info:doi/10.1371/issue.pcol.v01.i13)). Thus, the goal of our paper was to present an analysis of all of the data generated by the HMP clinical study.

Data curation

The data

Sequence data. The 16S rRNA gene sequencing data was generated by sequencing the V35 region of gene on the 454 Titanium platform. Initially the project sequenced the V13 and V69 regions; however, these were eventually dropped from the analysis plan. Thus, we focused our effort on the V35 region. All of the 16S rRNA gene sequence data are available at the NCBI Short Read Archive (SRA): the Clinical Pilot Project (accession SRP002012 (http://www.ncbi.nlm.nih.gov/Traces/sra/?study=SRP002012)), Phase I (the May 1 freeze; accession SRP002395 (http://www.ncbi.nlm.nih.gov/Traces/sra/?study=SRP002395)), and Phase II (accession SRP002860 (http://www.ncbi.nlm.nih.gov/Traces/sra/?study=SRP002860)). Because we were part of the original Data Analysis Working Group (DAWG), we originally obtained some of the data from Jonathan Crabtree at the HMP Data Analsyis Coordination Center (DACC) (http://www.hmpdacc.org). Thus, there may be slight variations in the data that we used and the data available throgh the SRA.

The 16S rRNA gene sequence analysis pipeline used in this study is based on the results of our previous work analyzing the mock community data generated as part of the PPS (Schloss et al. 2011 (http://www.ploscollections.org/article/info%3Adoi%2F10.1371%2Fjournal.pone.0027310)). We used the PyroNoise-based approach with *de novo* chimera checking using the UCHIME algorithm (Quince et al. 2011 (http://www.biomedcentral.com/1471-2105/12/38) and Edgar et al. 2011 (http://bioinformatics.oxfordjournals.org/content/27/16/2194.long)). All analysis steps were performed within mothur (Schloss et al. 2009 (http://aem.asm.org/content/75/23/7537.long)). As we described there, this curation pipeline yields an error rate of ~0.02%. The mothur package can be obtained from the mothur website (http://www.mothur.org) or our GitHub repository (https://github.com/mothur/mothur). The wiki also has a shorter description of our SOP for processing 454 sequencing data (http://www.mothur.org/wiki/454_SOP).

Clinical data. The clinical data for this project is available through the dbGaP project site (http://www.ncbi.nlm.nih.gov/projects/gap/cgi-bin/study.cgi?study_id=phs000228.v3.p1). Unfortunately, because this is considered PHI, I am not at liberty to distribute these data. You may obtain the data from dbGaP by going through the appropriate channels. *Don't ask us to send the files to you!*

Sequence curation

Preliminaries

To organize the files we created several files in our directory:

```
In [ ]:
```

```
%%bash
mkdir scripts oligos_files references sff_files sffinfo trim.flows shhh.flows t
rim.seqs v13 v35 v69 dbGaP
```

You can obtain the oligos_file files from the GitHub repository for this project. If you are trying to run this, you will need to go in to the file and decompress the files (see oligos_files.tgz):

```
In [ ]:
```

```
%%bash
cd oligos_files
tar xvzf oligos_files.tgz
cd ../
```

These oligos files will tell mothur the barcode sequence that corresponds to each sample so that we can map reads to samples. If you open one of these files you'll see something that looks like this:

In [ ]:

```
forward ATTACCGCGGCTGCTGG       v13
forward CCGTCAATTCMTTTRAGT      v35
forward TACGGYTACCTTGTTAYGACTT  v69
barcode ACGAGAAC        phase1.SRR040576.BI.700034678
```

The first three lines are the "forward" PCR primer that were adjacent to the sequencing primer. By putting all three possible primers in the file we let mothur figure out which region the seuqences belong to. The barcode line gives the barcode sequence and then the sample name. The "phase 1" indicates that the sample was from before the May 1 data freeze. The "SRR040576" indicates the sequencing run that the sample came from. The "BI" indicates that the sample was sequenced at the Broad Institute". Finally, the "700034678" corresponds to the anonymized sample id. Together this string represents a unique identifier for every sequencing collection (some samples were sequenced twice, some at two centers, and some in multiple phases of the study).

You will also want to download the sequence data files from the SRA and convert them into sff format using the SRA's tools. Put the resulting sff files into the sff_files folder and return to the parent directory. Because we obtained several of the PPS datasets before they were publically available, the names of our files are slightly different from those on the SRA. At this point you'll need to double check that you have an oligos file for every sff file. Runing the following two commands should give the same number. The third command will tell you what you're missing. If you're using the same files that we used, there will be 3 files extra: F5672XE02, F5672XE02.v13, and F5672XE02.v35. There was somethign screwy about how these were sequenced and so later we will replace the F5672XE02 with the v13 and v35 versions and all will be well with the world.

In [ ]:

```
%%bash
ls sff_files | wc -l
ls oligos_files | wc -l
ls oligos_files/* sff_files/* | cut -f 2 -d "/" | cut -f 1 -d "." | sort | uniq
 -u
```

Next, we want to get the reference files that we'll be using to process the data. We chose to use the SILVA-based reference alignment (Pruesse et al. 2007 (http://nar.oxfordjournals.org/lookup/pmid?view=long&pmid=17947321)) because of its superiority to the greengenes-based alignment. Previous work from our lab has demonstrated this (Schloss 2010 (http://www.ploscompbiol.org/article/info%3Adoi%2F10.1371%2Fjournal.pcbi.1000844)). In addition, we used a modified version of the RDP's training set (Wang et al. 2007 (http://aem.asm.org/content/73/16/5261.long); v.9). Our modifications included adding several sequences from the Archaea, Eukarya, chloroplasts, and mitochondria.

In [ ]:

```
%%bash
cd references
wget http://www.mothur.org/w/images/5/59/Trainset9_032012.pds.zip
unzip Trainset9_032012.pds.zip

wget http://www.mothur.org/w/images/9/98/Silva.bacteria.zip
unzip Silva.bacteria.zip
mv silva.bacteria/silva.bacteria.fasta ./

cd ../
```

Now we want to extract the flowgram data from the sff files. Let's move to the sff_info folder. Then we'll want to run sffinfo from mothur on each of the sff files. When we actually ran this we submitted these jobs in parallel using pbs rather than in series and we are limited to about 100 jobs per user. You can find the actual sff files that we processed in sff_files.txt. We had 12943 sff files:

In [ ]:

```
%%bash
cd sffinfo

> sffinfo.batch

for SFF in $(ls ../sff_files/*sff)
do
    #make a big file with all of the sffinfo commands in the file
    echo "mothur \"#sffinfo(sff=$SFF, outputdir=./)\"" >> sffinfo.batch
done

#split sffinfo.batch in to files with 130 lines each and named with sffinfo_??
split -l 130 sffinfo.batch sffinfo_

cat > header.batch << "_EOF_"
#!/bin/sh
#PBS -l nodes=1:ppn=1
#PBS -l walltime=500:00:00
#PBS -l mem=1gb
#PBS -j oe
#PBS -m abe
#PBS -V
#PBS -M your@email.here
#PBS -q first

echo "ncpus-2.pbs"
cat $PBS_NODEFILE
qstat -f $PBS_JOBID

cd $PBS_O_WORKDIR
```

```
NCPUS=`wc -l $PBS_NODEFILE | awk '{print $1}'`

_EOF_


cat > tail.batch << "_EOF_"

echo "qsub working directory absolute is"
echo $PBS_O_WORKDIR
exit
_EOF_

for BATCH in $(ls sffinfo_*)
do
    cat header.batch $BATCH tail.batch > $BATCH.batch
    qsub $BATCH.batch
done

#there were those screwy files above that needed to be copied for v13 and v35 a
nd removed
#be sure to run these only after the scripts are done
cp F5672XE02.flow F5672XE02.v13.flow
cp F5672XE02.flow F5672XE02.v35.flow
rm F5672XE02.flow

#check to make sure we have the right number of files. If not, some baby sittin
g may be needed
ls ../sff_files/*sff | wc -l    #should equal 12943
ls *flow | wc -l                #should equal 12944
```

As mentioned above there was a PPS stage of the project and then the actual production stages. Accidentally, some of the data from WUGSC was uploaded in the production stage even though it had already been uploaded in the PPS stage. To keep track of what was from the PPS stage and what was from the production stage, we ran the following Perl script (modPPSFlowFiles.pl) on the flow gram data from the PPS stage go ahead and save a copy to the scripts folder:

In [ ]:

```
#!perl
use strict;
use warnings;

foreach my $file (@ARGV){

        open(INPUT, $file) or die;

        $file =~ /(.*).flow/;
        my $outfile = "$1.mod.flow";

        open(OUTPUT, ">$outfile") or die;


        my $header = <INPUT>;
        print OUTPUT $header;

        while(<INPUT>){
                $_ =~ s/^(\S*)/$1p/;
                print OUTPUT $_;
        }
        close(INPUT);
        close(OUTPUT);
}
```

This generates new files ending in "mod.flow" where the sequence name has a "p" appended to the end. The SRA files all start with "SRR" and since the PPS files weren't in the SRA, we'll use that to our advantage to pick those files out...

In [ ]:

```
%%bash
perl ../scripts/modPPSFlowFiles.pl `ls *flow | grep -v "^SRR"`
```

Now we want to move over to the trim.flows folder and run trim.flows. As described in Schloss et al. 2009 (http://aem.asm.org/content/75/23/7537.long) we will allow one mismatch to the barcode sequence, two mismatches to the primer sequence, no ambiguous base calls, and a maximum homopolymer length of 8 nt. Also, by default, mothur will trim all reads to 450 flows and cull any reads with fewer than 450 flows. This results in reads that are about 270 bp. Again we parallelize this and fire everything off in batches of 130 flow files at a time.

In [ ]:

```
%%bash
```

```
cd ../trim.flows

> trimflows.batch

for FLOW in $(ls ../sffinfo/SRR*flow)
do
    OLIGOS=${FLOW//sffinfo/oligos_files}
    OLIGOS=${OLIGOS//flow/oligos}

    echo "mothur \"#trim.flows(flow=$FLOW, oligos=$OLIGOS, pdiffs=2, bdiffs=1,
maxhomop=8, outputdir=./)\"" >> trimflows.batch
done

for FLOW in $(ls ../1_sffinfo/*.mod.flow)
do
    OLIGOS=${FLOW//1_sffinfo/oligos_files}
    OLIGOS=${OLIGOS//mod.flow/oligos}

    echo "mothur \"#trim.flows(flow=$FLOW, oligos=$OLIGOS, pdiffs=2, bdiffs=1,
maxhomop=8, processors=8, outputdir=./)\"" >> trimflows.batch
done


split -l 130 trimflows.batch trimflows_   #split trimflows.batch in to files wi
th 130 lines each and named with trimflows_??

cat > header.batch << "_EOF_"
#!/bin/sh
#PBS -l nodes=1:ppn=1
#PBS -l walltime=500:00:00
#PBS -l mem=1gb
#PBS -j oe
#PBS -m abe
#PBS -V
#PBS -M your@email.here
#PBS -q first

echo "ncpus-2.pbs"
cat $PBS_NODEFILE
qstat -f $PBS_JOBID

cd $PBS_O_WORKDIR

NCPUS=`wc -l $PBS_NODEFILE | awk '{print $1}'`

_EOF_


cat > tail.batch << "_EOF_"

echo "qsub working directory absolute is"
echo $PBS_O_WORKDIR
exit
```

```
_EOF_


for BATCH in $(ls trimflows_*)
do
    cat header.batch $BATCH tail.batch > $BATCH.batch
    qsub $BATCH.batch
done
```

This generates files ending in *v13.flow, *v35.flow, and *v69.flow as well as a file ending in *.flow.files that corresponds to each sff file that we started with. After running trim.flows, we had 12944 *.flow.files files, which represented 42,768 samples. Of these *.flow.files files 176 had no samples in them indicating that trim.flows was unable to find any sequences with the right barcode or primer. Manual inspection of the original flow files showed that these were samples with very few sequences.

In [ ]:

```
%%bash
wc -l *files > flow.files.counts
```

In [ ]:

```
%%R
a <-read.table(file="flow.files.counts"); colnames(a) <- c("nsamples", "sample_
id")
sum(a$nsamples)                     #total number of samples remaining
sum(a$nsamples==0)                  #number of flow.files files without any sam
ples
nrow(a) - sum(a$nsamples==0)        #number of flow.files with samples in them
b <- a[a$nsamples!=0,]
write.table(b$sample_id, "good.flow.files", row.names=F, col.name=F, quote=F)
  #output the names of the good files
```

We pressed on with the 12,768 *.flow.files files that had samples in them. At this point we could have separated the three 16S rRNA gene sequence regions, but we just plugged ahead with everything together. The next step was to denoise the sequences using the PyroNoise algorithm, which is implemented in mothur as shhh.flows. Investigators interested in repeating our analysis should note that this step involves a random number generator to break ties during the initial clustering step - the effects should be minor. This step may need a bit of baby siting depending on how much RAM you allocate and how much is needed for each job. Some may crap out and you'll have to figure out whether anything crapped out. We'll show how this was done at the end of the step.

In [ ]:

```
%%bash
cd ../shhh.flows
cd ../shhh.flows


> shhhflows.batch

for FLOWFILE in $(ls ../trim.flows/*flow.files)
do
    echo "mothur \"#shhh.flows(file=$FLOWFILE, outputdir=./)\"" >> shhhflows.ba
tch
done



split -l 20 shhhflows.batch shhhflows_   #split trimflows.batch in to files wit
h 20 lines each and named with shhhflows_??

cat > header.batch << "_EOF_"
#!/bin/sh
#PBS -l nodes=1:ppn=1
#PBS -l walltime=500:00:00
#PBS -l mem=8gb
#PBS -j oe
#PBS -m abe
#PBS -V
#PBS -M your@email.here
#PBS -q first

echo "ncpus-2.pbs"
cat $PBS_NODEFILE
qstat -f $PBS_JOBID

cd $PBS_O_WORKDIR

NCPUS=`wc -l $PBS_NODEFILE | awk '{print $1}'`

_EOF_


cat > tail.batch << "_EOF_"

echo "qsub working directory absolute is"
echo $PBS_O_WORKDIR
exit
_EOF_

for BATCH in $(ls shhhflows_*)
do
    cat header.batch $BATCH tail.batch > $BATCH.batch
    qsub $BATCH.batch
done
```

We found that this step required some handholding as certain files needed more RAM devoted to it and others would unexpectedly fail. To make sure everything went right we used a few tricks to make sure we had everything

In [ ]:

```
#how many shhh.fasta files were created (ignoring the indivdiual ones for each sample)?
ls *shhh.fasta | grep -cv "v"
#12766 - three short of what we found above...

ls *shhh.fasta | grep -v "v" > shhh.files
cut -c 9- ../trim.flows/flow.files.counts > flow.files
cat shhh.files ../trim.flows/good.flow.files  | rev | cut -f 3- -d "." | rev | sort | uniq -u

#F5672XE02.v13.mod
#F5672XE02.v35.mod
#total
```

So this makes sense - the two files that end in mod were inadvertantly left out in the line with "grep -v" and we had a total after running the wc command before. So we're all here adn ready to move on! Running shhh.flows results in a fasta file and a name file. The name file indicates the names of the sequences that were duplicates and were original sequence after denoising. We'll have to include this in all of the commands going forward so that we can have a proper accounting of the sequences.

Now we were ready to remove the barcodes and primers using trim.seqs. This step also generates a "groups" file, that indicate which sample each sequence belongs to. We will use many of the same parameters as we used in running trim.flows and we will use allfiles=T and flip=T. The former is so that we get a separate fasta, name, and group file for each sample in the original sff file and the latter is to return the reverse complement of the sequences since they were sequenced from the V5 region towards the V3 region

In [ ]:

```
%%bash
cd ../trim.seqs

> trimseqs.batch

for FASTA in $(ls ../shhh.flows/SRR??????.shhh.fasta)
do
    OLIGOS=${FASTA//shhh.flows/oligos_files}
    OLIGOS=${OLIGOS//shhh.fasta/oligos}

    NAMES=${FASTA//fasta/names}
    echo "mothur \"#trim.seqs(fasta=$FASTA, name=$NAMES, oligos=$OLIGOS, pdiffs
=2, bdiffs=1, maxhomop=8, maxambig=0, allfiles=T, flip=T, outputdir=./)\"" >> t
rimseqs.batch
done

for FASTA in $(ls ../shhh.flows/*mod.shhh.fasta)
do
    OLIGOS=${FASTA//shhh.flows/oligos_files}
    OLIGOS=${OLIGOS//mod.shhh.fasta/oligos}

    NAMES=${FASTA//fasta/names}
    echo "mothur \"#trim.seqs(fasta=$FASTA, name=$NAMES, oligos=$OLIGOS, pdiffs
=2, bdiffs=1, maxhomop=8, maxambig=0, allfiles=T, flip=T, outputdir=./)\"" >> t
rimseqs.batch
done


split -l 20 trimseqs.batch trimseqs_   #split trimseqs.batch in to files with 2
0 lines each and named with trimseqs_??

cat > header.batch << "_EOF_"
#!/bin/sh
#PBS -l nodes=1:ppn=1
#PBS -l walltime=500:00:00
#PBS -l mem=1gb
#PBS -j oe
#PBS -m abe
#PBS -V
#PBS -M your@email.here
#PBS -q first

echo "ncpus-2.pbs"
cat $PBS_NODEFILE
qstat -f $PBS_JOBID

cd $PBS_O_WORKDIR

NCPUS=`wc -l $PBS_NODEFILE | awk '{print $1}'`

_EOF_


cat > tail.batch << "_EOF_"

echo "qsub working directory absolute is"
echo $PBS_O_WORKDIR
exit
_EOF_

for BATCH in $(ls trimseqs_*)
do
    cat header.batch $BATCH tail.batch > $BATCH.batch
    qsub $BATCH.batch
```

```
done


ls ls ../trim.flows/*v??.flow | cut -f 3 -d "/" > flow.files; wc -l flow.files
  #21383
ls *.v??.fasta > fasta.files; wc -l fasta.files
  #21383
```

Running trim.seqs returns a fasta, name, and group file for each sample. Because of how we named our barcodes in the original oligos files we now have files named like: SRR350309.shhh.phase2.SRR350309.WUGSC.700172315.v35.fasta in our directory. Everything in this file name from phase2 through v35 comes from the oligos file. At this point we will separate our data into separate folders according to the region of the gene that the data were derived from.

In [ ]:

```
%%batch

cd ..

for REGION in v13 v35 v69
do
    cd $REGION
    cat ../trim.seqs/*$REGION.fasta > $REGION.fasta
    cat ../trim.seqs/*$REGION.names > $REGION.names
    cat ../trim.seqs/*$REGION.groups > $REGION.groups
    cd ../
done

cut -f 2 v13/v13.groups | sort | uniq | wc -l   #7194
cut -f 2 v35/v35.groups | sort | uniq | wc -l   #14015
cut -f 2 v69/v69.groups | sort | uniq | wc -l   #171
```

As mentioned above and indicated in running the last three commands, the v35 dataset has the most samples and that is what we based our analysis on for the paper. We'll go ahead and move into the v35 folder, but before proceeding, let's customize the reference files. To improve the analysis and reduce file sizes, we want to limit the reference files to the v35 region that was sequenced. As we are using the SILVA reference alignment (Pruesse et al. 2007 (http://nar.oxfordjournals.org/lookup/pmid?view=long&pmid=17947321)) we figured out that the primers that were used to amplify the DNA meant that a full length amplicon would start at position 6,426 in the alignment space and end at position 27,654 (the full alignment is 50,000 characters long). So we want to filter the silva.bacteria.fasta file based on those coordinates and then align the trainset9 fasta data to that region and filter it to remove the bases outside of the v35 region (Wang et al. 2007 (http://aem.asm.org/content/73/16/5261.long)).

In [ ]:

```
%%bash
cd v35

mothur "#pcr.seqs(fasta=../references/silva.bacteria.fasta, start=6426, end=276
54, outputdir=./, processors=8);filter.seqs(fasta=silva.bacteria.pcr.fasta, tru
mp=., processors=8)"
cp silva.bacteria.pcr.filter.fasta silva.v35.fasta

mothur "#align.seqs(fasta=../references/trainset9_032012.pds.fasta, reference=.
./references/silva.bacteria.fasta, outputdir=./, processors=8); pcr.seqs(fasta=
trainset9_032012.pds.align, taxonomy=../references/trainset9_032012.pds.tax, st
art=6426, end=27654, outputdir=./); degap.seqs(fasta=trainset9_032012.pds.pcr.a
lign)"
cp trainset9_032012.pds.pcr.tax trainset9_032012.three.tax
cp trainset9_032012.pds.pcr.ng.fasta trainset9_032012.three.fasta
```

Now we are ready to run mothur with the sequences as a single dataset. What follows are the commands to be run within mothur or placed within a batch file (i.e. v35.mothur.batch; shown below) and run from the command line. Relevant references for these commands are available: aligner (Schloss 2009 (http://www.plosone.org/article/info%3Adoi%2F10.1371%2Fjournal.pone.0008230) and DeSantis et al. 2006 (http://nar.oxfordjournals.org/content/34/suppl_2/W394.full)), pre-clustering (Schloss et al. 2011 (http://dx.plos.org/10.1371/journal.pone.0027310)), UChime (Edgar et al. 2011 (http://bioinformatics.oxfordjournals.org/content/27/16/2194.long)), and the classifier (Schloss & Westcott 2011 (http://aem.asm.org/content/77/10/3219.long) and Wang et al. 2007 (http://aem.asm.org/content/73/16/5261.long)). We'll first load the mothur magic module:

In [2]:

```
%install_ext https://raw.github.com/kdiverson/ipython-mothurmagic/master/mothur
magic.py
%load_ext mothurmagic
```

```
Installed mothurmagic.py. To use it, type:
  %load_ext mothurmagic
The mothurmagic extension is already loaded. To reload it, use:
  %reload_ext mothurmagic
```

In [ ]:

```
%%mothur

#unique the sequences because there are duplicate sequences between samples
unique.seqs(fasta=v35.fasta, name=v35.names)



#align the sequences against the customized SILVA reference alignment for the v
```

```
35 region
#the user should alter the processors option to however many processors they ha
ve available
align.seqs(fasta=current, reference=silva.v35.fasta, processors=8)


#find the start and end coordinates for the aligned sequences
summary.seqs(fasta=current, name=current)


#remove the sequences that don't map to the correct region and then remove
#any vertical gaps or columns that contain missing data
screen.seqs(fasta=current, name=current, group=v35.groups, start=10010, end=212
19)
filter.seqs(fasta=current, vertical=T, trump=.)


#unique again because the end trimming creates duplicate sequences
unique.seqs(fasta=current, name=current)

#pre-cluster sequences to remove residual errors and simplify the dataset.
pre.cluster(fasta=current, group=current, name=current, diffs=2, processors=8)


#use chimera.uchime to identify chimeras in the de novo detection mode. to do t
his, we
#will process each sample separately and will will only consider a sequence as
being
#chimeric if it is flagged as chimeric in that sample. we then remove the chime
ras from
#the samples they were found in
chimera.uchime(fasta=current, group=current, name=current, dereplicate=T, proce
ssors=8)
remove.seqs(fasta=current, group=current, name=current, accnos=current, dups=F)


#classify the sequences against our customized taxonomy reference based on the
RDP
#training set. we require that an assignment have a confidence score of at leas
t 80%
classify.seqs(fasta=current, reference=trainset9_032012.three.fasta, taxonomy=t
rainset9_032012.three.tax, cutoff=80, processors=8)


#remove sequences that classify as either Chloroplasts, Mitochondria, Archaea,
Eukaryota,
#or as being unknown
remove.lineage(fasta=current, group=current, name=current, taxonomy=current, ta
xon=Cyanobacteria_Chloroplast-Mitochondria-unknown-Archaea-Eukaryota)


#assign sequences to bins / phylotypes based on their taxonomic assignment
phylotype(taxonomy=current, name=current)
```

```
#count the number of times each phylotype was observed in each sample at the ge
nus level
#(label=1)
make.shared(list=current, group=current, label=1)


#finally, output the taxonomic assignment for each phylotype
classify.otu(list=current, group=current, name=current, taxonomy=current, label
=1)
```

Recall above we gave the samples some pretty funky names to keep every sequencing collection separate. We need to figure out which samples came from which subject and bodysite. We pooled datasets for each subject and bodysite where a sample was sequenced multiple times. Because WUGSC submitted their PPS data for the Phase 1 data, there is some redundancy. Now we need to remove the redundant reads with the help of a perl script (removeRedundant_WUGSC.pl). Go ahead and copy and paste the following into a file in the scripts folder.

In [ ]:

```
#!perl

use strict;
use warnings;


#       first we want to get the names of the WUGSC groups that were sequenced 
in
#       the pps.  we'll create the %groups hash and store the group names in th
e
#       hash...

my %groups;

open(GROUPS, $ARGV[0]) or die;
while(<GROUPS>){
        chomp(my $group = $_);
        $groups{$group} = 1;
}

close(GROUPS);


#       next, we want to look through the shared file and find any group name t
hat
#       starts "phase1" and ends with the group name in the hash.  if it isn't 
in
#       the hash then we'll keep it.  if not, we'll remove it.

open(ORIG_SHARED, $ARGV[1]) or die;
while(<ORIG_SHARED>){
        if($_ =~ /phase1\..*\.(WUGSC\.\S*)\t/){
                if(!exists($groups{$1})){
                        print "$_";
                }
        }
        else{
                print "$_";
        }
}

close(ORIG_SHARED);
```

In [ ]:

```
%%bash
#get the names of the samples from the shared file that came from WUGSC in the
PPS phase
#return the sequencing center (WUGSC; f=3) and the anonymized sample id (f=4)
cut -f 2 v35.unique.good.filter.unique.precluster.unique.pick.three.wang.pick.t
x.shared | grep "pps.*WUGSC" | cut -f 3,4 -d "." | sort | uniq > wugsc.pps.grou
ps

#remove any samples that start phase1 and end with the fields returned above
perl ../scripts/removeRedundant_WUGSC.pl wugsc.pps.groups v35.unique.good.filte
r.unique.precluster.unique.pick.three.wang.pick.tx.shared > v35.three_phases.tx
.shared
```

There were a number of "control" samples included in the analysis from sequencing mock communities, generous donor samples, and water blanks. Unfortunately, once Phase 1 and 2 started, the sequencing centers became inconsistent in running these controls. Therefore, they don't really tell us anything other than what we've already established in our earlier efforts to build this pipeline. So we'll remove them using a perl script (removeControlSamples.pl)

#!perl use strict; use warnings; open(SHARED, $ARGV[0]) or die; #the control samples all had underscores and none of the other samples did while(){ if(!(_ =~ /_/)){ print _ =~ /_/)){ print _; } } close(SHARED);

In [ ]:

```
%%bash
perl ../scripts/removeControlSamples.pl v35.three_phases.tx.shared > v35.three_
phases_nocontrol.tx.shared
```

Now we want to figure out who the anonymized sample id corresponds to and which body site it came from as well as the visit number that the sample was taken from. To do this, it is essential that you have the dbGaP files. I'll assume that you have the following files in a folder named dbGaP that is located in the parent directory of v35 (the current directory):

```
phs000228.v3.pht001184.v3.p1.EMMES_HMP_Subject.MULTI.txt
phs000228.v3.pht001185.v3.p1.EMMES_HMP_Sample.MULTI.txt
phs000228.v3.pht001187.v3.p1.c1.EMMES_HMP_DCM.HMP.txt
phs000228.v3.pht001193.v3.p1.c1.EMMES_HMP_GTV.HMP.txt
phs000228.v3.pht002156.v1.p1.c1.EMMES_HMP_DSU.HMP.txt
phs000228.v3.pht002157.v1.p1.c1.EMMES_HMP_DTP_DHX_DVD.HMP.txt
phs000228.v3.pht002158.v1.p1.c1.EMMES_HMP_DEM_ENR.HMP.txt
```

Again, don't ask me for these files, you must get them through dbGaP. For this operation you will need phs000228.v3.pht001193.v3.p1.c1.EMMES_HMP_GTV.HMP.txt and our perl script (mapSamplesToMetadata.pl), which should be in the scripts folder.

In [ ]:

```
#!perl

use strict;
use warnings;


my %tracker;

open(METADATA, $ARGV[1]) or die;
while(<METADATA>){
        chomp($_);
        $_ =~ s/ /_/g;

        if($_ =~ /^\d/){
                my @line = split /\t/, $_;

                $line[7] =~ s/(\d\d).*/$1/;
                $line[9] =~ s/.*DNA_(.*)/$1/;

                $tracker{$line[6]} = "$line[13].$line[7].$line[9]";
        }
}
close(METADATA);


open(SAMPLE_IDS, $ARGV[0]) or die;
while(<SAMPLE_IDS>){
        chomp($_);
        if($_ =~ /(\d{9})/){

                print "$_\t$tracker{$1}\n";

        }
}
close(SAMPLE_IDS);
```

In [ ]:

```
%%bash

#get the names of all of the samples we have
cut -f 2 v35.three_phases_nocontrol.tx.shared > sample.ids


#map the sample.ids to the subject id, body site, and visit number
perl ../scripts/mapSamplesToMetadata.pl sample.ids ../dbGaP/phs000228.v3.pht001
193.v3.p1.c1.EMMES_HMP_GTV.HMP.txt > sample.design


#the output of the perl script is a file that maps the sample names to more
#meaningful names. using mothur, we can merge the frequency of each phylotype
#for samples that need to be pooled
mothur "#merge.groups(shared=v35.three_phases_nocontrol.tx.shared, design=sampl
e.design)"
```

At this point we have sample names that look like this:

> 160461578.01.Throat
> 158256496.02.Anterior_nares
> 160643649.01.Mid_vagina
> 159005010.02.Tongue_dorsum
> 159753524.01.Tongue_dorsum
> 160481808.01.R_Antecubital_fossa
> 159753524.01.Anterior_nares
> 160481808.01.Tongue_dorsum
> 159207311.01.R_Retroauricular_crease
> 159713063.02.Tongue_dorsum

We'd like to separate these samples now so that each body site has its own shared file:

In [ ]:

```
%%bash
head -n 1 v35.three_phases_nocontrol.tx.merge.shared > header.txt
cp header.txt Anterior_nares.tx.shared
cp header.txt Buccal_mucosa.tx.shared
cp header.txt Hard_palate.tx.shared
cp header.txt Keratinized_gingiva.tx.shared
cp header.txt L_Antecubital_fossa.tx.shared
cp header.txt L_Retroauricular_crease.tx.shared
cp header.txt Mid_vagina.tx.shared
cp header.txt Palatine_Tonsils.tx.shared

cp header.txt Posterior_fornix.tx.shared
cp header.txt R_Antecubital_fossa.tx.shared
cp header.txt R_Retroauricular_crease.tx.shared
cp header.txt Saliva.tx.shared
cp header.txt Stool.tx.shared
cp header.txt Subgingival_plaque.tx.shared
cp header.txt Supragingival_plaque.tx.shared
cp header.txt Throat.tx.shared
cp header.txt Tongue_dorsum.tx.shared
cp header.txt Vaginal_introitus.tx.shared

grep "Anterior_nares" v35.three_phases_nocontrol.tx.merge.shared >> Anterior_na
res.tx.shared
grep "Buccal_mucosa" v35.three_phases_nocontrol.tx.merge.shared >> Buccal_mucos
a.tx.shared
grep "Hard_palate" v35.three_phases_nocontrol.tx.merge.shared >> Hard_palate.tx
.shared
grep "Keratinized_gingiva" v35.three_phases_nocontrol.tx.merge.shared >> Kerati
nized_gingiva.tx.shared
grep "L_Antecubital_fossa" v35.three_phases_nocontrol.tx.merge.shared >> L_Ante
cubital_fossa.tx.shared
grep "L_Retroauricular_crease" v35.three_phases_nocontrol.tx.merge.shared >> L_
Retroauricular_crease.tx.shared
grep "Mid_vagina" v35.three_phases_nocontrol.tx.merge.shared >> Mid_vagina.tx.s
hared
grep "Palatine_Tonsils" v35.three_phases_nocontrol.tx.merge.shared >> Palatine_
Tonsils.tx.shared
grep "Posterior_fornix" v35.three_phases_nocontrol.tx.merge.shared >> Posterior
_fornix.tx.shared
grep "R_Antecubital_fossa" v35.three_phases_nocontrol.tx.merge.shared >> R_Ante
cubital_fossa.tx.shared
grep "R_Retroauricular_crease" v35.three_phases_nocontrol.tx.merge.shared >> R_
Retroauricular_crease.tx.shared
grep "Saliva" v35.three_phases_nocontrol.tx.merge.shared >> Saliva.tx.shared
grep "Stool" v35.three_phases_nocontrol.tx.merge.shared >> Stool.tx.shared
grep "Subgingival_plaque" v35.three_phases_nocontrol.tx.merge.shared >> Subging
ival_plaque.tx.shared
grep "Supragingival_plaque" v35.three_phases_nocontrol.tx.merge.shared >> Supra
gingival_plaque.tx.shared
grep "Throat" v35.three_phases_nocontrol.tx.merge.shared >> Throat.tx.shared
grep "Tongue_dorsum" v35.three_phases_nocontrol.tx.merge.shared >> Tongue_dorsu
m.tx.shared
grep "Vaginal_introitus" v35.three_phases_nocontrol.tx.merge.shared >> Vaginal_
introitus.tx.shared
```

In addition several of the body sites will be combined to facilitate their analysis (i.e. left and right antecubital fossa and retroauricular crease and the vaginal introitus, mid vagina, and posterior fornix).

In [ ]:

```
%%bash
cp header.txt Vagina.tx.shared
cp header.txt Retroauricular_crease.tx.shared
cp header.txt Antecubital_fossa.tx.shared

grep "Vaginal_introitus" v35.three_phases_nocontrol.tx.merge.shared >> Vagina.t
x.shared
grep "Mid_vagina" v35.three_phases_nocontrol.tx.merge.shared >> Vagina.tx.share
d
grep "Posterior_fornix" v35.three_phases_nocontrol.tx.merge.shared >> Vagina.tx
.shared
grep "L_Antecubital_fossa" v35.three_phases_nocontrol.tx.merge.shared >> Antecu
bital_fossa.tx.shared
grep "R_Antecubital_fossa" v35.three_phases_nocontrol.tx.merge.shared >> Antecu
bital_fossa.tx.shared
grep "L_Retroauricular_crease" v35.three_phases_nocontrol.tx.merge.shared >> Re
troauricular_crease.tx.shared
grep "R_Retroauricular_crease" v35.three_phases_nocontrol.tx.merge.shared >> Re
troauricular_crease.tx.shared
```

All of our next steps will depend on our ability to subsample the data to a common number of sequences. To figure out what our threshold should be, we will calculate the number of reads per sample and bodysite.

In [ ]:

```
%%bash
mothur "#summary.single(shared=Anterior_nares.tx.shared, calc=nseqs);"
mothur "#summary.single(shared=Buccal_mucosa.tx.shared, calc=nseqs);"
mothur "#summary.single(shared=Hard_palate.tx.shared, calc=nseqs);"
mothur "#summary.single(shared=Keratinized_gingiva.tx.shared, calc=nseqs);"
mothur "#summary.single(shared=L_Antecubital_fossa.tx.shared, calc=nseqs);"
mothur "#summary.single(shared=L_Retroauricular_crease.tx.shared, calc=nseqs);"
mothur "#summary.single(shared=Mid_vagina.tx.shared, calc=nseqs);"
mothur "#summary.single(shared=Palatine_Tonsils.tx.shared, calc=nseqs);"
mothur "#summary.single(shared=Posterior_fornix.tx.shared, calc=nseqs);"
mothur "#summary.single(shared=R_Antecubital_fossa.tx.shared, calc=nseqs);"
mothur "#summary.single(shared=R_Retroauricular_crease.tx.shared, calc=nseqs);"
mothur "#summary.single(shared=Saliva.tx.shared, calc=nseqs);"
mothur "#summary.single(shared=Stool.tx.shared, calc=nseqs);"
mothur "#summary.single(shared=Subgingival_plaque.tx.shared, calc=nseqs);"
mothur "#summary.single(shared=Supragingival_plaque.tx.shared, calc=nseqs);"
mothur "#summary.single(shared=Throat.tx.shared, calc=nseqs);"
mothur "#summary.single(shared=Tongue_dorsum.tx.shared, calc=nseqs);"
mothur "#summary.single(shared=Vaginal_introitus.tx.shared, calc=nseqs);"
```

So let's use some R to figure out what percentage of the samples at each body site will be kept for various thresholds

```
In [ ]:
```

```
%%R

threshold <- 1000       #change this to see how the fraction of samples changes

summary.files <- c("Anterior_nares.tx.groups.summary", "Buccal_mucosa.tx.groups
.summary",
                                "Hard_palate.tx.groups.summary", "Keratinized_g
ingiva.tx.groups.summary",
                                "L_Antecubital_fossa.tx.groups.summary", "L_Ret
roauricular_crease.tx.groups.summary",
                                "Mid_vagina.tx.groups.summary", "Palatine_Tonsi
ls.tx.groups.summary",
                                "Posterior_fornix.tx.groups.summary", "R_Antecu
bital_fossa.tx.groups.summary",
                                "R_Retroauricular_crease.tx.groups.summary", "S
aliva.tx.groups.summary",
                                "Stool.tx.groups.summary", "Subgingival_plaque.
tx.groups.summary",
                                "Supragingival_plaque.tx.groups.summary", "Thro
at.tx.groups.summary",
                                "Tongue_dorsum.tx.groups.summary", "Vaginal_int
roitus.tx.groups.summary")

percent <- numeric()

for(file in summary.files){
        data <- read.table(file=file, header=T)
        percent[file] <- sum(data$nseqs >= threshold)/nrow(data)
}

percent
hist(percent)
```

If you play around with a few thresholds and look at the percentages you'll see that the antecubital foss and vaginal samples have a small number of reads in general. But if you start increasing the threshold above 1000 we start to lose a good number of samples from the other bodysites. We chose to go with 1000 because it gave us a good number of samples and reads per body site. With this information, we subsampled all of the bodysites to 1000 reads per sample. Any samples with fewer than 1000 reads were removed from the analysis.

In [ ]:

```
%%mothur
sub.sample(shared=Antecubital_fossa.tx.shared, size=1000)
sub.sample(shared=Anterior_nares.tx.shared, size=1000)
sub.sample(shared=Buccal_mucosa.tx.shared, size=1000)
sub.sample(shared=Hard_palate.tx.shared, size=1000)
sub.sample(shared=Keratinized_gingiva.tx.shared, size=1000)
sub.sample(shared=Palatine_Tonsils.tx.shared, size=1000)
sub.sample(shared=Retroauricular_crease.tx.shared, size=1000)
sub.sample(shared=Saliva.tx.shared, size=1000)
sub.sample(shared=Stool.tx.shared, size=1000)
sub.sample(shared=Subgingival_plaque.tx.shared, size=1000)
sub.sample(shared=Supragingival_plaque.tx.shared, size=1000)
sub.sample(shared=Throat.tx.shared, size=1000)
sub.sample(shared=Tongue_dorsum.tx.shared, size=1000)
sub.sample(shared=Vagina.tx.shared, size=1000)
```

Assigning samples from each body site to community types

Now we'd like to assign the samples in each body site to community types. We do this using the get.communitytype function in mothur (Holmes et al. 2012 (http://www.plosone.org/article/info%3Adoi%2F10.1371%2Fjournal.pone.0030126)). This is a function that utilizes a random number generator and so there may be run-to-run variation. In our experience, with complicated datasets such as these, there is very little variation in the number of community types or the assignments. The label of the community type is most likely to vary between runs (e.g. in one run a group of samples might be labeled as "Partition_1" and in another "Partition_3"). We ran the everything through get.communitytype five times in separate folders:

In [ ]:

```
%%bash

mkdir dmm_analysis
cd dmm_analysis
mkdir rep1 rep2 rep3 rep4 rep5


cat > header.batch << "_EOF_"
#!/bin/sh
#PBS -l nodes=1:ppn=1
#PBS -l walltime=500:00:00
#PBS -l mem=1gb
#PBS -j oe
#PBS -m abe
#PBS -V
#PBS -M your@email.here
#PBS -q first
```

```
echo "ncpus-2.pbs"

cat $PBS_NODEFILE
qstat -f $PBS_JOBID

cd $PBS_O_WORKDIR

NCPUS=`wc -l $PBS_NODEFILE | awk '{print $1}'`

_EOF_


cat > tail.batch << "_EOF_"

echo "qsub working directory absolute is"
echo $PBS_O_WORKDIR
exit
_EOF_



for REP in rep1 rep2 rep3 rep4 rep5
do
    cd $REP

    for SHARED in $(ls ../../*subsample.shared)
    do
            SITE=${SHARED//..\/..\//}
        SITE=${SITE//.tx.1.subsample.shared/}

        cat ../header.batch > $REP.$SITE.batch
        echo "mothur \"#get.communitytype(shared=$SHARED, outputdir=./)\"" >> $
REP.$SITE.batch
        cat ../tail.batch >> $REP.$SITE.batch

        qsub $REP.$SITE.batch

    done
    cd ../
done
```

Next we needed to go through, find the number of community types that gave the best Laplace value for each body site and move the files to a single folder.

In [ ]:

```
%%R

bodysites <- c("Antecubital_fossa", "Anterior_nares", "Buccal_mucosa", "Hard_pa
late", "Keratinized_gingiva", "Palatine_Tonsils", "Retroauricular_crease", "Sal
iva", "Stool", "Subgingival_plaque", "Supragingival_plaque", "Throat", "Tongue_
dorsum", "Vagina")


bs.scores <- matrix(rep(NA, rep(14*3)), ncol=3)
colnames(bs.scores) <- c("min.iteration", "min.k", "min.laplace")
rownames(bs.scores) <- bodysites

dir.create("dmm_best")

for(bs in bodysites){
        reps <- c("rep1", "rep2", "rep3", "rep4", "rep5")
        scores <- matrix(rep(NA, 5*10), ncol=5)
        colnames(scores) <- reps
        rownames(scores) <- 1:10

        for(r in reps){
                filename <- paste("dmm_analysis/", r,"/",bs, ".tx.1.subsample.1
.dmm.mix.fit", sep="")
                data <- read.table(file=filename, header=T, row.names=1)
                scores[1:nrow(data),r] <- data$Laplace
        }

        min.index <- which.min(scores)
        min.k <- (min.index) %% 10
        min.index <- min.index - min.k
        min.iteration <- (min.index / 10) + 1

        c(min.k, min.iteration, scores[min.k, min.iteration])

        bs.scores[bs, ] <- c(min.iteration, min.k, scores[min.k, min.iteration]
)

        source.filename <- list.files(path=paste("dmm_analysis/", colnames(scor
es)[min.iteration], sep=""), pattern=glob2rx(paste(bs, ".*", sep="")))
        write(paste("dmm_analysis/",colnames(scores)[min.iteration], "/", sourc
e.filename, sep=""), "")
        file.copy(paste("dmm_analysis/",colnames(scores)[min.iteration], "/", s
ource.filename, sep=""), to="dmm_best/")
}
```

The get.communitytype outputs a number of files that we will use. For now, let's worry about the design file, which will tell us which community type each sample belongs to. Let's get the body specific design files for those bodysites that we pooled

In [ ]:

```
%%bash
cd dmm_best
grep "L_Retroauricular_crease" Retroauricular_crease.tx.1.subsample.1.dmm.mix.d
esign > L_Retroauricular_crease.tx.1.subsample.1.dmm.mix.design
grep "R_Retroauricular_crease" Retroauricular_crease.tx.1.subsample.1.dmm.mix.d
esign > R_Retroauricular_crease.tx.1.subsample.1.dmm.mix.design
grep "L_Antecubital_fossa" Antecubital_fossa.tx.1.subsample.1.dmm.mix.design >
L_Antecubital_fossa.tx.1.subsample.1.dmm.mix.design
grep "R_Antecubital_fossa" Antecubital_fossa.tx.1.subsample.1.dmm.mix.design >
R_Antecubital_fossa.tx.1.subsample.1.dmm.mix.design
grep "Vaginal_introitus" Vagina.tx.1.subsample.1.dmm.mix.design > Vaginal_intro
itus.tx.1.subsample.1.dmm.mix.design
grep "Mid_vagina" Vagina.tx.1.subsample.1.dmm.mix.design > Mid_vagina.tx.1.subs
ample.1.dmm.mix.design
grep "Posterior_fornix" Vagina.tx.1.subsample.1.dmm.mix.design > Posterior_forn
ix.tx.1.subsample.1.dmm.mix.design
```

One last thing we'd like to do is to create separate files where each file is a table with the subject along the rows and the body site is along the columns.

In [ ]:

```
%%R
body.design <- c("Anterior_nares.tx.1.subsample.1.dmm.mix.design", "Buccal_muco
sa.tx.1.subsample.1.dmm.mix.design",
                                 "Hard_palate.tx.1.subsample.1.dmm.mix.design",
 "Keratinized_gingiva.tx.1.subsample.1.dmm.mix.design",
                                 "L_Antecubital_fossa.tx.1.subsample.1.dmm.mix.
design", "L_Retroauricular_crease.tx.1.subsample.1.dmm.mix.design",
                                 "Mid_vagina.tx.1.subsample.1.dmm.mix.design",
"Palatine_Tonsils.tx.1.subsample.1.dmm.mix.design",
                                 "Posterior_fornix.tx.1.subsample.1.dmm.mix.des
ign", "R_Antecubital_fossa.tx.1.subsample.1.dmm.mix.design",
                                 "R_Retroauricular_crease.tx.1.subsample.1.dmm.
mix.design", "Saliva.tx.1.subsample.1.dmm.mix.design",
                                 "Stool.tx.1.subsample.1.dmm.mix.design", "Subg
ingival_plaque.tx.1.subsample.1.dmm.mix.design",
                                 "Supragingival_plaque.tx.1.subsample.1.dmm.mix
.design", "Throat.tx.1.subsample.1.dmm.mix.design",
                                 "Tongue_dorsum.tx.1.subsample.1.dmm.mix.design
", "Vaginal_introitus.tx.1.subsample.1.dmm.mix.design")
```

```
subjects <- character()


#i know this isn't efficient...
for(design in body.design){
        d <- read.table(file=design)
        s <- gsub("^(\\d*).*", "\\1", d$V1)
        subjects <- c(subjects, s)
}

subjects <- sort(unique(subjects))
nsubjects <- length(subjects)   #should be ~300

visit1 <- matrix(rep(NA, nsubjects*18), nrow=nsubjects)
rownames(visit1) <- subjects
colnames(visit1) <- gsub("^([^.]*).*", "\\1", body.design)

visit2 <- visit1
visit3 <- visit1

for(design in body.design){
        d <- read.table(file=design)
        s <- gsub("^(\\d*).*", "\\1", d$V1)                             #get th
e subject id
        v <- gsub("^\\d*.(\\d\\d).*", "\\1", d$V1)              #get the visit
number
        b <- gsub("^\\d*.\\d\\d.(.*)", "\\1", d$V1)             #get the visit
number

        visit1[as.character(s[v == "01"]), b] <- as.character(d[v == "01", "V2"
])
        visit2[as.character(s[v == "02"]), b] <- as.character(d[v == "02", "V2"
])
        visit3[as.character(s[v == "03"]), b] <- as.character(d[v == "03", "V2"
])
}

sum.na <- function(x){
        return(sum(is.na(x)))
}

keep <- apply(visit1, 1, sum.na) != 18  #get rid of subject ids that only have
NAs
visit1 <- visit1[keep,]

keep <- apply(visit2, 1, sum.na) != 18  #get rid of subject ids that only have
NAs
visit2 <- visit2[keep,]

keep <- apply(visit3, 1, sum.na) != 18  #get rid of subject ids that only have
NAs
visit3 <- visit3[keep,]

write.table(visit1, "community_types_v1.txt", quote=F)
write.table(visit1, "community_types.v1.txt", quote=F)
write.table(visit2, "community_types.v2.txt", quote=F)

write.table(visit3, "community_types.v3.txt", quote=F)
```

In [ ]:

```
%%bash
mkdir ../../analysis
mv community_types.v* ../../analysis
```

Clinical metadata

A lot of clinical metadata were collected on the 300 subjects in this study. Unfortunately, you always think of more metadata that you would like to have after you start to look at the data. Another issue with clinical metadata, especially in the case of the HMP dataset was that because the subjects were so young and healthy, there often isn't a lot of variation and the metadata don't turn out to be as interesting as originally hoped. For instance, if only a few women have had a baby, it is difficult to see whether having a baby has affected her microbiome. Ditto for a bunch of other variables. Again, all of the clinical metadata are available through dbGaP and unfortunately, you have to get them there. There is a process to getting the data that is tedious but not horrible. But whatever you do, don't ask me for them. I'll say no. Finally, once you get the dbGaP files, unencrypt them and throw them into the dbgap folder that we made way back at the top of this notebook.

In [ ]:

```
%%bash
cd ../../dbGaP
ls *txt

#should produce...
#phs000228.v3.pht001184.v3.p1.EMMES_HMP_Subject.MULTI.txt
#phs000228.v3.pht001185.v3.p1.EMMES_HMP_Sample.MULTI.txt
#phs000228.v3.pht001187.v3.p1.c1.EMMES_HMP_DCM.HMP.txt
#phs000228.v3.pht001193.v3.p1.c1.EMMES_HMP_GTV.HMP.txt
#phs000228.v3.pht002156.v1.p1.c1.EMMES_HMP_DSU.HMP.txt
#phs000228.v3.pht002157.v1.p1.c1.EMMES_HMP_DTP_DHX_DVD.HMP.txt
#phs000228.v3.pht002158.v1.p1.c1.EMMES_HMP_DEM_ENR.HMP.txt
```

Before we do science, we need to sift through the clinical metadata and see what of it is useful and what isn't. Recall that you can go to the dbGaP project site (http://www.ncbi.nlm.nih.gov/projects/gap/cgi-bin/study.cgi?study_id=phs000228.v3.p1) to figure out what the various files contain and what the variable names mean. Because some of these txt files contain apostrophes that were wrecking havoc with R, I used TextWrangeler to zap all of them away. A smarter person could probalby have done it with sed. Good for you. The first thing we'll do is recreate the data in Extended Data Table 1 from the paper.

Three of these files contain useful categorical metadata that we'll read in, sort, and merge into one table. Also, because patients provided the data at different visits in some of the files, we will have to be careful about picking the right visit (VISNO=="01")...

In [ ]:

```
%%R
categ.1 <-read.table(file="phs000228.v3.pht002157.v1.p1.c1.EMMES_HMP_DTP_DHX_DV
D.HMP.txt", header=T, sep="\t",
                     na.string=c("", NA))
categ.1 <- categ.1[categ.1$VISNO == "01",]
rownames(categ.1) <- categ.1$RANDSID
categ.1 <- categ.1[,-c(ncol(categ.1))]  #the RANDSID column is the last one
categ.1 <- categ.1[sort(rownames(categ.1)),]


categ.2 <-read.table(file="phs000228.v3.pht002158.v1.p1.c1.EMMES_HMP_DEM_ENR.HM
P.txt", header=T, sep="\t", row.names=2,
                     na.string=c("", NA))
categ.2 <- categ.2[sort(rownames(categ.2)),]

duplicates <- intersect(colnames(categ.1), colnames(categ.2)) #the columns that
 are in both files
categ.2 <- categ.2[,-which(colnames(categ.2) %in% duplicates)]  #get rid of the
 duplicates
sum(rownames(categ.1) == rownames(categ.2)) == 300      #make sure the rows are
 in the same order

categorical <- cbind(categ.1, categ.2)


#using textwrangler I first zapped all of the apostrophes
categ.3 <- read.table(file="phs000228.v3.pht002156.v1.p1.c1.EMMES_HMP_DSU.HMP.t
xt", header=T, sep="\t", na.string=c("", NA))
rownames(categ.3) <- categ.3$RANDSID
categ.3 <- categ.3[,-c(ncol(categ.3))]  #the RANDSID column is the last column
setdiff(rownames(categ.3), rownames(categorical))
dim(categ.3)    #there weren't 300 subjects here and so some are missing

missing <- rownames(categorical)[!(rownames(categorical) %in% rownames(categ.3)
)]
categ.3[as.character(missing),] <- rep(NA, ncol(categ.3))

categ.3 <- categ.3[sort(rownames(categ.3)),]
duplicates <- intersect(colnames(categorical), colnames(categ.3)) #the columns
that are in both files
categ.3 <- categ.3[,-which(colnames(categ.3) %in% duplicates)]  #get rid of the
 duplicates
sum(rownames(categ.1) == rownames(categ.3)) == 300      #make sure the rows are
 in the same order

categorical <- cbind(categorical, categ.3)

ncats <- ncol(categorical)      #196
npeople <- nrow(categorical)
```

So now we have our massive table of 300 subjects by 196 metadata variables. Now we will convert "vague" answers to "NA" and then identify those metadata columns that were NA for every subject:

```
In [ ]:
```

```
%%R
for(i in colnames(categorical)){
        categorical[grepl("Dont know/remember",categorical[,i]),i] <- NA
        categorical[grepl("NotEvaluated",categorical[,i]),i] <- NA
        categorical[grepl("Forgot",categorical[,i]),i] <- NA
}

#there's a lot of crap in here, let's figure out how to remove the variables th
at are all NAs
sum.na <- function(x) sum(is.na(x) | x == "")
na.count <- apply(categorical, 2, sum.na)
notall.na <- na.count <= 250    #empirically determined number

#things with more than 250 NAs - ethnicities, tobacco usage, medical history, s
ome drug usage

categorical <- categorical[,notall.na]
ncats <- ncol(categorical)      #101

summary(categorical)
```

So we effectively reduced the amount of metadata in half. With the output of the summary command we see the various metadata fields. Looking through the dbGaP site we get a sense of what they are. A couple things pop out. First, many of the variables are numerically coded categoritcal varaibles (e.g. SITE) and some are the same thing but coded with a string (e.g. SITE_C). We only need one of each of these. Second, some variables aren't useful (e.g. PROT). Third, some variables are continuous (e.g. AGEENR) and will be dealt with separately. Finally, some variables merely indicate whether a sample was taken at that bodysite (e.g. DVDINTRO_C). Here are the variables we decided to remove and why...

Variables to remove: redundant / non-informative

> SITE - redundant with SITE_C
> DVDEDLVL_C - redundant with DVDEDLVL
> DVDINSRH - redundant with DVDINSRH_C
> DVDINSRD - redundant with DVDINSRD_C
> DVDOCPTN - redundant with DVDOCPTN_C
> DVDTOBC - redundant with DVDTOBC_C
> DVDPGRES - Pregnant? no one was pregnant
> DVDPGRES_C - Pregnant? no one was pregnant
> START - starting quarter of the study
> GENDER - redundnat with GENDER_C
> ETHNIC - redundant with ETHNIC_C
> AMERIND - american indian (only 1)
> AMERIND_C - american indian (only 1)
> ASIAN - redundant with ASIAN_C
> PACIFC - pacific islander (none)
> PACIFC_C - pacific islander (none)
> WHITE - redundant with WHITE_C
> BRTHCTRY - redundant with BRTHCTRY_C
> MOMCTRY - redundant with MOMCTRY_C
> DADCTRY - redundant with DADCTRY_C
> DSUSCRN - redundant with DSUSCRN_C
> DSUCON - redundant with DSUCON_C
> DSUDIET_C - redundant with DSUDIET
> DSUBIR - redunant with DSUBIR_C
> DSUBFED - redundant with DSUBFED_C
> BLACK - redundant with BLACK_C
> DVDVIRRT / DVDVIRRT_C - vaginal or vulval irritation - only 8 yes

not useful

> dbGaP.SubjID - dbGaP Subject ID
> PROT - Protocol
> PROTSEG - Protocol
> protseg - Protocol
> VISNO - Visit number
> study_day - day of the study
> dsuotrpd - time since last oral treatment - eh.
> dsuvspd - time since enrollment
> IDSUZIP - zip code
> DSUSCRN_C - screened more than once
> DSUCON_C - subject contact

Variables to remove: continuous variables

```
DVDINTPH - vagina introitus pH
DVDPFPH - posterior fornix pH
DVDTMPF - temperature in F
DVDTMPC - temperature in C
AGEENR - age
```

Variables to remove: indicate whether various body sites were sampled

```
DVDBLOOD / DVDBLOOD_C - blood
DVDSERUM / DVDSERUM_C - serum
DVDSTOOL / DVDSTOOL_C - stool
DVDSAL / DVDSAL_C - saliva
DVDGING / DVDGING_C - gingiva
DVDTONG / DVDTONG_C - tongue
DVDPTON / DVDPTON_C - tonsils
DVDHPAL / DVDHPAL_C - hard palate
DVDTHRO / DVDTHRO_C - throat
DVDBUCC / DVDBUCC_C - buccal mucosa
DVDSUPRA / DVDSUPRA_C - supragingival plaque
DVDSUB / DVDSUB_C - subgingial plaque
DVDRACR / DVDRACR_C - right retroauricular crease
DVDRACL / DVDRACL_C - left retroauricular crease
DVDACRR / DVDACRR_C - right antecuibtal fossa
DVDACRL / DVDACRL_C - left antecuibtal fossa
DVDNASAL / DVDNASAL_C - anterior nares
DVDINTRO / DVDINTRO_C - vaginal introitus
DVDPFORN / DVDPFORN_C - posterior fornix
DVDMIDVG / DVDMIDVG_C - mid vagina
```

In [ ]:

```
%%R
to.remove <- c("dbGaP.SubjID", "PROT", "PROTSEG", "SITE_C", "protseg", "VISNO",
                           "study_day", "DVDEDLVL_C", "DVDINSRH", "DVDINSRD", "
DVDOCPTN",
                           "DVDTOBC", "DVDPGRES", "DVDPGRES_C", "START", "GENDE
R", "ETHNIC",
                           "AMERIND", "AMERIND_C", "ASIAN", "PACIFC", "PACIFC_C
", "WHITE",
                           "BRTHCTRY", "MOMCTRY", "DADCTRY", "DSUSCRN", "DSUCON
", "BLACK",
                           "DSUDIET_C", "DSUBIR", "dsuotrpd", "dsuvspd", "DSUZI
P",
                           "DVDINTPH", "DVDPFPH", "DVDTMPF", "DVDTMPC", "AGEENR
", "DSUBFED",
                           "DVDBLOOD", "DVDBLOOD_C", "DVDSERUM", "DVDSERUM_C",
"DVDSTOOL",

                           "DVDSTOOL_C", "DVDSAL", "DVDSAL_C", "DVDGING", "DVDG
ING_C",
                           "DVDTONG", "DVDTONG_C", "DVDPTON", "DVDPTON_C", "DVD
HPAL",
                           "DVDHPAL_C", "DVDTHRO", "DVDTHRO_C ", "DVDBUCC", "DV
DBUCC_C",
                           "DVDSUPRA", "DVDSUPRA_C", "DVDSUB", "DVDSUB_C", "DVD
RACR",
                           "DVDRACR_C", "DVDRACL", "DVDRACL_C", "DVDACRR", "DVD
ACRR_C",
                           "DVDACRL", "DVDACRL_C", "DVDNASAL", "DVDNASAL_C", "D
VDVIRRT",
                           "DVDVIRRT_C", "DVDINTRO", "DVDINTRO_C", "DVDPFORN",
"DVDPFORN_C",
                           "DVDMIDVG", "DVDMIDVG_C", "DVDTHRO_C", "DSUSCRN_C",
"DSUCON_C")

categorical <- categorical[,-which(colnames(categorical) %in% to.remove)]

dim(categorical) #17 categories remain
summary(categorical)
```

So we're left with 17 variables at this point Looking at the output of the summary command we see that many of them are now binary variables:

> SITE - city of origin - Houston / StLouis
> GENDER_C - gender - Male / Female
> DVDTOBC_C - tobacco user - yes / no
> BLACK_C - yes / no
> WHITE_C - yes /no
> DSUBFED_C - Were you breastfed as a child? (don't know was pooled with NA)
> DVDINSRH_C - health insurance?
> DVDINSRD_C - dental insurance?

and others have numerous levels without much heft to any of them. Therefore we re-keyed the other variables to make them binary:

In [ ]:

```
%%R

#DVDEDLVL_C - education level - 8 different levels
#turn it into BS or higher
categorical$DVDEDLVL <- categorical$DVDEDLVL == 8 | categorical$DVDEDLVL == 9 |
 categorical$DVDEDLVL == 10 |
                        categorical$DVDEDLVL == 11


#DVDOCPTN_C - Occupation - 8 different levels
#turn it into Student or not
categorical$DVDOCPTN_C <- categorical$DVDOCPTN_C == "42 Student"


#BRTHCTRY_C, MOMCTRY_C, DADCTRY_C - birth country of subject and parents
#14 different levels - turn into whether or not "Canada/US"
categorical$BRTHCTRY_C <- categorical$BRTHCTRY_C == "Canada/US"
categorical$MOMCTRY_C <- categorical$MOMCTRY_C == "Canada/US"
categorical$DADCTRY_C <- categorical$DADCTRY_C == "Canada/US"


#DSUDIET - four levels of meat consumption - two with meat and two without
#turn it into meat or not
categorical$DSUDIET <- categorical$DSUDIET == 1 | categorical$DSUDIET == 2


#DSUBIR_C - whether subject had given birth never, once or more than once
#turn it into whether the subject had given birth
categorical$DSUBIR_C <- categorical$DSUBIR_C == "More than once" | categorical$
DSUBIR_C == "Once"


#ETHNIC_C - Hispanic / latino / spanish?
categorical$ETHNIC_C <- categorical$ETHNIC_C == "Hispanic or Latino or Spanish"

summary(categorical)
```

When we looked through the medication data we noticed that a number of people were chronically on medications. So we decided to make this a categorical varible. We found that the most commonly used medications were the following:

- 13=Contraceptives (oral, implant, injectable)(N=102)
- 21=Vitamins, minerals, food supplements (N=54)
- 9=Antidepressants/mood-altering drugs (N=37)
- 8=Antibiotics/(anti)-infectives, parasitics, microbials (N=33)
- 10=Antihistamines/Decongestants (N=28)

To get going, we want to get the start and stop dates for each subject's medication usage as well as the dates for each subject's visit:

```
In [ ]:
```

```
%%R

medication <- read.table(file="phs000228.v3.pht001187.v3.p1.c1.EMMES_HMP_DCM.HM
P.txt", header=T, sep="\t", na.string=c("", NA))

sort(summary(as.factor(medication$DCMCODE_C)))

#the most commonly used medications in the study...
#13=Contraceptives (oral, implant, injectable)                         102
#21=Vitamins, minerals, food supplements                               54
#9=Antidepressants/mood-altering drugs
        37
#8=Antibiotics/(anti)-infectives, parasitics, microbials    33
#10=Antihistamines/Decongestants
            28

#pull out thse five medication classes
medication <- medication[,c("RANDSID", "DCMCODE", "MSTART", "MSTOP")]
popular.user <- medication$DCMCODE == 13 | medication$DCMCODE == 21 | medicatio
n$DCMCODE == 9 |
                    medication$DCMCODE == 8 | medication$DCMCODE == 10
medication <- medication[popular.user,] #only want those lines from the file wi
th these medications
medication <- medication[order(medication$RANDSID),]   #make sure everything is
 in the right order


#figure out the study date that each person visited
metadata <- read.table(file="phs000228.v3.pht002157.v1.p1.c1.EMMES_HMP_DTP_DHX_
DVD.HMP.txt", header=T, sep="\t",
                       na.string=c("", NA))
normalvisit <- metadata$VISNO == "01" | metadata$VISNO == "02" | metadata$VISNO
 == "03"
metadata <- metadata[normalvisit,]  #only get those rows that corresponded to a
metadata <- metadata[normal.visit,]  #only get those rows that corresponded to a
 normal visit
subjects <- levels(as.factor(metadata$RANDSID))
subjects == sort(subjects)   #make sure everything is in the right order


#now we want to populate a table with the date for each person's visits
visit.dates <- matrix(rep(NA, length(subjects)*3), ncol=3) #start with everyone
 at NA
rownames(visit.dates) <- subjects #subject ids in the rows
colnames(visit.dates) <- c("Visit1", "Visit2", "Visit3")  #visits in the column
s


#loop through and fill in the study dates for each person's first, second, and
third visit
for(r in rownames(metadata)){
        if(metadata[r, "VISNO"] == "01"){
                visit.dates[as.character(metadata[r,"RANDSID"]), "Visit1"] = me
tadata[r, "study_day"]
        }
        else if(metadata[r, "VISNO"] == "02"){
                visit.dates[as.character(metadata[r,"RANDSID"]), "Visit2"] = me
tadata[r, "study_day"]
        }
        else if(metadata[r, "VISNO"] == "03"){
                visit.dates[as.character(metadata[r,"RANDSID"]), "Visit3"] = me
tadata[r, "study_day"]
        }
}
```

Note that at this point if someone didn't have a second or third visit, they will be listed as NA. Now we want to figure out whether a visit corresponded to the subjects being on the medications. We defined subjects that last took the medication within 30 days of a visit as "being on the medication" for that visit. Then we defined chronic usage as being on the medication for all of that subject's visits. With these criteria the time from taking the medication didn't have a big effect on the number of chronic users.

In [ ]:

```
%%R

#this function asks whether the subject was on the medication at their visit
getDrugVisitData <- function(drug.metadata){

        drug <- matrix(rep(NA, length(subjects)*3), ncol=3)
        colnames(drug) <- c("Visit1", "Visit2", "Visit3")
        rownames(drug) <- rownames(visit.dates)
        drug[!is.na(visit.dates)] <- FALSE  #if the subject visited, then we as
sume they're negative to start
        drug[is.na(visit.dates)] <- NA      #if they didn't visit, we don't car
```

```
e



        visitwindow <- 30       #the person discontinued the drug in the last 3
0 days

        for(r in rownames(drug.metadata)){
                s <- drug.metadata[as.character(r), "RANDSID"]
                start <- drug.metadata[as.character(r), "MSTART"]
                stop <- drug.metadata[as.character(r), "MSTOP"]
                if(is.na(stop)) stop = 100000   #if the stop date is NA then as
sume that they were on the medication through the end
                                        #of the study



        #these work like so... - subject is assumed false initially and if they
're subsequently changed to true, then we can't
        #change them to false. if the start date is before the visit date adn t
he stop date is earlier than the stop date plus
        #the visitwindow (30 days) then we consider them true. otherwise, keep
them as false.

                if(drug[as.character(s), "Visit1"] == FALSE & start <= visit.da
tes[as.character(s), "Visit1"] &
           stop+visitwindow >= visit.dates[as.character(s), "Visit1"] & !is.na(
visit.dates[as.character(s), "Visit1"])){
                        drug[as.character(s), "Visit1"] = TRUE
                } else if(drug[as.character(s), "Visit1"] == FALSE & !is.na(vis
it.dates[as.character(s), "Visit1"]))
                        drug[as.character(s), "Visit1"] = FALSE

                if(drug[as.character(s), "Visit2"] == FALSE & start <= visit.da
tes[as.character(s), "Visit2"] &
           stop+visitwindow >= visit.dates[as.character(s), "Visit2"] & !is.na(
visit.dates[as.character(s), "Visit2"])){
                        drug[as.character(s), "Visit2"] = TRUE
                } else if(drug[as.character(s), "Visit2"] == FALSE & !is.na(vis
it.dates[as.character(s), "Visit2"]))
                        drug[as.character(s), "Visit2"] = FALSE

                if(drug[as.character(s), "Visit3"] == FALSE & start <= visit.da
tes[as.character(s), "Visit3"] &
           stop+visitwindow >= visit.dates[as.character(s), "Visit3"] & !is.na(
visit.dates[as.character(s), "Visit3"])){
                        drug[as.character(s), "Visit3"] = TRUE
                } else if(drug[as.character(s), "Visit3"] == FALSE & !is.na(vis
it.dates[as.character(s), "Visit3"]))
                        drug[as.character(s), "Visit3"] = FALSE
        }

    #send back a table with visits as columns, subjects as rows, and values ind
icate whether the subject was on the medication
        return(drug)
```

```
}



#this function defines whether the subject was a chronic user or not
getChronic <- function(drug){
        chronic <- rep(FALSE, nrow(drug))

    #subject had to be on medication for all of their visits to be considered a
 chronic user
        chronic[((drug[,"Visit1"] == T & is.na(drug[,"Visit2"]) & is.na(drug[,"
Visit3"])) |
                         (drug[,"Visit1"] == T & drug[,"Visit2"] == T & is.na(d
rug[,"Visit3"])) |
                         (drug[,"Visit1"] == T & drug[,"Visit2"] == T & drug[,"
Visit3"] == T))] <- T

    #returns a vector indicating whether each subject was chronic
        return(chronic)

}


contra <- getDrugVisitData(medication[medication$DCMCODE == 13,])
vit_min <- getDrugVisitData(medication[medication$DCMCODE == 21,])
antidep <- getDrugVisitData(medication[medication$DCMCODE == 9,])
antibiot <- getDrugVisitData(medication[medication$DCMCODE == 8,])
antihist <- getDrugVisitData(medication[medication$DCMCODE == 10,])

contra.chronic <- getChronic(contra)
vit_min.chronic <- getChronic(vit_min)
antidep.chronic <- getChronic(antidep)
antihist.chronic <- getChronic(antihist)
antibiot.chronic <- getChronic(antibiot)
contra.chronic[categorical$GENDER_C=="Male"] <- NA


sum(contra.chronic, na.rm=T); sum(apply(contra, 1, sum, na.rm=T) > 0)  #72; 87
sum(vit_min.chronic); sum(apply(vit_min, 1, sum, na.rm=T) > 0)         #34; 44
sum(antidep.chronic); sum(apply(antidep, 1, sum, na.rm=T) > 0)         #22; 31
sum(antihist.chronic); sum(apply(antihist, 1, sum, na.rm=T) > 0)       #9; 22
sum(antibiot.chronic); sum(apply(antibiot, 1, sum, na.rm=T) > 0)       #2; 11
#not enough antibiotic.chronic subjects to do anything

#finally, append chronic contraception, vitamin/mineral, antidepressents, and a
ntihistamine users to categorical table
categorical <- cbind(categorical, contra.chronic, vit_min.chronic, antidep.chro
nic, antihist.chronic)
```

There's one more piece of categorical data we'd like to get - the subjects' BMI expressed as a categorical varaible.

```
In [ ]:
```

```
cont <- read.table(file="phs000228.v3.pht002157.v1.p1.c1.EMMES_HMP_DTP_DHX_DVD.
HMP.txt", header=T, sep="\t",
                   na.string=c("", NA))
cont <- cont[cont$VISNO == "00",]  #BMI was recorded based on the intake visit
rownames(cont) <- cont$RANDSID
cont <- cont[order(rownames(cont)),]

bmi <- cont[,"DTPBMI"] #this is the only variable we need from the table

BMI_C <- character()
BMI_C[bmi<18.5] <- "underweight"
BMI_C[bmi>=18.5 & bmi < 25] <- "normal"
BMI_C[bmi>=25 & bmi < 30] <- "overweight"
BMI_C[bmi>=30] <- "obese"
BMI_C <- factor(BMI_C)
rownames(cont) == rownames(categorical) #double check that our rows are in the
correct order


#let's append the BMI_C varaiable to the end of categorical
categorical <- cbind(categorical, BMI_C)
for(c in colnames(categorical)){        categorical[,c] <- factor(categorical[,
c])     }


#output the table to a file
write.table(categorical, "categorical.metadata", quote=F)
```

With that last command we have outputted the categorical data to a file. Let's move on to the continuous data. We found the following continuous data via dbGaP:

> DVDINTPH - pH - vaginal introitus
> DVDPFPH - pH - posterior fornix
> DTPPULSE - pulse
> DTPBMI - BMI
> DTPSYSTL - diastolic pressure
> DTPDIAST - systolic pressure

In [ ]:

```
#Get continuous data...
cont.1 <- read.table(file="phs000228.v3.pht002157.v1.p1.c1.EMMES_HMP_DTP_DHX_DV

D.HMP.txt", sep="\t",
                     header=T, na.string=c(NA, ""))

#some variables were taken at the intake appointment
continuous <- cont.1[cont.1$VISNO=="00",c("DTPPULSE","DTPBMI","DTPSYSTL","DTPD
IAST", "RANDSID")]
rownames(continuous) <- continuous$RANDSID

#define a continuous table variable
continuous <- continuous[order(rownames(continuous)),]




#vaginal pH was taken at each visit. we created a new file - vaginal_ph.txt - t
hat contains
#the two pH values for each woman at each visit:
cont.1$VISNO <- gsub("^(\\d\\d).*", "\\1", cont.1$VISNO)
pH <- cont.1[cont.1$VISNO == "01" | cont.1$VISNO == "02" | cont.1$VISNO == "03
",]
pH <- pH[,c("RANDSID", "VISNO", "SITE", "DVDINTPH", "DVDPFPH")]
pH$VISNO <- as.numeric(gsub("^0", "", pH$VISNO))
pH <- pH[!is.na(pH$DVDINTPH) | !is.na(pH$DVDPFPH),]
write.table(pH, file="vaginal_ph.txt", quote=F, row.names=F, sep="\t")



#let's add the subjects' ages
cont.2 <- read.table(file="phs000228.v3.pht002158.v1.p1.c1.EMMES_HMP_DEM_ENR.HM
P.txt", sep="\t", header=T, na.string=c(NA, ""))
rownames(cont.2) <- cont.2$RANDSID
cont.2 <- cont.2[order(rownames(cont.2)),]
rownames(cont.2) == rownames(continuous)  #make sure the rows are in the right
order
#The following was the only continuous data avaialble in this metadata file
#AGEENR - Age in years
AGEENR <- cont.2$AGEENR
continuous <- cbind(continuous, AGEENR)

#remove the RANDSID column from the table
continuous <- continuous[,-which(colnames(continuous) %in% c("RANDSID"))]

#write continuous data out to a file
write.table(continuous, "continuous.metadata", quote=F)
```

At this point all of the categorical and continuous data have been identified and stored to a file. We have enverything that was used to generate Extended Data Table 1:

In [ ]:

```
summary(categorical)

#count bfed "not remember"
bfed <- categ.3$DSUBFED_C
bfed[bfed == ""] <- NA
summary(factor(bfed))

#count transients
sum(apply(contra, 1, sum, na.rm=T) > 0)-sum(contra.chronic, na.rm=T);
sum(apply(vit_min, 1, sum, na.rm=T) > 0)-sum(vit_min.chronic, na.rm=T);
sum(apply(antidep, 1, sum, na.rm=T) > 0)-sum(antidep.chronic, na.rm=T);
sum(apply(antihist, 1, sum, na.rm=T) > 0)-sum(antihist.chronic, na.rm=T);

summary(continuous)

pH.summary <- pH[,c("DVDINTPH", "DVDPFPH")]
apply(pH.summary, 2, summary)
```

Let's copy these metadata files to our analysis folder and head over there to do our analysis.

In [ ]:

```
%%bash
cp categorical.metadata continuous.metadata vaginal_ph.txt ../analysis/
cd ../analysis/
```

Analysis

Now we have the sequence data and clinical metadata curated and into a form that we can handle. Let's do some analysis. We have several tasks ahead of us...

- Describe community types
- Identify associations between body sites and clinical metadata
- Identify associations between body sites
- Quantify rate of change between community types for each body site
- Effect of time on stability of community types
- Validate DMM models as a community type approach relative to the PAM-based approach

Describe community types

First we wanted to inspect the DMM model fits and see the top taxa in each community type. This analysis was performed using the data found in the dmm_best folder. Let's generate the types of plots shown in Figure 1A and 1B. First we'll make the line plot of the Laplace fit metric vs. the number of community types (e.g. Figure 1A). For fun we'll bold and color in red the x-axis label that corresponds to the minimum Laplace metric. These will be outputted as pdfs:

In [ ]:

```
%%R
dmm_best_path <- "../v35/dmm_best/"
fit.files <- c("Antecubital_fossa.tx.1.subsample.1.dmm.mix.fit", "Anterior_nare
s.tx.1.subsample.1.dmm.mix.fit",
                                "Buccal_mucosa.tx.1.subsample.1.dmm.mix.fit", "
Hard_palate.tx.1.subsample.1.dmm.mix.fit",
                                "Keratinized_gingiva.tx.1.subsample.1.dmm.mix.f
it", "Palatine_Tonsils.tx.1.subsample.1.dmm.mix.fit",
                                "Retroauricular_crease.tx.1.subsample.1.dmm.mix
.fit", "Saliva.tx.1.subsample.1.dmm.mix.fit",
                                "Stool.tx.1.subsample.1.dmm.mix.fit", "Subgingi
val_plaque.tx.1.subsample.1.dmm.mix.fit",
                                "Supragingival_plaque.tx.1.subsample.1.dmm.mix.
fit", "Throat.tx.1.subsample.1.dmm.mix.fit",
                                "Tongue_dorsum.tx.1.subsample.1.dmm.mix.fit", "
Vagina.tx.1.subsample.1.dmm.mix.fit")

for(bs in fit.files){
        pdf.file <- paste(bs, ".pdf", sep="")
        fit <- read.table(file=paste(dmm_best_path, bs, sep=""), header=T)

        pdf(file=pdf.file)

        bs <- gsub("([^.]*).*", "\\1", bs)
        bs <- gsub("_", " ", bs)

        par(mar=c(5, 5, 0.5, 0.5))

        plot(fit$Laplace~fit$K, xlab="Number of Dirichlet\nComponents", ylab=pa
ste("Model Fit\n(",bs, ")", sep=""), type="l")
        points(fit$Laplace~fit$K, pch=19)

        axis(1, at=which.min(fit$Laplace), label=which.min(fit$Laplace), col.ax
is="red", font=2)

        dev.off()
}
```

Now let's make the box plots that show the relative abundance for the top 5 genera in each community type for each body site (e.g. Figure 1B)

In [ ]:

```
%%R

dmm_best_path <- "../v35/dmm_best/"

sites <- c("Antecubital_fossa.tx.1.subsample.1.dmm.mix.design", "Anterior_nares
.tx.1.subsample.1.dmm.mix.design",
                "Buccal_mucosa.tx.1.subsample.1.dmm.mix.design", "Hard_palate.t
x.1.subsample.1.dmm.mix.design",
                "Keratinized_gingiva.tx.1.subsample.1.dmm.mix.design", "Palatin
e_Tonsils.tx.1.subsample.1.dmm.mix.design",
                "Retroauricular_crease.tx.1.subsample.1.dmm.mix.design", "Saliv
a.tx.1.subsample.1.dmm.mix.design",
                "Stool.tx.1.subsample.1.dmm.mix.design", "Subgingival_plaque.tx
.1.subsample.1.dmm.mix.design",
                "Supragingival_plaque.tx.1.subsample.1.dmm.mix.design", "Throat
.tx.1.subsample.1.dmm.mix.design",
                "Tongue_dorsum.tx.1.subsample.1.dmm.mix.design", "Vagina.tx.1.s
ubsample.1.dmm.mix.design")

#need to get the last level that was given a classification for each phylotype
taxonomy <- read.table(file="../v35/v35.unique.good.filter.unique.precluster.un
ique.pick.three.wang.pick.tx.1.cons.taxonomy",
                       header=T, row.names=1)
taxonomy$Taxonomy <- gsub("\\(\\d*\\)", "", taxonomy$Taxonomy)
taxonomy$Taxonomy <- gsub("unclassified;", "", taxonomy$Taxonomy)
taxonomy$Taxonomy <- gsub(";$", "", taxonomy$Taxonomy)
taxonomy$Taxonomy <- gsub(".*;", "", taxonomy$Taxonomy)
taxonomy$Taxonomy <- gsub('"', "", taxonomy$Taxonomy)

for(bs in sites){
        design.fname <- bs #need the design file to know which sample belongs t
o each community type
        design <- read.table(file=paste(dmm_best_path, design.fname, sep=""))

        shared.fname <- gsub("1.dmm.mix.design", "shared", bs) #need the relati
ve abundance data to know
        shared <- read.table(file=paste("../v35/", shared.fname, sep=""), heade
r=T, row.names=2)#the abundance of each
        shared <- shared[,-c(1,2)]                              #OTU in each sample

    summary.fname <- gsub("design", "summary", bs) #need the summary file to kn
ow the top 5 OTUs
        summary <- read.table(file=paste(dmm_best_path, summary.fname, sep=""),
 header=T, row.names=1)#for defining each commmunity type

    site <- gsub("^([^.]*).*", "\\1", design.fname)
        site <- gsub("_", " ", site)

        ntypes <- length(levels(design$V2)) #get the number of community types
        top.otus <- summary[1:5,]
        otu.ids <- rownames(top.otus) #get the names of the top 5 phylotypes
        otu.tax <- taxonomy[otu.ids,"Taxonomy"] #get their taxonomic affiliatio
n

        otu.shared <- shared[,otu.ids]/1000     #get the relative abundance
        rownames(otu.shared) == design$V1       #check they're in the right ord
er

        pdf(file=paste(summary.fname, ".pdf", sep=""))  #let's save these as pd
```

```
is
        par(mar=c(10, 4, 0.5, 0.5))

        plot(x=c(0,5*(ntypes+2)), y=c(0,1), type="n", axes=F, xlab="", ylab="Re
lative abundance (%)")

        width = 0.5
        left <- 1
        clrs <- rainbow(ntypes)

        for(i in otu.ids){
        #calculate the intra quartile ranges
                iqr <- aggregate(otu.shared[,i], by=list(design$V2), quantile,
probs=c(0.025, 0.25, 0.50, 0.75, 0.975))

        #error bars represent the 95% confindence interval
                arrows(x0=left:(left+ntypes-1), x1=left:(left+ntypes-1), y0=iqr
$x[,"50%"], y1=iqr$x[,"97.5%"], angle=90,
               length=0.05, lwd=2)
                arrows(x0=left:(left+ntypes-1), x1=left:(left+ntypes-1), y0=iqr
$x[,"50%"], y1=iqr$x[,"2.5%"], angle=90,
               length=0.05, lwd=2)

        #rectangles represent the 50% confidence interval with a line in the mi
ddle for the median
        rect(xleft=(left:(left+ntypes-1))-width, xright=(left:(left+ntypes-1))+
width, ytop=iqr$x[,"75%"],
             ybottom=iqr$x[,"50%"], col=clrs, lwd=3)
                rect(xleft=(left:(left+ntypes-1))-width, xright=(left:(left+nty
pes-1))+width, ytop=iqr$x[,"25%"],
             ybottom=iqr$x[,"50%"], col=clrs, lwd=3)

        #bounce down the x-axis for the next phylotype
                left <- left + ntypes + 2
        }

        axis(2, label=seq(0,100,20), at=seq(0,1,0.2), las=2)
        start <- (ntypes+1) / 2
        increment <- ntypes + 2

    #label the phylotypes on the x-axis
        axis(1, label=otu.tax, at=c(start, start+increment, start+2*increment,
start+3*increment, start+4*increment),
         cex=0.5, las=2)
        box()

    #get a pretty legend
        legend("topright", legend=paste("Community Type", LETTERS[1:ntypes]), f
ill=clrs, pt.cex=1.5, cex=1, bty="n")
        mtext(1, line=8.5, text=site, font=2)  #tell us what the body site was
        dev.off()
}
#expect some warning messages: these indicate that the median for some phylotyp
es was zero and so
#the bottom rectangle for the IQR is zero, whatever
```

```
#the bottom rectangle for the IQR is zero. whatever.
```

Identify associations between body sites and clinical metadata

Categorical metadata

In attempting to deal with the community type data we have a small difficulty that we have up to 3 community type assignments for each person at each body site but the clinical metadata did not change. Our strategy was to use an iterative procedure. Our basic method for each body site was the following. First, we randomly picked one visit from each subject and used Fisher's exact test to test for an association with each of the clinical variables. We corrected for multiple comparisons using the Benjamini-Hochberg algorithm. Second, we again randomly selected one visit from each subject and tested. We did a total of 1000 iterations and quantified the median P-value and the fraction of iterations that yielded a significant test. The output from this section is discussed in the manuscript and provdied in SourceDataFigure3.xlsx

In [ ]:

```
%%R
totalIters <- 1000

#Set up the community type data - recall we generated these above to describe t
he community type
#each subject at each visit
v1<-read.table(file="community_types.v1.txt", header=T, row.names=1)
v2<-read.table(file="community_types.v2.txt", header=T, row.names=1)
v3<-read.table(file="community_types.v3.txt", header=T, row.names=1)



#get the collection of the ids for all 300 subjects
randsids <- union(rownames(v1), union(rownames(v2), rownames(v3)))



#Set up the categorical data
categ <- read.table(file="categorical.metadata", header=T)
categ.good <- categ[randsids,]  #only want those metadata that we have microbio
me data for

variables <- colnames(categ.good)
bodysites <- colnames(v1)

#this table will hold the fraction of iterations that a bodysite had a signific
ant test
sig.rate <- matrix(rep(0, length(variables)*length(bodysites)), nrow=length(var
iables))
```

```
rownames(sig.rate) <- variables
colnames(sig.rate) <- bodysites



#this table will hold the median P-value across the iterations for each bodysit
e
medians <- matrix(rep(0, length(variables)*length(bodysites)), nrow=length(vari
ables))
rownames(medians) <- variables
colnames(medians) <- bodysites

for(i in bodysites){
        print(i)
        times.sig <- rep(0, length(variables))  #set up the counting vector for
 the number of sig iters
        names(times.sig) <- variables
        total <- 0

        sig <- rep(NA, length(variables))       #
        names(sig) <- variables

    #the record of the community type for each subject (rows) at each visit (co
lumn)
        visits <- data.frame(matrix(rep(NA, length(randsids)*3), ncol=3))
        colnames(visits) <- c("v1", "v2", "v3")
        rownames(visits) <- randsids

        visits[rownames(v1), "v1"] <- as.character(v1[,i])
        visits[rownames(v2), "v2"] <- as.character(v2[,i])
        visits[rownames(v3), "v3"] <- as.character(v3[,i])



    #set up the table of p-values for each body site and iteration
        median.iter <- matrix(rep(NA, length(variables)*totalIters), nrow=lengt
h(variables))
        rownames(median.iter) <- variables
        colnames(median.iter) <- 1:totalIters

        for(iter in 1:totalIters){  #start looping over the iterations
        iter <- 1
                type <- as.character()
                for(j in randsids){     #for each subject, randomly select the
community type at one visit
                        type[j] <- sample(na.omit(as.character(visits[j,])))[1]
 #omits those visits that didn't happen
                }

                for(j in variables){  #now loop through the categorical variabl
es
            contingency <- table(type, categ.good[,j]) #make a contingency tabl
e

                        if(sum(apply(contingency, 2, sum)>10) > 1){ #if either
value of the metadata was seen less than 10 times, move on
value of the metadata was seen less than 10 times, move on
                                if(ncol(contingency)>2 | nrow(contingency) > 2)
{       #if we have more than 2 communitytypes or variable levels
                                                                    #we'll simulate
 the p-values
                                        sig[j] <- fisher.test(contingency, simu
late.p.value=T, B=1e5)$p.value
                                } else{
                                        sig[j] <- fisher.test(contingency)$p.va
lue
                                }
                        } else{
                                sig[j] <- NA
                        }
                }

                adjusted <- p.adjust(sig, method="BH")
                correct <- adjusted < 0.05

                times.sig <- correct + times.sig
                total <- total + 1
                median.iter[, iter] <- sig
        }
        sig.rate[,i] <- times.sig / total       #calculate the fraction of sign
ificant tests
        medians[,i] <- apply(median.iter, 1, median)    #calculate the median p
-value
}

write.table(sig.rate, file="categorical.sig.rate", quote=F, sep="\t")
write.table(medians, file="categorical.medians", quote=F, sep="\t")

interesting <- apply(sig.rate, 1, max, na.rm=T) > 0.5  #cut through the data to
 see which data had any signficance
sig.rate[interesting,]
```

In the following cell we see that there are 9 bodysite/metadata combinations that were significant in more than 75% of the iterations. We inadvertantly left out the observation that there is a significant association between Asians and their community type. We go ahead and make bar plots representing the contingency tables (e.g. Figure 1C and D and Extended Figures 2 and 3)

In [ ]:

```
threshold <- 0.75   #what's the minimum fraction of iterations that need to be
significant
totalIters <- 100   #number of iterations to calculate the average contingency
table for
                    #we dont need a lot of precision here
```

```
#we read back in the categorical significance rates but will ignore the SITE da
ta becuase

#of the confounding described above and chronic Antihistamine usage since all t
he tests
#resuled in NAs.
sig.rate <- read.table(file="categorical.sig.rate", header=T)
sig.rate <- sig.rate[-which(rownames(sig.rate) %in% c("SITE", "antihist.chronic
")),]

#Let's get the most frequently significant bodysites and metadata
row.max <- apply(sig.rate, 1, max, na.rm=T) > threshold
col.max <- apply(sig.rate, 2, max, na.rm=T) > threshold
most.sig <- sig.rate[row.max, col.max]
#there were 8 combinations


#we're essentially going to recreate the radonomized contingency tables like we
 did to
#run the Fisher's exact test
v1<-read.table(file="community_types.v1.txt", header=T, row.names=1)
v2<-read.table(file="community_types.v2.txt", header=T, row.names=1)
v3<-read.table(file="community_types.v3.txt", header=T, row.names=1)
good.ids <- unique(c(rownames(v1), rownames(v2), rownames(v3)))

categ <- read.table(file="categorical.metadata", header=T)
categ.good <- categ[good.ids,]

for(md in rownames(most.sig)){      #scan over the metadata in most.sig
        for(bs in colnames(most.sig)){  #and scan over the bodysites in most.si
g

                if(!is.na(sig.rate[md, bs] ) & sig.rate[md, bs] > threshold){#i
f it isn't an NA and
                                                                        #is over o
ur threshold...
            #let's set up a visits matrix like before
            visits <- data.frame(matrix(rep(NA, length(good.ids)*3), ncol=3))
                        colnames(visits) <- c("v1", "v2", "v3")
                        rownames(visits) <- good.ids
                        visits[rownames(v1), "v1"] <- as.character(v1[,bs])
                        visits[rownames(v2), "v2"] <- as.character(v2[,bs])
                        visits[rownames(v3), "v3"] <- as.character(v3[,bs])

            #get the possible community types that were observed at the bodysit
e
                        types <- levels(factor(c(visits[,1], visits[,2], visits
[,3])))
                        ntypes <- length(types)

            #get the possible values for the metadata
                        levels <- levels(as.factor(categ.good[,md]))
                        nlevels <- length(levels)

            #initialize the contingency table
            #initialize the contingency table
                        contingency <- as.table(matrix(rep(0, ntypes*nlevels),
nrow=ntypes))
                        rownames(contingency) <- types
                        colnames(contingency) <- levels

            #let it rip...
                        for(iter in 1:totalIters){
                                type <- as.character()
                                for(k in good.ids){     #four nested for loops!
                                        type[k] <- sample(na.omit(as.character(
visits[k,])))[1]
                                }
                                type <- factor(type)
                                contingency <- contingency + table(type, categ.
good[,md])
                        }

            #add up the contingency tables
                        contingency <- contingency / totalIters    #get the pro
portions of each community type
                        p.contingency <- prop.table(contingency, 2)#in each met
adata class

            #make a pretty barplot and save it as a pdf
                        pdf(file=paste(bs, ".", md, ".pdf", sep=""))
                        par(mar=c(6,5, 0.5,0.5))
                        barplot(as.matrix(t(p.contingency)), beside=T, yaxt="n"
, names=LETTERS[1:nrow(p.contingency)],
                    legend.text=colnames(p.contingency), args.legend=c(cex=1.2)
, ylim=c(0,max(p.contingency)+0.05))

                        axis(2, at=seq(0,max(p.contingency+0.05),0.10), label=s
eq(0,max(p.contingency+0.05)*100, 10), las=1)

                        mtext(side=1, line=2.5, text=paste(gsub("_", " ", bs),
"Community Type"), cex=1.2)
                        mtext(side=2, line=2.5, text="Percentage of Samples", c
ex=1.2)
                        mtext(side=1, line=4, text=md)
                        box()
                        dev.off()
                }

        }
}
```

In [ ]:

```
totalIters <- 1000

#Set up the community type data
v1<-read.table(file="community types.v1.txt", header=T, row.names=1)
```

```
v2<-read.table(file="community_types.v2.txt", header=T, row.names=1)
v3<-read.table(file="community_types.v3.txt", header=T, row.names=1)

randsids <- union(rownames(v1), union(rownames(v2), rownames(v3)))


#Set up the continuous data
cont <- read.table(file="continuous.metadata", header=T, na.string=c("", NA))

#recall that we need to bring in the pH values from each visit - let's just ini
tialize it for now
cont <- cbind(cont, "DVDINTPH"=rep(NA, nrow(cont)), "DVDPFPH"=rep(NA, nrow(cont
)))

cont.good <- cont[randsids,]
variables <- colnames(cont.good)[2:ncol(cont.good)]
bodysites <- colnames(v1)

#now we'll get that pH data
ph.file <- read.table(file="vaginal_ph.txt", header=T)

#get the subject ids that we have pH data for
v1.names <- as.character(ph.file[ph.file$VISNO==1,1])
v2.names <- as.character(ph.file[ph.file$VISNO==2,1])
v3.names <- as.character(ph.file[ph.file$VISNO==3,1])
ph.names <- union(v1.names, union(v2.names, v3.names)) #get the unique list

#let's make a table of vaginal introitus pH values for each woman at each visit
ph.int <- data.frame("visit1"=rep(NA, length(ph.names)), "visit2"=rep(NA, lengt
h(ph.names)),
                     "visit3"=rep(NA, length(ph.names)), row.names=ph.names)
ph.int[v1.names, "visit1"] <- ph.file[ph.file$VISNO==1, "DVDINTPH"]
ph.int[v2.names, "visit2"] <- ph.file[ph.file$VISNO==2, "DVDINTPH"]
ph.int[v3.names, "visit3"] <- ph.file[ph.file$VISNO==3, "DVDINTPH"]

#let's make a table of posterior fornix pH values for each woman at each visit
ph.pf <- data.frame("visit1"=rep(NA, length(ph.names)), "visit2"=rep(NA, length
(ph.names)),
                    "visit3"=rep(NA, length(ph.names)), row.names=ph.names)
ph.pf[v1.names, "visit1"] <- ph.file[ph.file$VISNO==1, "DVDPFPH"]
ph.pf[v2.names, "visit2"] <- ph.file[ph.file$VISNO==2, "DVDPFPH"]
ph.pf[v3.names, "visit3"] <- ph.file[ph.file$VISNO==3, "DVDPFPH"]


#set up a table with the frequency of significant tests for each variable / bod
ysite combinatino
sig.rate <- matrix(rep(0, (length(variables))*length(bodysites)), nrow=(length(
variables)))
rownames(sig.rate) <- c(variables)
colnames(sig.rate) <- bodysites

#set up a table with the median p-value for each variable / bodysite combinatin
o
medians <- matrix(rep(0, (length(variables))*length(bodysites)), nrow=(length(v
medians <- matrix(rep(0, (length(variables))*length(bodysites)), nrow=(length(v
ariables)))

rownames(medians) <- c(variables)
colnames(medians) <- bodysites


for(i in bodysites){
        print(i)
        times.sig <- rep(0, length(variables))
        names(times.sig) <- variables
        total <- 0

        sig <- rep(NA, length(variables))
        names(sig) <- variables

        visits <- data.frame(matrix(rep(NA, length(randsids)*3), ncol=3))
        colnames(visits) <- c("v1", "v2", "v3")
        rownames(visits) <- randsids

        visits[rownames(v1), "v1"] <- as.character(v1[,i])
        visits[rownames(v2), "v2"] <- as.character(v2[,i])
        visits[rownames(v3), "v3"] <- as.character(v3[,i])

        median.iter <- matrix(rep(NA, length(variables)*totalIters), nrow=lengt
h(variables))
        rownames(median.iter) <- variables

        for(iter in 1:totalIters){
                type <- character()

                for(j in randsids){
                        x<-1:3

                        rand.vis <- sample(x)
                        if(!is.na(visits[j,rand.vis[1]])){
                                rand.vis <- rand.vis[1]
                        } else if(!is.na(visits[j,rand.vis[2]])){
                                rand.vis <- rand.vis[2]
                        } else {
                                rand.vis <- rand.vis[3]
                        }
                        type[j] <- visits[j,rand.vis]
                        cont.good[j, "DVDINTPH"] <- ph.int[j, rand.vis]#fill in
 the pH values for that visit
                        cont.good[j, "DVDPFPH"] <- ph.pf[j, rand.vis]
                }

                for(j in variables){  #do the anova
                       sig[j] <- anova(lm(cont.good[,j]~factor(type)))$Pr[1]
                }

                adjusted <- p.adjust(sig, method="BH") #correct for multiple co
mparisons
                correct <- adjusted < 0.05  #significant?
                correct <- adjusted < 0.05  #significant?


                times.sig <- correct + times.sig #add 1 if significant
                total <- total + 1
                median.iter[, iter] <- sig #keep track of p-values
        }
        sig.rate[,i] <- times.sig / total #get rate of significance
        medians[,i] <- apply(median.iter, 1, median) #get median p-values
}

write.table(sig.rate, file="continuous.sig.rate", quote=F, sep="\t")
write.table(medians, file="continuous.medians", quote=F, sep="\t")

interesting <- apply(sig.rate, 1, max, na.rm=T) > 0.5  #cut through the data to
 see which data had any signficance
sig.rate[interesting,]  #maybe not all that interesting afterall
```

There were weak associations between vaginal pH and the vaginal community types

Continuous metadata

Now, let's use a similar procedure to look at the coninuous data. Instead of using Fisher's exact test, we'll use ANOVA and we'll use the pH values that were obtained from that day's sampling.

In [ ]:

```
rm(list=ls())
#set up the community type data
#first we get the subject ids
categ <- read.table(file="categorical.metadata", header=T, row.names=1)
randsids <- rownames(categ)

#next we read in the community type assignments
v1<-read.table(file="community_types.v1.txt", header=T, row.names=1)
v2<-read.table(file="community_types.v2.txt", header=T, row.names=1)
v3<-read.table(file="community_types.v3.txt", header=T, row.names=1)

#now we build the visit tables for the three vaginal sites - intialize
vi.types <- matrix(rep(NA, length(randsids)*3), nrow=length(randsids))
rownames(vi.types) <- randsids
colnames(vi.types) <- c("visit1", "visit2", "visit3")
mv.types <- vi.types
pf.types <- vi.types
pf.ph <- vi.types
int.ph <- vi.types

#now we build the visit tables for the three vaginal sites - fill
```

```
#now we build the visit tables for the three vaginal sites - fill
vi.types[rownames(v1), "visit1"] <- as.character(v1[,"Vaginal_introitus"])
vi.types[rownames(v2), "visit2"] <- as.character(v2[,"Vaginal_introitus"])
vi.types[rownames(v3), "visit3"] <- as.character(v3[,"Vaginal_introitus"])

mv.types[rownames(v1), "visit1"] <- as.character(v1[,"Mid_vagina"])
mv.types[rownames(v2), "visit2"] <- as.character(v2[,"Mid_vagina"])
mv.types[rownames(v3), "visit3"] <- as.character(v3[,"Mid_vagina"])

pf.types[rownames(v1), "visit1"] <- as.character(v1[,"Posterior_fornix"])
pf.types[rownames(v2), "visit2"] <- as.character(v2[,"Posterior_fornix"])
pf.types[rownames(v3), "visit3"] <- as.character(v3[,"Posterior_fornix"])


#now we'll get that pH data
ph.file <- read.table(file="vaginal_ph.txt", header=T)
ph1 <- ph.file[ph.file$VISNO==1, c("RANDSID", "DVDINTPH", "DVDPFPH")]
ph2 <- ph.file[ph.file$VISNO==2, c("RANDSID", "DVDINTPH", "DVDPFPH")]
ph3 <- ph.file[ph.file$VISNO==3, c("RANDSID", "DVDINTPH", "DVDPFPH")]

rownames(ph1) <- ph1$RANDSID; ph1 <- ph1[,-1]
rownames(ph2) <- ph2$RANDSID; ph2 <- ph2[,-1]
rownames(ph3) <- ph3$RANDSID; ph3 <- ph3[,-1]

#introitus pH
int.ph[rownames(ph1),"visit1"] <- ph1[, "DVDINTPH"]
int.ph[rownames(ph2),"visit2"] <- ph2[, "DVDINTPH"]
int.ph[rownames(ph3),"visit3"] <- ph3[, "DVDINTPH"]

#posterior pH
pf.ph[rownames(ph1),"visit1"] <- ph1[, "DVDPFPH"]
pf.ph[rownames(ph2),"visit2"] <- ph2[, "DVDPFPH"]
pf.ph[rownames(ph3),"visit3"] <- ph3[, "DVDPFPH"]

#concatenate the commmunity type data from the three visits into a single vecto
r
vi <- c(vi.types[,"visit1"], vi.types[,"visit2"], vi.types[,"visit3"])
mv <- c(mv.types[,"visit1"], mv.types[,"visit2"], mv.types[,"visit3"])
pf <- c(pf.types[,"visit1"], pf.types[,"visit2"], pf.types[,"visit3"])

#concatenate the pH data from the three visits into a single vector
intph <- c(int.ph[,"visit1"], int.ph[,"visit2"], int.ph[,"visit3"])
pfph <- c(pf.ph[,"visit1"], pf.ph[,"visit2"], pf.ph[,"visit3"])

#build the stripcharts...
pdf(file="Vagina_introitus.DVDINTPH.pdf")
        par(mar=c(5,4,0.5,0.5))
        good <- !is.na(vi) & !is.na(intph)
        stripchart(intph[good]~factor(vi[good]), pch=19, vertical=T, method="ji
tter", xlab="", ylab="",
                               ylim=c(3,7), yaxt="n", xaxt="n", col="red")
        title(ylab="Vaginal Introitus pH")
        mtext(1, line=3, text="Vagina Introitus Community Type")
        ave <- aggregate(intph[good], by=list(factor(vi[good])), mean)
        segments(x0=seq(0.8,4.8,1), x1=seq(1.2,5.2,1), y0=ave$x, lwd=4)
        segments(x0=seq(0.8,4.8,1), x1=seq(1.2,5.2, 1), y0=ave$x, lwd=4)
        axis(1, at=1:5, label=paste(LETTERS[1:5]))

        axis(2, las=2)
dev.off()

pdf(file="Vagina_introitus.DVDPFPH.pdf")
        par(mar=c(5,4,0.5,0.5))
        good <- !is.na(vi) & !is.na(pfph)
        stripchart(pfph[good]~factor(vi[good]), pch=19, vertical=T, method="jit
ter", xlab="", ylab="",
                               ylim=c(3,7), yaxt="n", xaxt="n", col="red")
        title(ylab="Posterior Fornix pH")
        mtext(1, line=3, text="Vagina Introitus Community Type")
        ave <- aggregate(pfph[good], by=list(factor(vi[good])), mean)
        segments(x0=seq(0.8,4.8,1), x1=seq(1.2,5.2, 1), y0=ave$x, lwd=4)
        axis(1, at=1:5, label=paste(LETTERS[1:5]))
        axis(2, las=2)
dev.off()


pdf(file="Mid_vagina.DVDINTPH.pdf")
        par(mar=c(5,4,0.5,0.5))
        good <- !is.na(mv) & !is.na(intph)
        stripchart(intph[good]~factor(mv[good]), pch=19, vertical=T, method="ji
tter", xlab="", ylab="",
                               ylim=c(3,7), yaxt="n", xaxt="n", col="red")
        title(ylab="Vaginal Introitus pH")
        mtext(1, line=3, text="Mid Vagina Community Type")
        ave <- aggregate(intph[good], by=list(factor(mv[good])), mean)
        segments(x0=seq(0.8,4.8,1), x1=seq(1.2,5.2, 1), y0=ave$x, lwd=4)
        axis(1, at=1:5, label=paste(LETTERS[1:5]))
        axis(2, las=2)
dev.off()

pdf(file="Mid_vagina.DVDPFPH.pdf")
        par(mar=c(5,4,0.5,0.5))
        good <- !is.na(mv) & !is.na(pfph)
        stripchart(pfph[good]~factor(mv[good]), pch=19, vertical=T, method="jit
ter", xlab="", ylab="",
                               ylim=c(3,7), yaxt="n", xaxt="n", col="red")
        title(ylab="Posterior Fornix pH")
        mtext(1, line=3, text="Mid Vagina Community Type")
        ave <- aggregate(pfph[good], by=list(factor(mv[good])), mean)
        segments(x0=seq(0.8,4.8,1), x1=seq(1.2,5.2, 1), y0=ave$x, lwd=4)
        axis(1, at=1:5, label=paste(LETTERS[1:5]))
        axis(2, las=2)
dev.off()


pdf(file="Posterior_fornix.DVDINTPH.pdf")
        par(mar=c(5,4,0.5,0.5))
        good <- !is.na(pf) & !is.na(intph)
        stripchart(intph[good]~factor(pf[good]), pch=19, vertical=T, method="ji
tter", xlab="", ylab="",
tter", xlab="", ylab="",
                              ylim=c(3,7), yaxt="n", xaxt="n", col="red")

        title(ylab="Vaginal Introitus pH")
        mtext(1, line=3, text="Posterior Fornix Community Type")
        ave <- aggregate(intph[good], by=list(factor(pf[good])), mean)
        segments(x0=seq(0.8,4.8,1), x1=seq(1.2,5.2, 1), y0=ave$x, lwd=4)
        axis(1, at=1:5, label=paste(LETTERS[1:5]))
        axis(2, las=2)
dev.off()

pdf(file="Posterior_fornix.DVDPFPH.pdf")
        par(mar=c(5,4,0.5,0.5))
        good <- !is.na(pf) & !is.na(pfph)
        stripchart(pfph[good]~factor(pf[good]), pch=19, vertical=T, method="jit
ter", xlab="", ylab="",
                              ylim=c(3,7), yaxt="n", xaxt="n", col="red")
        title(ylab="Posterior Fornix pH")
        mtext(1, line=3, text="Posterior Fornix Community Type")
        ave <- aggregate(pfph[good], by=list(factor(pf[good])), mean)
        segments(x0=seq(0.8,4.8,1), x1=seq(1.2,5.2, 1), y0=ave$x, lwd=4)
        axis(1, at=1:5, label=paste(LETTERS[1:5]))
        axis(2, las=2)
dev.off()
```

The SITE variable

The variable that was most frequently detected as being sigificantly associated with the various community types was SITE - the city that the subject was recruited from - Houston and St. Louis. This was also seen in the original Nature paper from 2012. As described way back at the beginning, the SITE varaible was perfectly confounded with where the DNA extraction, PCR and sequencing were done. Although it's entirely possible that people in Houston and St Louis have very different microbiomes, we wouldn't want to rest on this dataset for that comparison. Let's see whether any of our other metadata are associated with the SITE variable. Let's start with the categorical data.

In [ ]:

```
categ <- read.table(file="categorical.metadata", header=T)
site <- categ$SITE
categ <- categ[,-1]

p.values <- rep(0, ncol(categ))
names(p.values) <- colnames(categ)

for(md in colnames(categ)){
        p.values[md] <- fisher.test(site, categ[,md])$p.value
}

p.adjust(sort(p.values), method="BH")
p.adjust(sort(p.values), method="BH") < 0.05
```

What we glean from this is that their mom and dad's country of origin, whether they were Asian, white, or hispanic, whether they chronically use vitamins and mineral supplements, whether they had dental insurance, and their education level were significantly associated with the city of origin. This may cause us to not over interpret the result relating the vaginal community type to education level; however, considering that association was among the strongest observed, it is hard to discount.

Now we want to turn to the continuous metadata:

In [ ]:

```
cont <- read.table(file="continuous.metadata", header=T)
rownames(cont) == rownames(categ)

p.values <- rep(0, ncol(cont))
names(p.values) <- colnames(cont)

for(md in colnames(cont)){
        p.values[md] <- anova(lm(cont[,md]~factor(site)))$Pr[1]
}

p.adjust(p.values, method="BH")
p.adjust(p.values, method="BH") < 0.05
```

For some reason the diastolic blood pressure was significantly different between both cities. None of these variable were significantly associated with our community types so we'll move on.

Identify associations between body sites

Next we wanted to know whether the community type at one bodysite was more likely to be associated with the community type at a different bodysite. While we expected things like the left and right antecubital fossa to be associated, we were also interested in associations from disparate types of bodysites. Here we again used a method very similar to what we did for associating communitytypes with categorical metadata. An important difference is that instead of picking a random visit for each bodysite, we picked a random visit for each person and used all of the bodysite data for that person from that visit. We then iterated a bunch. These data became the basis for the heatmap in Figure 2.

```
In [ ]:
```

```
%%R

totalIters <- 1000

#Set up the community type data
v1<-read.table(file="community_types.v1.txt", header=T, row.names=1)
v2<-read.table(file="community_types.v2.txt", header=T, row.names=1)
v3<-read.table(file="community_types.v3.txt", header=T, row.names=1)

randsids <- unique(c(rownames(v1), rownames(v2), rownames(v3)))
bodysites <- colnames(v1)

#make a table to keep track of when a subject visited. we will want
#to take a person's entire microbiome, not just individual sites
visits <- matrix(as.numeric(rep(NA, length(randsids)*3)), ncol=3)
rownames(visits) <- randsids
colnames(visits) <- c("v1", "v2", "v3")
visits[rownames(v1),1] <- 1
visits[rownames(v2),2] <- 2
visits[rownames(v3),3] <- 3


total <- 0
times.sig <- matrix(rep(0, 18*18), ncol=18)
rownames(times.sig) <- bodysites
colnames(times.sig) <- bodysites

median.iter <- array(NA, dim=c(18,18,totalIters))

for(iter in 1:totalIters){  #this is going to take awhile (triple loop!)
        if(iter %% 10 == 0){
                print(iter)
        }

        #create a random visit across all individuals
        random.visit <- matrix(ncol=18, nrow=0)
        colnames(random.visit) <- bodysites
```

```
    for(i in randsids){#for each individual, randomly pick one of their vis
its...
        visit <- as.numeric(sample(na.omit(as.character(visits[i,])))[1
])

        #and once we've got that random visit, take all of their bodysi
te
        #data for that visit
        if(visit == 1){
            random.visit <- rbind(random.visit, v1[i,])
        } else if(visit ==2){
            random.visit <- rbind(random.visit, v2[i,])
        } else if(visit == 3){
            random.visit <- rbind(random.visit, v3[i,])
        } else {
            random.visit <- rbind(random.visit, rep(NA, length(body
sites)))
        }
    }

    #let's build a p-value matrix for each bodysite vs. each bodysite
    p.values <- matrix(rep(NA, 18*18), ncol=18)

    for(i in 2:18){
        for(j in 1:i){
            if(j<i){ #no need to do everything twice...
                contingency <- table(random.visit[,i], random.v
isit[,j])
                nrows <- sum(apply(contingency, 1, sum)>0)
                ncols <- sum(apply(contingency, 2, sum)>0)

                if(nrows == 1 | ncols == 1){#not sure this woul
d happen, but...
                    p.values[i,j] <- NA
                }
                else if(nrows > 2 | ncols > 2){ #if it's more t
han 2x2, bootstrap
                    p.values[i,j] <- fisher.test(contingenc
y, simulate.p.value=T, B=1e5)$p.value
                } else{ #but if it's 2x2 just do the normal met
hod
                    p.values[i,j] <- fisher.test(contingenc
y)$p.value
                }
            }
        }
    }

    adjusted <- p.adjust(p.values, method="BH")     #adjust the p.values
    correct <- adjusted < 0.05      #significant?

    times.sig <- correct + times.sig #increment if significant
    total <- total + 1                                      #increment numb
```

```
        total <- total + 1                                          #increment numb
er of times through

        median.iter[, , iter] <- p.values       #keep track of p.values
}

sig.rate <- times.sig / total           #get the fraction significant
sig.rate <- as.matrix(as.dist(sig.rate, upper=T))#make the matrix symmetric
diag(sig.rate) <- 1
        #make the diagonal 1
rownames(sig.rate) <- bodysites #clean things up a bit
colnames(sig.rate) <- bodysites #clean thigns up a bit
write.table(sig.rate, file="body_site.sig.rate", quote=F, sep="\t")#output!

#ditto on the median p-values
medians <- as.matrix(as.dist(apply(median.iter, c(1,2), median)))
diag(medians) <- 1e-6
rownames(medians) <- bodysites
colnames(medians) <- bodysites
write.table(medians, file="body_site.medians", quote=F, sep="\t")


#now to build the heatmap of p-values...
bsite_ordered <- c(             #want to sort the bodysites to make sense
        4,      #"Keratinized_gingiva",
        15,     #"Supragingival_plaque",
        14,     #"Subgingival_plaque",
        2,      #"Buccal_mucosa",
        3,      #"Hard_palate",
        17,     #"Tongue_dorsum",
        16,     #"Saliva",
        8,      #"Palatine_tonsils",
        16,     #"Throat",
        13,     #"Stool",
        1,      #"Anterior_nares",
        5,      #"L_Antecubital_fossa",
        10,     #"R_Antecubital_fossa",
        6,      #"L_Retroauricular_crease",
        11,     #"R_Retroauricular_crease",
        18,     #"Vaginal_introitus",
        7,      #"Mid_vagina",
        9       #"Posterior_fornix"
)

#scale the values between 1e-5 and 1
scaled.values <- 7 - ceiling(as.matrix((-log10(medians[bsite_ordered,bsite_orde
red]))))
scaled.values[scaled.values == 0] <- 1
scaled.values[is.na(scaled.values)] <- 6

pdf(file="body_site.pdf")
heatmap(scaled.values, Rowv=NA, Colv=NA, scale="none",
        labRow=gsub("_", " ", colnames(medians)[bsite_ordered]),
    labCol=gsub("_", " ", colnames(medians)[bsite_ordered]),margins=c(10,10), r
```

```
evC=T,
    col=c(heat.colors(5, alpha=0.8), "white"), font=2)


par(xpd=T)      #let's write anywhere!
#loc<-locator()
loc <- list("x"=0.83, "y"=0.1)

legend(loc, legend=c(expression(paste("P<10"^"-1")), expression(paste("P<10"^"-
2")),
                        expression(paste("P<10"^"-3")), expression(paste("P<10"
^"-4")),
                        expression(paste("P<10"^"-5"))), fill=rev(heat.colors(5
, alpha=0.8)))
par(xpd=F)      #no more writing anywhere :(
dev.off()
```

Quantify rate of change between community types for each body site

Now we turned to the question of whether some bodysites were more stable than others. Since the subjects didn't come in on a regular interval, it was difficult to quantify true rates of change or to model the change. We tried several things all of which pointed to the fact that time alone could not explain the change. So we just present the rates of change for each community type and body site (see Figure 3). First we'll start by generating transition matrices for each bodysite

In [ ]:

```
%%R

v1 <- read.table(file="community_types.v1.txt", sep=" ", header=T)
v2 <- read.table(file="community_types.v2.txt", sep=" ", header=T)
v3 <- read.table(file="community_types.v3.txt", sep=" ", header=T)

bodysites <- colnames(v1)

for(bs in bodysites){#for each bodysite
        v1.samples <- rownames(v1)#get the the randsid
        v1.parts <- as.character(v1[, grepl(bs, colnames(v1))])#get the communi
ty type
        v1.samples <- v1.samples[!is.na(v1.parts)]      #exclude those sampes w
here we don't have a
        v1.parts <- v1.parts[!is.na(v1.parts)]      #community type

        v2.samples <- rownames(v2)
        v2.parts <- as.character(v2[, grepl(bs, colnames(v2))])
        v2.samples <- v2.samples[!is.na(v2.parts)]
        v2.parts <- v2.parts[!is.na(v2.parts)]

        v3.samples <- rownames(v3)
        v3.parts <- as.character(v3[, grepl(bs, colnames(v3))])
        v3.samples <- v3.samples[!is.na(v3.parts)]
        v3.parts <- v3.parts[!is.na(v3.parts)]

    #now we know the samples that have community types

        samples <- unique(sort(c(v1.samples, v2.samples, v3.samples))) #the lis
t of samples with types
        visits <- data.frame(v1=rep(NA, length(samples)), v2=rep(NA, length(sam
ples)),
                         v3=rep(NA, length(samples)), row.names=samples)#initia
lize a frame

        visits[(v1.samples), 1] <- v1.parts #fill the data frame with community
 type data
        visits[(v2.samples), 2] <- v2.parts #from each visit
        visits[(v3.samples), 3] <- v3.parts

        first <- c(visits[,1], visits[,2]) #create a vector for visits 1 and 2
        second <- c(visits[,2], visits[,3])#create a vector for visits 2 and 3
        #now we can compare first and second to see whether a subject changed b
etween visits 1 and 2
    #and between visits 2 and 3

        #need to make sure that we have all the community types in the rows and
 columns
        types <- levels(factor(c(first, second)))
        ntypes <- length(types)
        transition <- matrix(rep(0, ntypes*ntypes), nrow=ntypes)
        rownames(transition) <- types
        colnames(transition) <- types

        t <- table(first, second)
        transition[rownames(t), colnames(t)] <- t #now we should have a square
transition matrix

    #output
        write.table(transition, paste(bs,".transition.matrix", sep=""), sep="\t
", quote=F)
}
```

Now that we have all of the transition matrices for each bodysite, let's generate a bubble plot indicating the stability of each bodysite along with its relative frequency (e.g. Figure 3A). Along the x-axis we'll plot the stability of each community type, along the y-axis we'll plot the bodysite (bunched into general types) and the icon will be sized proportionally to the fraction of samples in that community type. A vertical line will be drawn for each bodysite to indicate the average community type stability across all subjects.

In [ ]:

```
%%R

#need to read in the transition matrix and calculate the stability values and f
raction
#of subjects
getStabProp <- function(bodysite){
        data <- read.table(file=bodysite, header=T)
        parts <- unique(c(colnames(data), rownames(data)))

    #get the proportions - what fraction of the rows go to columns
        prop <- prop.table(as.matrix(data), 1)

        #define stability as 1 - the proportion of change
    stability <- 100*(1-diag(prop)) #get stability values

    #what fraction of people are in each community type
        proportion <- rowSums(data)/sum(data)
        data.frame(row.names=rownames(data), stability=stability, proportion=pr
oportion)
}


#need to scale the size of the blip to be proportional to the proportion of peo
ple in
#that community type
getCEX <- function(proportion){
        max.cex=5
        min.cex=0.25
        cex <- (max.cex-min.cex)*proportion+min.cex
        return(cex)
}


ll=0.25  #length of the line indicating the average above the y-position
jitter =0.1 #amount to jitter points if there are duplicate stability values

#need to plot the stability data along a line for each chunk of bodysites. If t
here are
#duplicate stability values for a bodysite, need to jitter the position on the
y-axis
plotStability <- function(site.vector, pos){

        for(site in site.vector){#do on all of the bodysites within a bodytype
chunk

                data <- getStabProp(site)#get the stability / proportion data

                ypos <- rep(pos, nrow(data))#set the default position on the y-
axis

        #if there are duplicate stability values jitter them to either position
 on the y-axis
                ypos[duplicated(data$stability)] <- ypos[duplicated(data$stabil
ity)] + jitter
                ypos[duplicated(data$stability, fromLast=T)] <- ypos[duplicated

(data$stability, fromLast=T)] - jitter

        #plot the blips on the line sized in proportion to relative abundance
                points(x=data$stability, y=ypos, pch=21, bg="gray", cex=getCEX(
data$proportion))

        #calculate the average stability for the bodysite
        avg <- sum(data$stability*data$proportion)

        #make a vertical line at the average
                lines(x=c(avg, avg), y=c(pos-ll, pos+ll), col="black", lwd=3)

        #decrement the position on the y-axis
        pos <- pos-1
        }

}



header.line = 11  #position left of the y-axis for the body type heading
indent.line = 9   #position left of the y-axis for the bodysite heading

pdf("bodysite.change.rate.pdf", height=11, width=8.5)
par(mar=c(5, header.line+0.5, 0.5, 0.5)) #need padding on the right

plot(x=c(0,100), y=c(1,27), type="n", main="", xlab="% Change", ylab="", axes=F
)
axis(1)
box()


#gastrointestinal
sites <- c("Stool.transition.matrix")
names <- gsub(".transition.matrix", "", sites)
names <- gsub("_", " ", names)
mtext(side=2, line=header.line, at=27, text="Gastrointestinal", las=1, adj=0)
mtext(side=2, line=indent.line, at=26, text="Stool", las=1, adj=0)
abline(h=26, col="gray")#make horizontal gray line

start <- 26
plotStability(sites, start)


#oral
sites <- c("Supragingival_plaque.transition.matrix", "Palatine_Tonsils.transiti
on.matrix",
                        "Subgingival_plaque.transition.matrix", "Throat.transit
ion.matrix",
                        "Tongue_dorsum.transition.matrix", "Keratinized_gingiva
.transition.matrix",
                        "Buccal_mucosa.transition.matrix", "Hard_palate.transit
                        "Buccal_mucosa.transition.matrix", "Hard_palate.transit
ion.matrix", "Saliva.transition.matrix")

names <- gsub(".transition.matrix", "", sites)
names <- gsub("_", " ", names)
mtext(side=2, line=header.line, at=24, text="Oral", las=1, adj=0)
mtext(side=2, line=indent.line, at=15:23, text=rev(names), las=1, adj=0)
abline(h=15:23, col="gray")

start <- 23
plotStability(sites, start)



#pulmonary
sites <- c("Anterior_nares.transition.matrix")
names <- gsub(".transition.matrix", "", sites)
names <- gsub("_", " ", names)
mtext(side=2, line=header.line, at=13, text="Pulmonary", las=1, adj=0)
mtext(side=2, line=indent.line, at=12, text=rev(names), las=1, adj=0)
abline(h=12, col="gray")

start <- 12
plotStability(sites, start)



#skin
sites <- c("R_Retroauricular_crease.transition.matrix", "L_Retroauricular_creas
e.transition.matrix",
                        "R_Antecubital_fossa.transition.matrix", "L_Antecubital
_fossa.transition.matrix")
names <- gsub(".transition.matrix", "", sites)
names <- gsub("_", " ", names)
mtext(side=2, line=header.line, at=10, text="Skin", las=1, adj=0)
mtext(side=2, line=indent.line, at=6:9, text=rev(names), las=1, adj=0)
abline(h=6:9, col="gray")

start <- 9
plotStability(sites, start)



#vagina
sites <- c("Vaginal_introitus.transition.matrix","Mid_vagina.transition.matrix"
,
                        "Posterior_fornix.transition.matrix")
names <- gsub(".transition.matrix", "", sites)
names <- gsub("_", " ", names)
mtext(side=2, line=header.line, at=4, text="Vagina", las=1, adj=0)
mtext(side=2, line=indent.line, at=1:3, text=rev(names), las=1, adj=0)
abline(h=1:3, col="gray")

start <- 3
plotStability(sites, start)



#make a legend, for Figure 3A we separated the points a bit so they were easier
```

```
#make a legend: for Figure 3A we separated the points a bit so they were easier
 to see

legend(x=80, y=13, legend=c("1%", "10%", "25%", "50%", "75%"), pch=21, pt.bg="g
ray",
                    bg="white", pt.cex=getCEX(c(0.01, 0.10, 0.25, 0.50, 0.75)))

dev.off()
```

Now we'd like to graphically depict the transition matrices as we did in Figure 3B. These will allow us to see how frequently one community type changes into another or stays the same. This took a fair amount of messing with parameters to get the formatting to look decent...

In [ ]:

```
%%R

install.packages("diagram")
library(diagram)

#set the layouts to use in plotmat for different number of community types
getPos <- function(n) {
        if(n==2)                {               return(c(2))            }

        else if(n==3)   {               return(c(1,2))          }

        else if(n==4)   {               return(c(2,2))          }

        else if(n==5)   {               return(c(2,1, 2))       }

        else if(n==6)   {               return(c(2,2,2))        }
        else if(n==7)   {               return(c(2,3,2))        }
}


#describe the curvature when there are different numbers of community types
getCurves <- function(n) {
        if(n==2){
                return(matrix(c(0,      0.1,
                                                0.1,    0), nrow=2))
        }
        else if(n==3){
                return(matrix(c(0,      0.08,   0.08,
                                                0.08,   0,      0.08,
                                                0.08,   0.08,   0), nrow=3))
        }
        else if(n==4){
                return(matrix(c(0.00,   0.08,   0.08,   0.65,
                                                0.08,   0.00,   0.65,   0.08,
                                                0.08,   0.65,   0.00,   0.08,
                                                0.65,   0.08,   0.08,   0.00),
        4))
nrow=4))
        }

        else if(n==5){
                return(matrix(c(0,      0.05,   0.1,    0.05,   0.6,
                                                0.05,   0,      0.1,    0.6,
        0.05,
                                                0.1,    0.1,    0,      0.1,
        0.1,
                                                0.05,   0.6,    0.1,    0,
        0.05,
                                                0.6,    0.05,   0.1,    0.05,
        0), nrow=5))
        }
        else if(n==6){
                return(matrix(c(0,      0.05,   0.05,   0.1,    0.2,    0.15,
                                                0.05,   0,      0.1,    0.05,
        0.15,   0.2,
                                                0.05,   0.1,    0,      0.05,
        0.05,   0.1,
                                                0.1,    0.05,   0.05,   0,
        0.1,    0.05,
                                                0.2,    0.15,   0.05,   0.1,
        0,      0.05,
                                                0.15,   0.2,    0.1,    0.05,
        0.05,   0), nrow=6))
        }
        else if(n==7){
                return(matrix(c(0,      0.15,   0.15,   0.15,   0.1,    0.1,
        0.25,
                                                0.15,   0,      0.1,    0.15,
        0.15,   0.25,   0.1,
                                                0.15,   0.1,    0,      0.15,
        0.25,   0.15,   0.1,
                                                0.15,   0.15,   0.15,   0,
        0.15,   0.15,   0.15,
                                                0.1,    0.15,   0.25,   0.15,
        0,      0.1,    0.15,
                                                0.1,    0.25,   0.15,   0.15,
        0.1,    0,      0.15,
                                                0.25,   0.1,    0.1,    0.15,
        0.15,   0.15,   0), nrow=7))
        }
}


#describe on which side (left/right) and how big self arrows should be
getShiftX <- function(n) {
        if(n==2)                {       return(c(-0.09, 0.09))
                                                }
        else if(n==3)   {       return(c(0,-0.09,0.09))
                                        }
        else if(n==4)   {       return(c(-0.09, 0.09, -0.09, 0.09))
                                }
        else if(n==5)   {       return(c(-0.09, 0.09, 0.09, -0.09, 0.09))
                                }

        else if(n==6)   {       return(c(-0.09, 0.09, -0.09, 0.09, -0.09, 0.09)
)               }
        else if(n==7)   {       return(c(-0.09, 0.09, -0.09, 0.09, 0.09,-0.09,
0.09))  }
}


#describe on which side (top/bottom) and how big self arrows should be
getShiftY <- function(n) {
        if(n==2)                { return(0) }
        else if(n==3)   { return(c(0.09,-0.04,-0.04)) }
        else if(n==4)   { return(0) }
        else if(n==5)   { return(0) }
        else if(n==6)   { return(0) }
        else if(n==7)   { return(0) }
}



matrices <- c("Anterior_nares.transition.matrix", "Buccal_mucosa.transition.mat
rix",
                        "Hard_palate.transition.matrix", "Keratinized_gingiva.t
ransition.matrix",
                        "L_Antecubital_fossa.transition.matrix", "L_Retroauricu
lar_crease.transition.matrix",
                        "Mid_vagina.transition.matrix", "Palatine_Tonsils.trans
ition.matrix",
                        "Posterior_fornix.transition.matrix", "R_Antecubital_fo
ssa.transition.matrix",
                        "R_Retroauricular_crease.transition.matrix", "Saliva.tr
ansition.matrix",
                        "Stool.transition.matrix", "Subgingival_plaque.transiti
on.matrix",
                        "Supragingival_plaque.transition.matrix", "Throat.trans
ition.matrix",
                        "Tongue_dorsum.transition.matrix", "Vaginal_introitus.t
ransition.matrix")

for(m in matrices){#go through all of the bodysites

        trans <- read.table(file=m, header=T)        #want to know the fraction
 of
        trans <- 100*prop.table(as.matrix(trans), 1) #community type row that b
ecome community type column
        trans <- format(trans, nsmall=1, digits=1)   #try to make the numbers p
retty
        ntypes <- nrow(trans) #get the number of types

        bs <- gsub("^([^.]*).*", "\\1", m) #get the bodysite
        bs <- gsub("_", " ", bs)

    #plot the transition state diagram
    #plot the transition state diagram
        pdf(file=gsub("matrix", "pdf", m))

        plotmat(
                t(trans),
                pos = getPos(ntypes),
                curve = getCurves(ntypes),
                self.shiftx = getShiftX(ntypes),
                self.shifty = getShiftY(ntypes),

                name = LETTERS[1:ntypes],
                lwd = 1.5,
                box.lwd = 2,
                cex.txt = 0.75,
                box.cex = 1.2,
                box.size = 0.06,
                arr.length = 0.5,
                box.type = "circle",
                box.prop = 1,
                shadow.size = 0,
                self.cex = 0.6,
                my = -0.075,
                mx = -0.01,
                relsize = 0.95,
                cex.main=1.5,
                box.col="gray",
                txt.col="black",
                arr.col="black",
                dtext=0.5,
                absent=-0.1,
                main = bs,
        latex=T
        )
        dev.off()
}
```

Effect of time on stability of community types

Since the community types were not fixed for each person and some community types were not stable, we next endeavored to determine whether we could associate changes in community types with the duration between samplings. As we mention in the paper, it was not possible to identify any significant relationships.

In [ ]:

```
%%R

#get the community type data for each visit
v1 <- read.table(file="community_types.v1.txt", sep=" ", header=T)
v2 <- read.table(file="community_types.v2.txt", sep=" ", header=T)
```

```
v3 <- read.table(file="community_types.v3.txt", sep=" ", header=T)
bodysites <- colnames(v1)


#get the dates of the various visits
md <- read.table(file="../dbGaP/phs000228.v3.pht002157.v1.p1.c1.EMMES_HMP_DTP_D
HX_DVD.HMP.txt", header=T, sep="\t", na.string=c(NA, ""))
md1 <- md[md$VISNO == "01", c("RANDSID", "study_day")]
md2 <- md[md$VISNO == "02", c("RANDSID", "study_day")]
md3 <- md[md$VISNO == "03", c("RANDSID", "study_day")]

randsids <- levels(factor(md$RANDSID))
dates <- matrix(rep(NA, length(randsids)*3), nrow=length(randsids))
rownames(dates) <- randsids
colnames(dates) <- c("visit1", "visit2", "visit3")

#create a dates matrix so we know when each visit occured
dates[as.character(md1$RANDSID), "visit1"] <- md1$study_day
dates[as.character(md2$RANDSID), "visit2"] <- md2$study_day
dates[as.character(md3$RANDSID), "visit3"] <- md3$study_day


for(bs in bodysites){#go through the bodysites
        types <- matrix(rep(NA, length(randsids)*3), ncol=3) #create a types ma
trix for the bodysite so we know the type
        colnames(types) <- c("visit1", "visit2", "visit3")   #at each visit
        rownames(types) <- randsids

        types[rownames(v1), "visit1"] <- v1[,bs] #fill the matirx
        types[rownames(v2), "visit2"] <- v2[,bs]
        types[rownames(v3), "visit3"] <- v3[,bs]

    #figure out who had the same community type at visits 1 and 2 and at visits
 2 and 3
        v12.same <- !is.na(types[,"visit1"]) & !is.na(types[,"visit2"]) & types
[,"visit1"] == types[,"visit2"]
        v23.same <- !is.na(types[,"visit2"]) & !is.na(types[,"visit3"]) & types
[,"visit2"] == types[,"visit3"]

    #get the number of days between visits that had the same community type
    dates.same <- c(dates[v12.same, "visit2"] - dates[v12.same, "visit1"], date
s[v23.same, "visit3"] -
                    dates[v23.same, "visit2"])


    #figure out who had different community type at visits 1 and 2 and at visit
s 2 and 3
    v12.different <- !is.na(types[,"visit1"]) & !is.na(types[,"visit2"]) & type
s[,"visit1"] != types[,"visit2"]
        v23.different <- !is.na(types[,"visit2"]) & !is.na(types[,"visit3"]) &
types[,"visit2"] != types[,"visit3"]

    #get the number of days between visits that had different community types
    dates.different <- c(dates[v12.different, "visit2"] - dates[v12.different,
"visit1"], dates[v23.different, "visit3"]
"visit1"], dates[v23.different, "visit3"] -
                            dates[v23.different, "visit2"])



    #test for a significant difference in the days between visits with differen
t community types vs. those that were the same
        pooled.dates <- c(dates.same, dates.different)
        category <- c(rep("Same\nCommunity Type", length(dates.same)), rep("Dif
ferent\nCommunity Type", length(dates.different)))
        p.value <- format(wilcox.test(pooled.dates~category)$p.value, digits=3)


    #plot the data
    pdf(file=paste(bs, ".time.pdf", sep=""))
        par(mar=c(5, 8, 0.5, 0.5))
        stripchart(pooled.dates~category, method="jitter", yaxt="n", xaxt="n",
ylab="", xlab="Days between samplings",
               xlim=c(0,max(md3$study_day)))
        axis(1)
        axis(2, at=c(1,2), label=levels(factor(category)), tick=F, las=2)
        text(x=450, y=2.18, label=paste(bs, "\n(P=", p.value, ")", sep=""), pos
=2)
        dev.off()
}
```

Running this and looking at the resulting pdfs makes it clear that none of the times between samplimgs for the community types that changed and those that didn't change were significantly different. The P-value in the upper right corner of each plot is from a non-parametric Wilcox Test.

Validate DMM models as a community type approach relative to the PAM-based approach

Nearly every study to perform community typing has used partitioning around the medoid with distances between samples as the basis for identifying community types. Metrics (e.g. CH and Silouette) are then used to pick the optimal number of community types. For some reason Koren et al. 2013 (http://www.ploscompbiol.org/article/info%3Adoi%2F10.1371%2Fjournal.pcbi.1002863) did not use the DMM approach eventhough it was published well before their study. We did our best to replicate their methods of simulating a community with four types and evaluate the PAM and DMM-based approach. We also, simulated a community with only one community type to look at the likelihood of falsely detecting a community type. Finally, we generated PAM-based community types for each bodysite and evaluated their Laplace score for those types. These results are summarized in the Supplemental Data section of our paper.

Simulating communities with one and four community types

The above mentioned Koren et al. 2013 (http://www.ploscompbiol.org/article/info%3Adoi%2F10.1371%2Fjournal.pcbi.1002863) study simulated four community types. Here's their methods:

> "We generated a synthetic dataset of 100 communities each containing 3,000 “sequences” belonging to 500 mock OTUs. For each synthetic community, 90% (2,700 sequences, or OTU observations) was drawn from the same randomly generated lognormal abundance distribution (shared across all communities) and the remaining 10% (300 sequences) drawn from one of four unique lognormal distributions, forcing the data into four clusters. We then applied the enterotyping methods as described above."

This was a bit vague (shape parameters?). Plus it didn't include a negative control, which would be a single community type. Plus they didn't do the DMM analysis. So we attempated to replicate and improve upon their analysis with two sets of imulations:

- **Single community type.** We constructed a truncated lognormal distribution with a normal mean of 1.25 and a normal standard deviation of 3.00 and a maximum richness of 500. For each of 100 simulated communities we drew 3,000 random integers (i.e. sequences) from this distribution where each integer represented a specific OTU.
- **Four community types.** We used the same truncated lognormal distribution described for the single cluster simulation to draw 2,700 integers sequences for each of 25 samples. We then permuted the distribution and drew an additional 300 integers from the permuted distribution and added those counts to non-permuted distribution. We repeated this procedure three additional times so that we had effectively simulated sampling 3,000 sequences from 100 samples representing 4 community types.

In the following code chunk we'll generate the **quad**ruple simulated data and the single simulated data and output them as shared files so we can analyze them using the PAM and DMM-based approaches

In [ ]:

```
%%bash
cd ..
mkdir simulation
cd simulation/
```

In [3]:

```
%%R

getTruncatedLogNormal <- function(nseqs, limit, mu, sd){
    mu <- 1.25
    sd <- 3.00

        dist <- round(rlnorm(nseqs*10, meanlog=mu, sdlog=sd))+1   #pull 10-fold
 more sequences than we need
```

```
        dist <- dist[dist < limit]                                  #in case some
 go over limit (maximum richness)

        dist <- dist[0:nseqs]                                       #return the f
irst nsequences
}


iterations <- 100

lower <- 1
upper <- 500

frac.z <- 0.90                      #fraction to draw from the common distribu
tion - z = common distribution
nseqs <- 3000                       #number of sequences we want to draw
ncommunities <- 100                 #number of communities to sample from
samples_per <- ncommunities * 0.25  #we want each community to have 25% of the
 samples

a.sample <- sample(1:upper)         #here is the permutation of the ordering o
f the OTUs that corresponds
b.sample <- sample(1:upper)         #to each of the four community types. this
 way the "extra" distribution
c.sample <- sample(1:upper)         #will be the same shaped distribution, jus
t in a different order
d.sample <- sample(1:upper)

for(iter in 1:iterations){
    #we're going to fill each of the 4 community types with data

        a<-matrix(nrow=samples_per, ncol=upper)
        for(i in 1:samples_per){   #now looping over each of the 25 samples...
                a.dist <- getTruncatedLogNormal(nseqs*(1-frac.z), upper, mu, sd
) #the a component draws from the LN distro and reorders
                z.dist <- getTruncatedLogNormal(nseqs*frac.z, upper, mu, sd)
  #the z component is the regular ordered LN distro
                a.table <- table(c(a.sample[a.dist], z.dist))
  #get the number of times each OTU/integer shows up
                x <- rep(0, upper); names(x) <-1:500       #make sure everythin
g is on the same OTU naming scheme
                x[names(a.table)] <- a.table
                a[i,] <- x     #viola - the a component of the shared file - re
peat for b, c, and d
        }

        b<-matrix(nrow=samples_per, ncol=upper)
        for(i in 1:samples_per){
                b.dist <- getTruncatedLogNormal(nseqs*(1-frac.z), upper, mu, sd
)
                z.dist <- getTruncatedLogNormal(nseqs*frac.z, upper, mu, sd)
                b.table <- table(c(b.sample[b.dist], z.dist))
                x <- rep(0, upper); names(x) <-1:500
                x[names(b.table)] <- b.table
                b[i,] <- x
        }


        c<-matrix(nrow=samples_per, ncol=upper)
        for(i in 1:samples_per){
                c.dist <- getTruncatedLogNormal(nseqs*(1-frac.z), upper, mu, sd
)
                z.dist <- getTruncatedLogNormal(nseqs*frac.z, upper, mu, sd)
                c.table <- table(c(c.sample[c.dist], z.dist))
                x <- rep(0, upper); names(x) <-1:500
                x[names(c.table)] <- c.table
                c[i,] <- x
        }

        d<-matrix(nrow=samples_per, ncol=upper)
        for(i in 1:samples_per){
                d.dist <- getTruncatedLogNormal(nseqs*(1-frac.z), upper, mu, sd
)
                z.dist <- getTruncatedLogNormal(nseqs*frac.z, upper, mu, sd)
                d.table <- table(c(d.sample[d.dist], z.dist))
                x <- rep(0, upper); names(x) <-1:500
                x[names(d.table)] <- d.table
                d[i,] <- x
        }

        shared <- rbind(a, b, c, d)  #make shared file

        shared <- shared[, apply(shared, 2, sum) > 0]  #remove columns that onl
y have zeros
        notus <- ncol(shared) #build the shared file...
        label <- rep(1, ncommunities)  #the first column has a common label

    #the second column has the sample name - here we indicate the community typ
e for each
        groups <- c(paste(1:samples_per, ".A", sep=""),
                paste(1:samples_per, ".B", sep=""),
                paste(1:samples_per, ".C", sep=""),
                paste(1:samples_per, ".D", sep=""))
        notus.col <- rep(notus, ncommunities)  # the third column has the numbe
r of OTUs
        full.shared <- cbind(label, groups, notus.col, shared)  #...build the s
hared file
        colnames(full.shared) <- c("label", "Group", "numOTUs", 1:notus)
        write.table(file=paste("quad_", iter, ".shared", sep=""), full.shared,
row.names=F, quote=F, sep="\t")
    #done!
}


#generate the single case...
frac.z <- 1.0000  #now we want to build a shared file where there's only one co
mmunity type
for(iter in 1:iterations){
        z<-matrix(nrow=ncommunities, ncol=upper)
        for(i in 1:ncommunities){

                z.dist <- getTruncatedLogNormal(nseqs*frac.z, upper, mu, sd)  #
again draw from our lognormal disribution
                z.table <- table(z.dist)                        #everything else is p
retty much the same...
                x <- rep(0, upper); names(x) <-1:500
                x[names(z.table)] <- z.table
                z[i,] <- x
        }
        shared <- z
        shared <- shared[, apply(shared, 2, sum) > 0]
        notus <- ncol(shared)
        label <- rep(1, ncommunities)
        groups <- c(paste(1:ncommunities, ".A", sep=""))
        notus.col <- rep(notus, ncommunities)

        full.shared <- cbind(label, groups, notus.col, shared)
        colnames(full.shared) <- c("label", "Group", "numOTUs", 1:notus)

        write.table(file=paste("single_", iter, ".shared", sep=""), full.shared
, row.names=F, quote=F, sep="\t")
}
```

```
ERROR: Cell magic `%%R` not found.
```

Now that we've built the single and quadruple community type shared files we want to know whether we can detect those community types. First we'll start with the PAM-based approach. As we mention in the supplement, the Silhouette Index (SI) is prefered over the Calinski-Harabasz (CH) statistic because it had a more objective basis. They asserted that SI values below 0.25 have a lack of substantial structure to the clusters. Let's test this out...

In [ ]:

```
require(cluster)
require(fpc)

n.simulations <- 100

dist.JSD <- function(inMatrix, pseudocount=0.000001, ...) {

        KLD <- function(x,y) sum(x *log(x/y))

#nb:    the line below is from the MetaHit group, which is actually the
#               root JSD, not the straight up JSD
#       JSD <- function(x,y) sqrt(0.5 * KLD(x, (x+y)/2) + 0.5 * KLD(y, (x+y)/2)
)
        JSD <- function(x,y) (0.5 * KLD(x, (x+y)/2) + 0.5 * KLD(y, (x+y)/2))
        matrixColSize <- length(colnames(inMatrix))
        matrixRowSize <- length(rownames(inMatrix))

        colnames <- colnames(inMatrix)
        resultsMatrix <- matrix(0, matrixColSize, matrixColSize)

        #I'm not sure I "believe" in JSD, but it's what all the enterotype pape
rs
        #have used so I use it here. There seems to be a funkiness here in that
        #you have to add a very small number to the relative abundances since y
ou
        #can easily have a divide by zero. Sketchy for such sparse data matrice
s. IMHO.
        inMatrix = apply(inMatrix,1:2,function(x) ifelse (x==0,pseudocount,x))

        for(i in 1:matrixColSize) {
                for(j in 1:matrixColSize) {
                        if(j > i){      #no need to do everything twice.
                                resultsMatrix[i,j] <- JSD(as.vector(inMatrix[,i
]),

                as.vector(inMatrix[,j]))
                                resultsMatrix[j,i] <- resultsMatrix[i,j]
                        }
                }
        }
        colnames -> colnames(resultsMatrix) -> rownames(resultsMatrix)
        as.dist(resultsMatrix)->resultsMatrix
        attr(resultsMatrix, "method") <- "dist"
        return(resultsMatrix)
}

#use the built in pam clustering algoirthm
pam.clustering <- function(x,k) { # x is a distance matrix and k the number of
clusters
        cluster <- as.vector(pam(as.dist(x), k, diss=TRUE)$clustering)
        return(cluster)
}


pam.approach <- function(prefix){
    max.ch <- vector()
    max.silhouette <- vector()

    write(paste("simulation", "k_CH", "max_CH", "k_Silhouette", "max_Silhoutte"
),
          paste(prefix, "enterotype_output.txt", sep=""), append=F)

    for(i in 1:n.simulations){

        #read back in the simulated shared file
        data <- read.table(paste(prefix, i, ".shared", sep=""), header=T, row.n
ames=2)
        data <- data[,-c(1,2)]
        data <- data / apply(data, 1, sum)
        data <- data / apply(data, 1, sum)
        data <- t(data)#need OTUs in rows and samples in columns


        data.dist <- dist.JSD(data)#get the distance matrix

        ch <- vector()
        width <- vector()

        for (k in 2:10) { #calcualte the ch and silhouette for k=2:10 (can't do
 k=1)
            data.cluster <- pam.clustering(data.dist, k)
            stats <- cluster.stats(data.dist, data.cluster)
            ch[k] <- stats$ch
            width[k] <- stats$avg.silwidth
        }

        max.ch[i] <- which.max(ch)      #get the maximum
        max.silhouette[i] <- which.max(width) #get the maximum

        write(paste(i, max.ch[i], max(ch, na.rm=T), max.silhouette[i], max(widt
h, na.rm=T)),
              paste(prefix, "enterotype_output.txt", sep=""), append=T)
    }
}

pam.approach("single_")
pam.approach("quad_")


single <- read.table(file="single_enterotype_output.txt", header=T, row.names=1
)
single$k_CH <- factor(single$k_CH)
single$k_Silhouette <- factor(single$k_Silhouette)
summary(single)

quadruple <- read.table(file="quad_enterotype_output.txt", header=T, row.names=
1)
quadruple$k_CH <- factor(quadruple$k_CH)
quadruple$k_Silhouette <- factor(quadruple$k_Silhouette)
summary(quadruple)
```

And so we see from these final commands that the PAM-based methods can't really call a single community type case (although the scores are admittedly low) and while it can call the four community type case, the scores are pretty mediocre by Koren et al.'s standards. Next we want to run the DMM models on the community type data. Like we did for the microbiome data, we will run these in parallel to control for picking the wrong starting condition and the natural variation in the algorithm.

In [4]:

```
%%bash
```

```
mkdir dmm.sim.r1 dmm.sim.r2 dmm.sim.r3 dmm.sim.r4 dmm.sim.r5

>simulated.batch

for SHARED in $(ls single_*.shared quad_*.shared)
do
    echo "mothur \"#get.communitytype(shared=../$SHARED, outputdir=./)\"" >> si
mulated.batch
done

split -l 10 simulated.batch simulated_


cat > header.batch << "_EOF_"
#!/bin/sh
#PBS -l nodes=1:ppn=1
#PBS -l walltime=500:00:00
#PBS -l mem=1gb
#PBS -j oe
#PBS -m abe
#PBS -V
#PBS -M your@email.here
#PBS -q first

echo "ncpus-2.pbs"
cat $PBS_NODEFILE
qstat -f $PBS_JOBID

cd $PBS_O_WORKDIR

NCPUS=`wc -l $PBS_NODEFILE | awk '{print $1}'`

_EOF_


cat > tail.batch << "_EOF_"

echo "qsub working directory absolute is"
echo $PBS_O_WORKDIR
exit
_EOF_


for DIR in dmm.sim.r1 dmm.sim.r2 dmm.sim.r3 dmm.sim.r4 dmm.sim.r5
do
    cd $DIR

    cp ../simulated_?? ./
    for BATCH in $(ls simulated_??)
    do
        cat ../header.batch $BATCH ../tail.batch > $BATCH.$DIR.batch
        qsub $BATCH.$DIR.batch
    done


    cd ../
done
```

```
ERROR: Cell magic `%%` not found.
```

Now that those jobs have completed we want to find the best fit for each simulation

In [ ]:

```
%%R

niters <- 100

getNTypes <- function(prefix){
        ntypes <- rep(NA, niters)

        for(i in 1:niters){
                r1 <- read.table(file=paste("dmm.sim.r1/", prefix, i, ".1.dmm.m
ix.fit", sep=""), header=T, row.names=1)
                r2 <- read.table(file=paste("dmm.sim.r2/", prefix, i, ".1.dmm.m
ix.fit", sep=""), header=T, row.names=1)
                r3 <- read.table(file=paste("dmm.sim.r3/", prefix, i, ".1.dmm.m
ix.fit", sep=""), header=T, row.names=1)
                r4 <- read.table(file=paste("dmm.sim.r4/", prefix, i, ".1.dmm.m
ix.fit", sep=""), header=T, row.names=1)
                r5 <- read.table(file=paste("dmm.sim.r5/", prefix, i, ".1.dmm.m
ix.fit", sep=""), header=T, row.names=1)

                r1[is.infinite(as.matrix(r1))] <- NA
                r2[is.infinite(as.matrix(r2))] <- NA
                r3[is.infinite(as.matrix(r3))] <- NA
                r4[is.infinite(as.matrix(r4))] <- NA
                r5[is.infinite(as.matrix(r5))] <- NA

                maxK <- max(c(nrow(r1), nrow(r2), nrow(r3), nrow(r4), nrow(r5))
)
                laplace <- matrix(rep(NA, 5*maxK), nrow=maxK)
                colnames(laplace) <- c("r1", "r2", "r3", "r4", "r5")
                rownames(laplace) <- 1:maxK

                laplace[1:nrow(r1),"r1"] <- r1[, "Laplace"]
                laplace[1:nrow(r2),"r2"] <- r2[, "Laplace"]
                laplace[1:nrow(r3),"r3"] <- r3[, "Laplace"]
                laplace[1:nrow(r4),"r4"] <- r4[, "Laplace"]
                laplace[1:nrow(r5),"r5"] <- r5[, "Laplace"]

                laplace[apply(is.na(laplace), 1, sum) ==5 ,] <- rep(1e6,5)
                composite <- apply(laplace, 1, min, na.rm=T)
                ntypes[i] <- which.min(composite)

        }
        return(ntypes)
}

n.singletypes <- getNTypes("single_")
table(n.singletypes)

n.quadtypes <- getNTypes("quad_")
table(n.quadtypes)
```

As these results indicate, the DMM-based approach correctly called all of the single and quadruple community type iterations.

Applying PAM-based approach to HMP data

Next we wanted to see how the PAM-based approach did on the observed community data. First, we assigned all of the bodysites to community types for varying numbers of clusters and found the best assignment using SI and CH. Next, we used those assignments to calculate the Laplace metric. This allowed us to compare the PAM and DMM-based assignments directly. Note that a key difference between the PAM and DMM-based approaches is that PAM-based methods require the use of a distance calculator whereas the DMM-based approach does not.

```
In [ ]:
```

```
%%R

shared <- c("Antecubital_fossa.tx.1.subsample.shared", "Anterior_nares.tx.1.sub
sample.shared",
        "Buccal_mucosa.tx.1.subsample.shared", "Hard_palate.tx.1.subsample.shar
ed",
        "Keratinized_gingiva.tx.1.subsample.shared", "Palatine_Tonsils.tx.1.sub
sample.shared",
        "Retroauricular_crease.tx.1.subsample.shared", "Saliva.tx.1.subsample.s
hared",
        "Stool.tx.1.subsample.shared", "Subgingival_plaque.tx.1.subsample.share
d",
        "Supragingival_plaque.tx.1.subsample.shared",   "Throat.tx.1.subsample.
shared",
        "Tongue_dorsum.tx.1.subsample.shared", "Vagina.tx.1.subsample.shared")

path <- "../v35/"

#set up a summary table where the rows are the bodysites an the columns are the
 silhouette width, the number
#of clusters based on SI, the CH index, and the number of clusters based on the
```

```
 CH index
summary <- data.frame(matrix(rep(NA, 14*4), nrow=14, ncol=4))
rownames(summary) <- gsub(".tx.1.subsample.shared", "", shared)
colnames(summary) <- c("width", "w_clusters", "ch", "ch_clusters")

for(s in shared){
        write(s, "")
        stub <- gsub("shared", "", s)
    file <- paste(path, s, sep="")
        data <- read.table(file, header=T, row.names=2)  #get the shared file a
nd turn it into relative abundance data
        data <- data[,-c(1,2)]
        data <- data / apply(data, 1, sum)
        data <- t(data) #transpose it

        data.dist <- dist.JSD(data)  #get the distance matrix

        ch <- vector()
        width <- vector()

        for (k in 2:10) {  #get the si and ch indices for 2 to 10 clusters
                data.cluster <- pam.clustering(data.dist, k)
                stats <- cluster.stats(data.dist, data.cluster)
                ch[k] <- stats$ch
                width[k] <- stats$avg.silwidth
        }

        pdf(paste(stub, "enterotype.pdf", sep=""), height=10, width=5)  #genera
te a spike plot for the SI and CH indices
        par(mfrow=c(2,1))
        plot(ch, type="h", xlab="k clusters", ylab="CH index")
        plot(width~seq(1:10), type="h", xlab="kclusters", ylab="Average Silhoue
tte Width")
        dev.off()

        max.ch <- which.max(ch)  #get the number of types by CH
        max.silhouette <- which.max(width)  #get the number of types by SI

        summary[gsub(".tx.1.subsample.shared", "", s),] <- c(max(width, na.rm=T
), max.silhouette, max(ch, na.rm=T), max.ch)#update the summary file
        write.table(cbind(colnames(data), paste("Partition_", pam.clustering(da
ta.dist, max.ch), sep="")), #output a design file
                paste(stub, "pam.ch.enterotype", sep=""), quote=F, row.names=F,
 col.names=F) #based on the CH index

    write.table(cbind(colnames(data), paste("Partition_", pam.clustering(data.d
ist, max.silhouette), sep="")), #output a design
                paste(stub, "pam.sw.enterotype", sep=""), quote=F, row.names=F,
 col.names=F) #file based on the SI index
}

write.table(summary, file="pam.enterotype.summary", quote=F, sep="\t") #output
the summary file
```

Now that we have the community types by the PAM-based approach, let's run the community type assignments into the DMM model and calculate the Laplace value for each bodysite based on the number of clusters found using the SI and CH indices. To do this, we use a homebrew C++ program that will take in a shared file and a design file. You can get our code from GitHub (https://github.com/SchlossLab/pds_dmm):

In [ ]:

```
%%bash

git clone git@github.com:SchlossLab/pds_dmm.git
cd pds_dmm
make
cd ..

for ENTEROTYPE in $(ls *enterotype)
do
    SHARED=${ENTEROTYPE//pam.*/shared}
        pds_dmm/pds_dmm -shared ../v35/$SHARED -design $ENTEROTYPE
done
```

Now we want to synthesize everything to make a table that looks like Extended Data Table 2.

In [ ]:

```
body_stubs <- c("Antecubital_fossa", "Anterior_nares", "Buccal_mucosa", "Hard_p
alate",
                                "Keratinized_gingiva", "Palatine_Tonsils", "Ret
roauricular_crease",
                                "Saliva", "Stool", "Subgingival_plaque", "Supra
gingival_plaque", "Throat",
                                "Tongue_dorsum", "Vagina")

sausage <- ".tx.1.subsample."

mix_suffix <- "1.dmm.mix.fit"
sw_suffix <- "pam.sw.fit"
ch_suffix <- "pam.ch.fit"

pam.summary <- read.table(file="pam.enterotype.summary", header=T)

extended.table.2 <- matrix(rep(NA, 8*length(body_stubs)), ncol=8)
rownames(extended.table.2) <- body_stubs
colnames(extended.table.2) <- c("SI.clusters", "SI.score", "SI.Laplace", "CH.cl
usters",
                                                                  "CH.score", "CH
.Laplace", "DMM.clusters", "DMM.Laplace")

for(bs in body_stubs){
        dmm.fit <- read.table(file=paste("../v35/dmm_best/", bs, sausage, mix_s
uffix, sep=""), header=T)
        dmm.K <- which.min(dmm.fit$Laplace)
        dmm.Laplace <- min(dmm.fit$Laplace, na.rm=T)

        sw.Laplace <- read.table(file=paste(bs, sausage, sw_suffix, sep=""), he
ader=T)$Laplace
        ch.Laplace <- read.table(file=paste(bs, sausage, ch_suffix, sep=""), he
ader=T)$Laplace

        pam <- pam.summary[bs,]
       extended.table.2[bs,] <- c(pam$w_clusters, pam$width, sw.Laplace, pam$
ch_clusters, pam$ch, ch.Laplace, dmm.K, dmm.Laplace)
}

write.table(file="extended_table_2.txt", extended.table.2, quote=F)
```

Representing community types in ordinations

We strongly resisted the reviewer's request to put in an ordination of the samples colored by the community types. Why? Well, because we know that many OTUs are responsible for dividing the samples amongst the community types and an ordination really only allows you to use a couple of variables. Futhermore, construction of an ordination requires filtering the data through a distance matrix, which is one of the problems with the PAM-based approach. Whatever, here we present code generating the ordinations for each bodysite with coloring for each community type. First we use mothur to calculate the JSD distances between samples. We do this by rarefying the samples to 1000 sequences 100 times. After all of the iterations we calculate the average JSD matrix:

In [ ]:

```
%%bash

for SHARED in $(ls ../v35/*subsample.shared)
do
    DIST=${SHARED//shared/jsd.1.lt.dist}
    DIST=${DIST//..\/v35\//}
        mothur "#dist.shared(shared=$SHARED, calc=jsd, processors=8, outputdir=
./); nmds(phylip=$DIST, maxdim=2)"
done
```

Now we build the NMDS ordination figures and output them as PDFs.

In [ ]:

```
bodysites <- c("Antecubital_fossa", "Anterior_nares", "Buccal_mucosa", "Hard_pa
late", "Keratinized_gingiva",
               "Palatine_Tonsils", "Retroauricular_crease", "Saliva", "Stool",
"Subgingival_plaque",
               "Supragingival_plaque", "Throat", "Tongue_dorsum", "Vagina")


for(bs in bodysites){
    axes <- read.table(file=paste(bs, ".tx.1.subsample.jsd.1.lt.nmds.axes", sep
=""), header=T, row.names=1)
    axes <- axes[order(rownames(axes)),]

    stress <- read.table(file=paste(bs, ".tx.1.subsample.jsd.1.lt.nmds.stress",
 sep=""), header=T)
    iter <- which.min(stress$Stress)
    stress.score <- format(stress[iter, "Stress"], digits=2)
    r.squared <- format(stress[iter, "Rsq"], digits=2)
    pdf(file=paste(bs,".ordination.pdf", sep=""), width=8.5, height=11)


    par(mfrow=c(2,1))
    par(mar=c(1, 4, 3, 1))
    dmm <- read.table(file=paste("../v35/dmm_best/", bs, ".tx.1.subsample.1.dmm
.mix.design", sep=""), row.names=1)
    rownames(dmm)==rownames(axes)

    ntypes <- length(levels(factor(dmm$V2)))
    clrs <- rainbow(ntypes)

    plot(axes, col=clrs[dmm$V2], pch=19, cex=0.75, xaxt="n", xlab="", ylab="NM
DS Axis 2", ylim=c(-0.7,0.7), xlim=c(-0.7,0.7))
    text(x=-0.75, y=0.68, "DMM", cex=1.5, font=2, pos=4)
    legend("bottomleft", legend=paste("Community Type", LETTERS[1:ntypes]),
           pch=19, col=clrs, pt.cex=1.5)
    mtext(side=3, line=1.25, text=gsub("_", " ", bs), at=0.7, adj=1, cex=1.5, f
ont=2)


    par(mar=c(4, 4, 0, 1))
    pam <- read.table(file=paste(bs, ".tx.1.subsample.pam.sw.enterotype", sep="
"), row.names=1)
    rownames(pam)==rownames(axes)
    ntypes <- length(levels(factor(pam$V2)))
    clrs <- rainbow(ntypes)

    plot(axes, col=clrs[pam$V2], pch=19, cex=0.75, ylab="NMDS Axis 2", xlab="N
MDS Axis 1", ylim=c(-0.7,0.7), xlim=c(-0.7,0.7))
    text(x=-0.75, y=0.68, "PAM (SI)", cex=1.5, font=2, pos=4)
    legend("bottomleft", legend=paste("Community Type", LETTERS[1:ntypes]), pch
=19, col=rainbow(ntypes), pt.cex=1.5)

    mtext(side=1, line=2, at=-0.9, text=paste("Stress: ", stress.score, "\nRsqu
ared: ", r.squared, sep=""), adj=0)
    dev.off()
}
```

Conclusions

We hope you've found this notebook useful at providing greater details into the methods we used to characterize the full HMP 16S rRNA gene sequencing dataset. Here are the take home points from the study and our methods development / application:

- Identifying community types is a useful way of categorizing community data at a bodysite and can be used as a tool to reduce the complexity of the data in order to associate these types with clinical metadata (e.g. whether one was breastfed, gender, level of education, etc.)
- Your mouth is connected to your rectum. You wouldn't believe the number of people (including one reviewer!) that were shocked to hear that the community type in your mouth is predictive of those in feces. Again, it is important to note that the bacterial taxa themselves are not necessarily the same, but the type of community you have in one site is predictive of others.
- Community types are not fixed and each type appears to have its own level of variation. Unfortunately, we're somewhat powerless to figure out why they change. Other studies suggest the same thing. For example Wu et al. 2012 () found two community types (using PAM) since they were interested in diet and couldn't find a short term result concluded that it was based on long term diet changes. We just don't know.
- We appreciate that the concept of community types has been controversial. We have demonstrated that applying the DMM-based approach is more robust than the previously-applied PAM-based approaches, which do not seem robust. This is especially the case when there is only one community type since it will not be detected by the current PAM methods.

Some things for the future:

- If we can bin people into community types we can begin to associate community types with cases of disease or risk of disease. This has already been done in a number of studies (see the paper for references).
- We need better/more temporal metadata that we can use to explain changes in community types
- We suspect that as we get more and more subjects we will see more and more community types. The types represent clusters of points. We know that each individual is more like themselves than anyone else. Therefore, if we sample more points from each person, we would expect them to form their own type.

Finally, if you find this set of methods useful, please let us know (pschloss@umich.edu)!

Postscript

The data analysis that was performed in the original study used a phylotype-based approach to characterize the structure of the human microbiome. We have also been interested in using the data to generate OTUs and mothur-based shared files. We did this as follows:

In [ ]:

```
mothur "#cluster.split(fasta=data/raw/v35.unique.good.filter.unique.precluster.

unique.pick.pick.fasta, name=data/raw/v35.unique.good.filter.unique.precluster.
unique.pick.pick.names, taxonomy=data/raw/v35.unique.good.filter.unique.preclus
ter.unique.pick.three.wang.pick.taxonomy, taxlevel=5, classic=T, processors=4,
cluster=T, outputdir=data/mothur/);
        make.shared(list=data/mothur/v35.unique.good.filter.unique.precluster.u
nique.pick.pick.an.list, group=data/raw/v35.good.pick.pick.groups, label=0.03,
outputdir=mothur);
        classify.otu(list=data/mothur/v35.unique.good.filter.unique.precluster.
unique.pick.pick.an.list, group=data/raw/v35.good.pick.pick.groups, name=data/r
aw/v35.unique.good.filter.unique.precluster.unique.pick.pick.names, taxonomy=da
ta/raw/v35.unique.good.filter.unique.precluster.unique.pick.three.wang.pick.tax
onomy, label=0.03, outputdir=data/mothur);
        get.oturep(fasta=data/raw/v35.unique.good.filter.unique.precluster.uniq
ue.pick.pick.fasta, name=data/raw/v35.unique.good.filter.unique.precluster.uniq
ue.pick.pick.names, list=data/mothur/v35.unique.good.filter.unique.precluster.u
nique.pick.pick.an.list, label=0.03, method=abundance);"
```

We then went back and ran these output files through the various scripts that we did above starting at the block staring "Recall above we gave the samples..."

Processing math: 100%